INTRODUCTION – CT 216 SOFTWARE ENGINEERING I

Dr. Enda Barrett
Enda.Barrett@universityofgalway.ie

Ollscoil na Gaillimhe
University of Galway

Module overview (subject to change)

- **Thursday IT 101 1-3 PM**

- Software Engineering (**Dr. Enda Barrett**) – S1 + S2
- Group project – (**Dr. Enda Barrett**) – S1 + S2

- Blackboard
  - Notes
  - Problem sheets
  - Assignment submission
  - Announcements

- Lab Tutors: Daniel Kelly
  - Labs start in a couple of weeks – **Friday 12 - 2PM IT 106**

Module Details

- Exam at the end of the year (Summer 2023)
  - No exam at the end of Semester 1
  - 4 questions answer 3

- Group project will account for **40%** of the final mark
  - Delivery of a spec – 5% - Group
  - Project Demo and Final Report (last week of term)
  - Graded holistically at the end

Semester 1 goal

- Learn about the software engineering principles and methods which enable the building of large team based software systems
- Understand the importance of version control and team based development
- Get cloud services experience and configure the deployment environment for your group based projects

Blackstone Launchpad

- Blackstone LaunchPad

Blackstone LaunchPad

Group Projects

- Groups of 4 people
  - Web based application i.e. You will build a **web application** using (HTML, CSS, JavaScript), you will build the backend in **Node.js**, deploy to Firebase and utilise a data storage component i.e. **Firestore**
  - Past projects included
    - Chat rooms
    - Personal Dashboards
    - Photo sharing application

  - Some difference between groups!
  - So get thinking about what you would like to do!
  - I wish to keep the project groups within the class splits during the face to face slots where possible.

Group project dates

- Please form your project groupings by

  Friday 23rd September at 17:00

- Nominated group lead should email me (Enda.Barrett@nuigalway.ie) the following
  - Team/Group name ("The coders"),
  - Names of each member,
  - Group sizes of 4 work best
  - Come up with an idea and mail it to me
    - real time event app
    - instant messenger
    - social media tool
  - If you don't have a group I will randomly assign you
- If you opt out you will get 0!

Web applications

- Clear separation of concerns
  - Frontend view code or UI (CSS, HTML)
    - Look and feel, structuring content
  - Frontend dynamic content (JavaScript, VueJS (potentially))
    - GET/POST methods, handling/updating data
  - Backend server side code (Node.js) – **Firebase functions**
    - Returning data, developing APIs
  - DB component (**Firestore**)
    - Schemas, queries for document retrieval
- Basic app up and running by Christmas deployed using Firebase.

CT216 SOFTWARE ENGINEERING 1

CLOUD COMPUTING

Dr. Enda Barrett
Enda.Barrett@universityofgalway.ie

Ollscoil na Gaillimhe
University of Galway

What is cloud computing?

- "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." (NIST, May 2011)

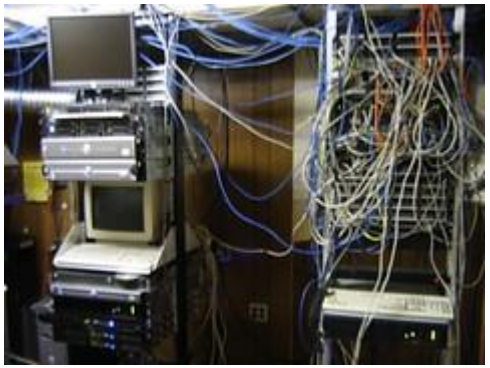
What is cloud computing

- Back in the early 00's there was this bookseller called Amazon, who made their money by shipping books (and everything else) around the world.
- To support their web app (Amazon.com) they had built up some neat hosting infrastructure and software to manage it at scale with a couple of data centers
- The centers were somewhat underutilised and Amazon decided to start selling this spare capacity.

Server room

- At that time most businesses/organisations maintained a server room on premises. In it they would have separate rack mounted PCs/servers.
  - Data would be stored on large Storage Networks
  - Backups would be run on this data
  - Multiple machines (servers) would run business critical software

- There were challenges to this
  - Maintenance
  - Upgrading machines
  - Upfront purchasing costs
  - Hire staff to manage it (Sysadmins)



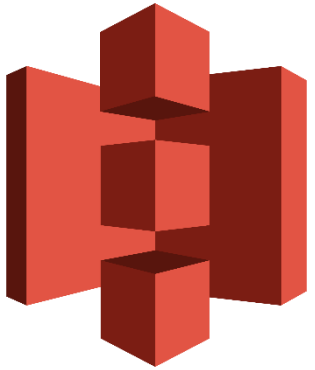
What is cloud computing

- Started with storage
  - Simple Storage Service launched on **March 14 2006** marking the beginning of Amazon Web Services
  - This allowed users to store documents, files, data on an S3 bucket without having to manage, purchase, maintain the underlying disk hardware.

What is cloud computing

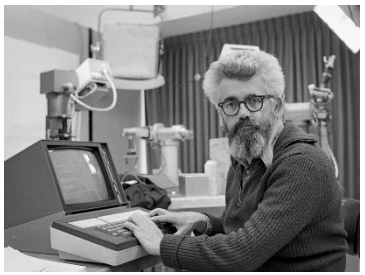
- Then came computing
  - It followed up its successful storage launch of S3 with EC2 or Elastic Compute Cloud in August 2006
  - This allowed you to have access to a remote server accessible via the internet!

Demand was strong…



History of cloud computing

- Computing may someday be organized as a public utility just as the telephone system is a public utility," Professor John McCarthy said at MIT's centennial celebration in 1961. "Each subscriber needs to pay only for the capacity he actually uses, but he has access to all programming languages characteristic of a very large system … Certain subscribers might offer service to other subscribers … The computer utility could become the basis of a new and important industry."



https://www.technologyreview.com/2011/10/03/190237/the-cloud-imperative/

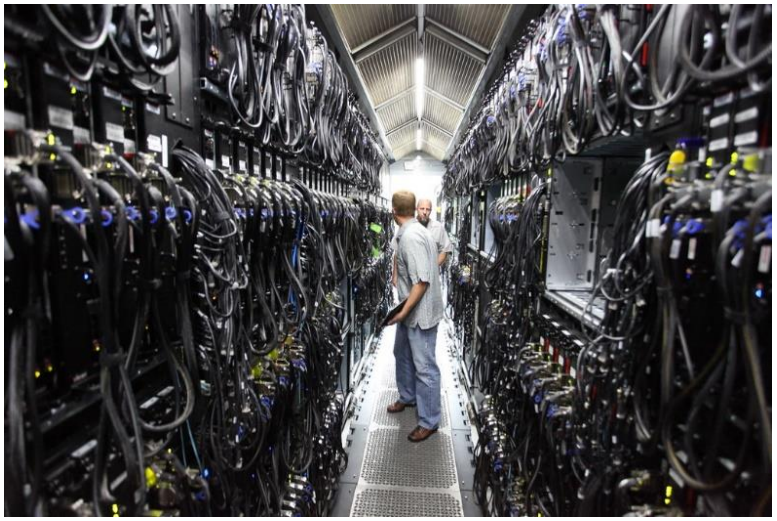
Cloud computing growth

- The growth in cloud computing over the past decade has been phenomenal.
- In April 2011, Forrester projected that it would be worth $160bn dollars by 2020, in reality it was 27% larger at $219bn.
- In 2022 it hit over $480bn in value and is projected to exceed $1tn by 2029.

https://www.fortunebusinessinsights.com/cloud-computing-market-102697

Where is the cloud?

Amazon US-East N. Virginia



Cloud types – Public cloud

- Amazon, MS Azure, Google Cloud are examples of public clouds.
- Any member of the public can sign up and start provisioning compute resources within minutes
- They are highly scalable and allow an organisation to grow its infrastructure rapidly

Cloud types – Private cloud

- Private cloud
  - Computing resources are dedicated to a single customer and not shared with other customers. Considered to be more secure.
  - AWS do offer a virtual private cloud
  - https://aws.amazon.com/vpc/
  - Organisations can also host their on cloud on-prem using software such as OpenStack.

Cloud types - Hybrid

- Finally a hybrid cloud is simply a mix of public cloud resources and private resources. An organisation may choose this option if there is a mixture in the criticality of their data or computational requirements.
- Data that doesn't require heightened security can be pushed on to the public cloud whereas that which does can be hosted on the private cloud.

Cloud services

- **Software as a Service** (SaaS) - provides users with—essentially—a cloud application, the platform on which it runs, and the platform's underlying infrastructure.
- **Platform as a Service** (PaaS) - provides users with a platform on which applications can run, as well as all the IT infrastructure required for it to run.
- **Infrastructure as a Service** (IaaS) - provides users with compute, networking, and storage resources.

Virtual Servers

- Infrastructure as a Service (IaaS)
  - Amazon, Google, Microsoft
- Create virtual machines
  - t1.micro, m1.small, c1.medium, m1.large…
- Customise instances and add greater amounts of storage.
- Each instance can be booted up with a different AMI, you can even create your own!
- Xen Hypervisor (Sun, AMD, IBM, Dell, Intel)
- Storage area networks provide the storage

Advantages/Disadvantages of cloud computing

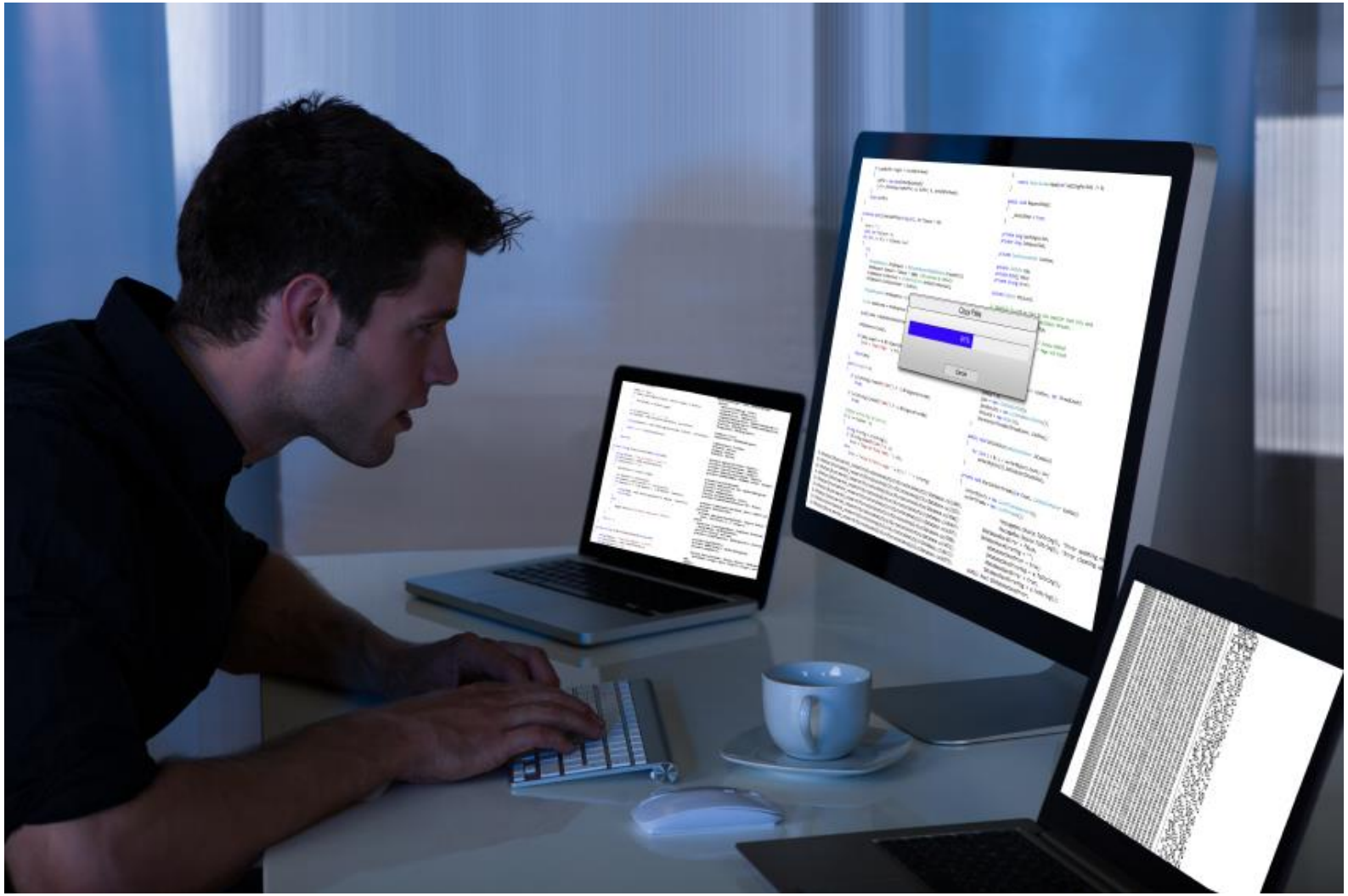
- When compared to hosting in-house cloud computing has a number of benefits
  - Elasticity – if your application becomes very popular you can procure new resources in minutes
  - Reduced capital expenditure
  - Economies of scale
- There are also some drawbacks
  - Security/privacy
  - Cost
  - Migration issues

SOFTWARE PROCESSES

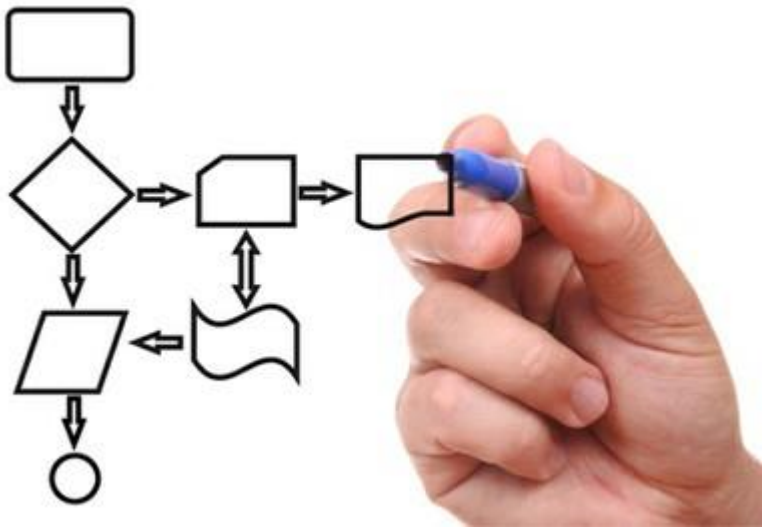
Dr. Enda Barrett

OÉ Gaillimh
NUI Galway

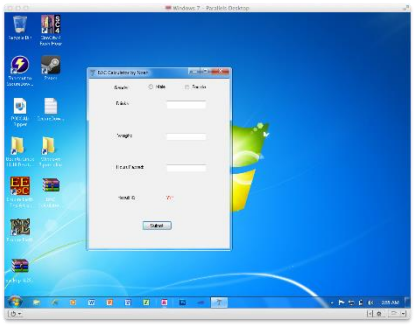
A Software Process: Who is like this?



Is there anyone like this?

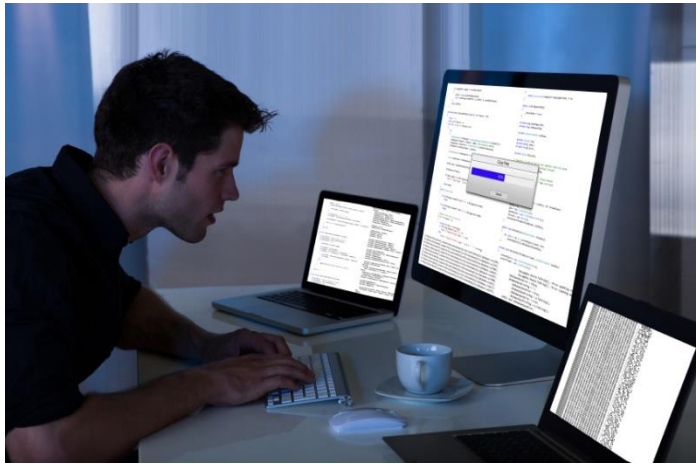


Recap: Software Dev. is complex and varied

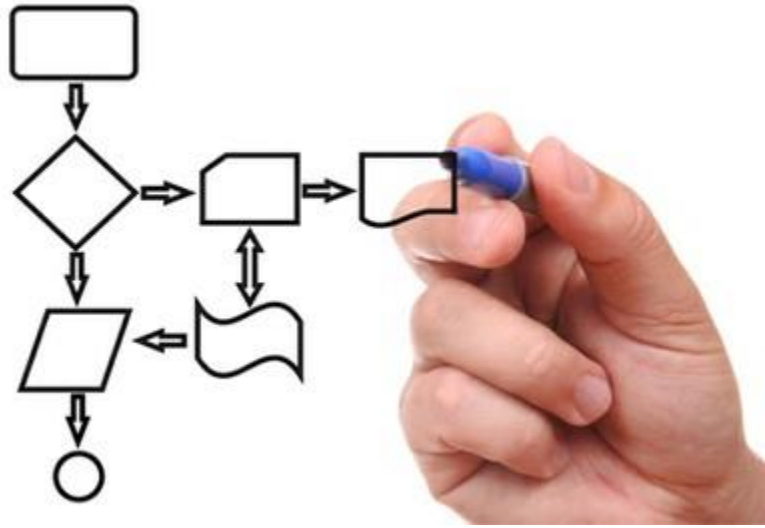


- Ada
- 3 levels of redundancy
- Different dev teams

Difference between these two?



vs



Bad Process

Bad Engineer

Good Process

Good Engineer

Building a house

- Plan
  - Sketch the layout/structure
  - Determine how the components will fit

- Construction
  - Laying foundations/block laying/engineer testing

- Deployment
  - Delivered to the customer who provides feedbac
    list

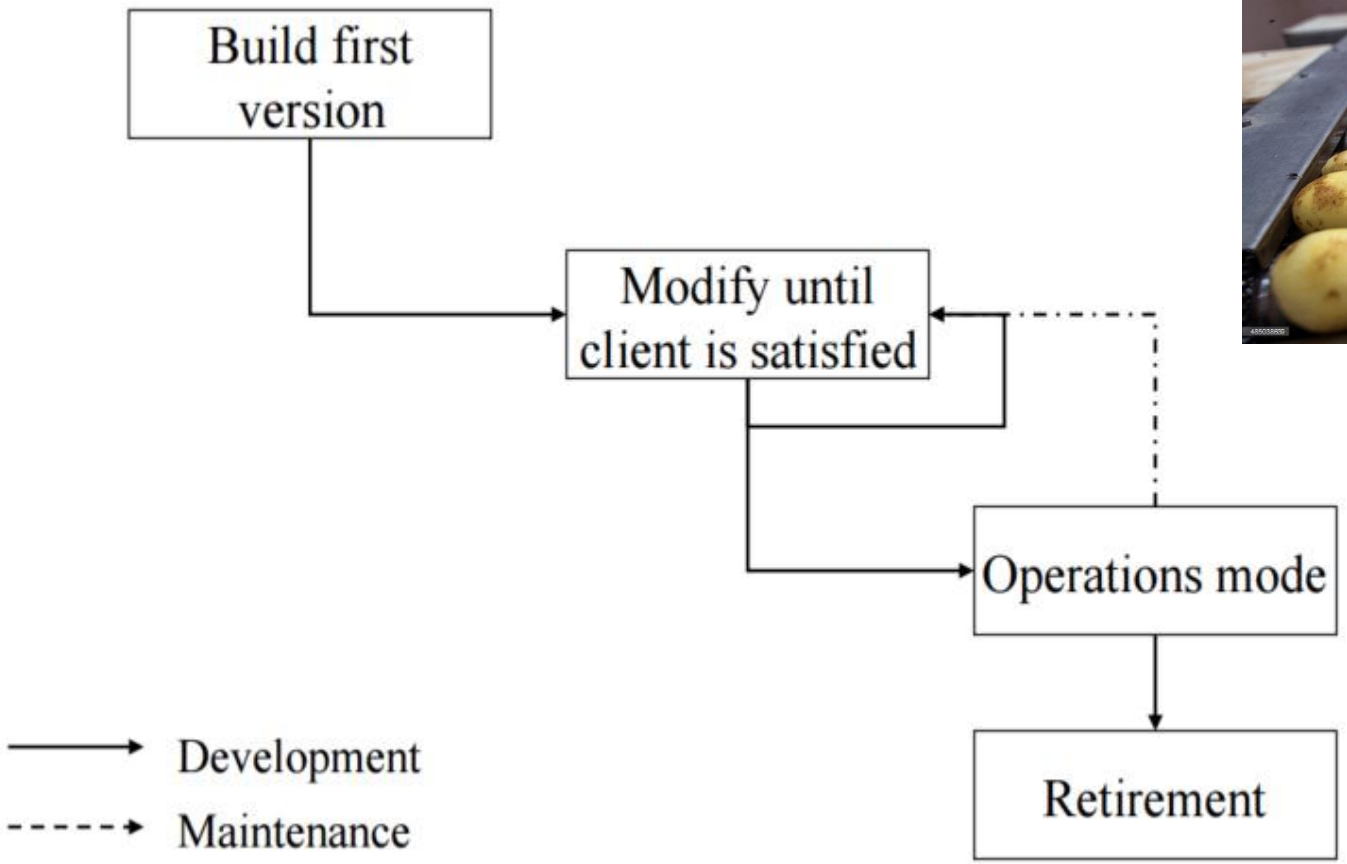
The software process

- A structured set of activities required to develop a software system
- Four fundamental process activities
  - Specification
  - Development
  - Validation
  - Evolution
- The foundation of software engineering is in the **process**
- **Goal:** To efficiently and predictably deliver a product that meets the requirements

Motivating case

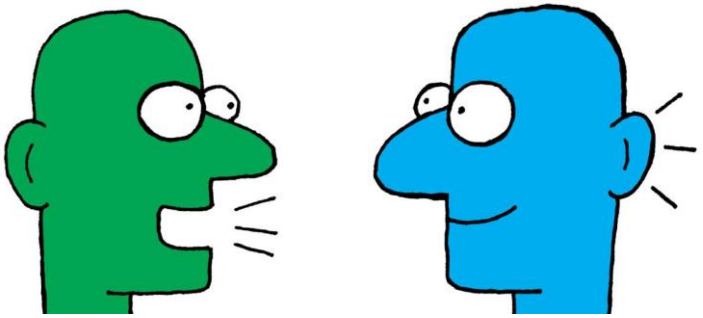
- You've been hired by a local independent retailer to build their potato peeling system

Build and fix model…worst approach



Software Process Model

- 1) Software Specification
  - Talk to the customer
  - Understand the problems
  - Talk to any relevant stakeholders



- 2) Software Development
  - Map out the tasks
  - Design the software
  - Develop the solution

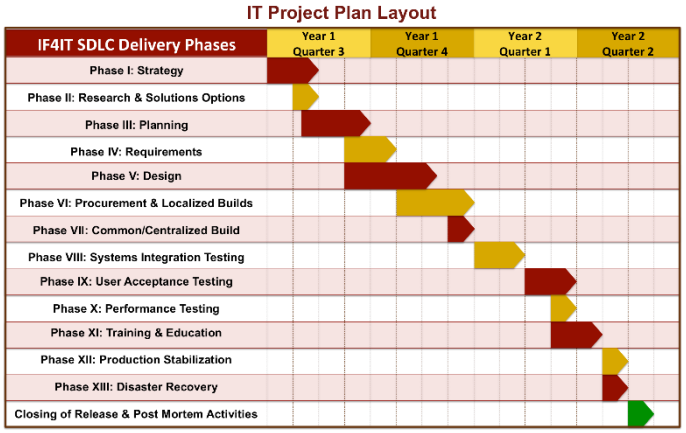


Figure: Example Layout of SDLC Phases as Key IT Project Milestones

Software Process Model…

- 3) Software validation
  - Does it meet requirements
  - Is it what the customer wanted

- 4) Software evolution (maintenance)
  - Modified to adapt
  - Changes in requirements
  - Customer & Market conditions

Software Engineering Practice

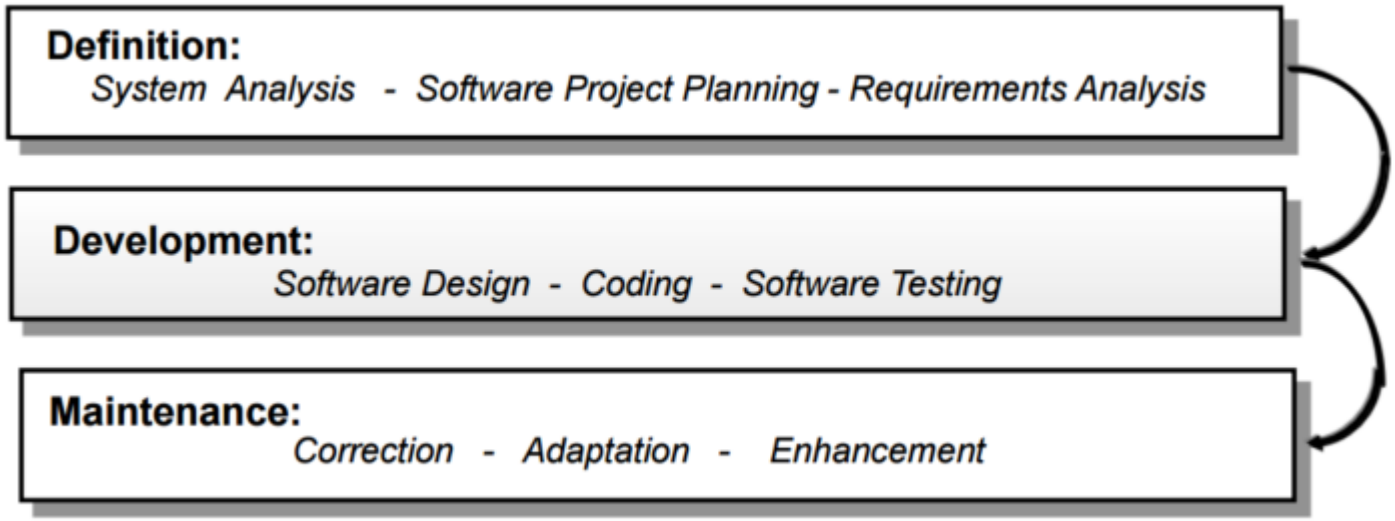
- 1) **Understand the problem** (*Communication and analysis*)
  - Who are the stakeholders?
  - What are the unknowns?
- 2) **Plan the solution** (*Modelling and software design*)
  - Have we seen this problem before?
  - Has a similar problem been already solved? Plagiarism
  - Can sub problems be found?
- 3) **Carry out the plan** (*Write the code*)
  - Does the solution conform to the plan?
  - Has the code been reviewed for correctness?
- 4) **Examine the result** (*Test it*)
  - Is each component testable?
  - Does the solution produce results as defined originally?

General Software Engineering Questions

- **What** is the problem to be solved?
  - Requirements definition
- **What** are the characteristics of the software (system) used to solve the problem?
  - Analysis
- **How** will the system be realised/constructed?
  - Design
- **How** will design and construction errors be uncovered and dealt with?
  - Test
- **How** will the system be supported long-term?
  - Maintenance

Overview of Software Engineering

- There are three generic phases, regardless of paradigm:
  - Definition, **a focus on the *What*.**
  - Development, **a focus on the *How*.**
  - *Maintenance,* ***focuses on Change***



Software Engineering should…

- Provide a clear statement of the project mandate & objectives;
- Create effective means of communication;
- Increase user involvement & ownership;
- Provide an effective management framework to support productivity & pragmatism;
- Establish quality assurance procedures;
- Provide sound resource estimation and allocation procedures;
- Ensure the effectiveness and durability of systems produced;
- Encourage the re-usability of code and/or solutions;
- Reduce the organisation's vulnerability to the loss of software development personnel (MJ);
- Reduce and support post implementation maintenance of systems;

CT216 SOFTWARE ENGINEERING 1

CLOUD COMPUTING

Dr. Enda Barrett
Enda.Barrett@universityofgalway.ie

Ollscoil na Gaillimhe
University of Galway

What is cloud computing?

- "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." (NIST, May 2011)

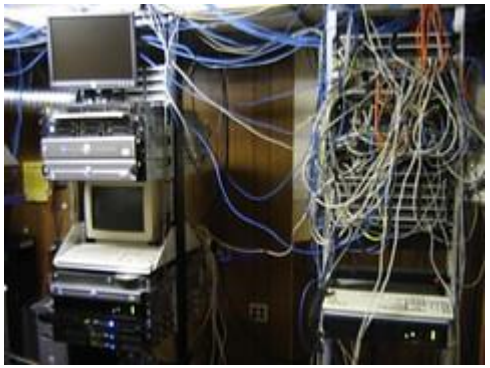
What is cloud computing

- Back in the early 00's there was this bookseller called Amazon, who made their money by shipping books (and everything else) around the world.
- To support their web app (Amazon.com) they had built up some neat hosting infrastructure and software to manage it at scale with a couple of data centers
- The centers were somewhat underutilised and Amazon decided to start selling this spare capacity.

Server room

- At that time most businesses/organisations maintained a server room on premises. In it they would have separate rack mounted PCs/servers.
  - Data would be stored on large Storage Networks
  - Backups would be run on this data
  - Multiple machines (servers) would run business critical software

- There were challenges to this
  - Maintenance
  - Upgrading machines
  - Upfront purchasing costs
  - Hire staff to manage it (Sysadmins)



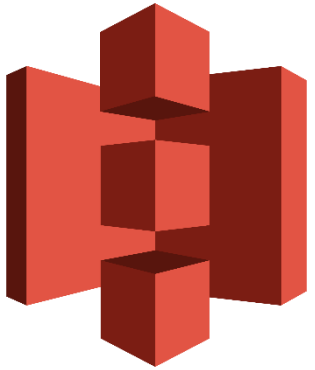
What is cloud computing

- Started with storage
  - Simple Storage Service launched on **March 14 2006** marking the beginning of Amazon Web Services
  - This allowed users to store documents, files, data on an S3 bucket without having to manage, purchase, maintain the underlying disk hardware.

What is cloud computing

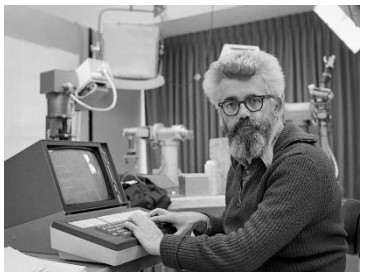
- Then came computing
  - It followed up its successful storage launch of S3 with EC2 or Elastic Compute Cloud in August 2006
  - This allowed you to have access to a remote server accessible via the internet!

Demand was strong…



History of cloud computing

- Computing may someday be organized as a public utility just as the telephone system is a public utility," Professor John McCarthy said at MIT's centennial celebration in 1961. "Each subscriber needs to pay only for the capacity he actually uses, but he has access to all programming languages characteristic of a very large system … Certain subscribers might offer service to other subscribers … The computer utility could become the basis of a new and important industry."



https://www.technologyreview.com/2011/10/03/190237/the-cloud-imperative/

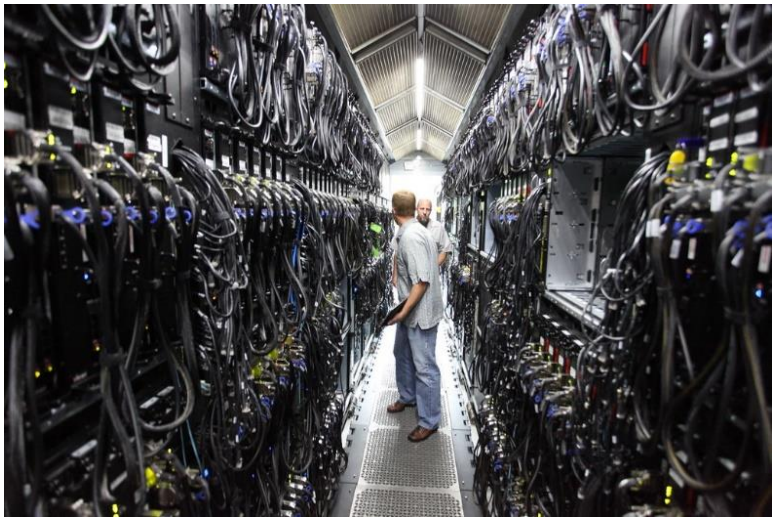
Cloud computing growth

- The growth in cloud computing over the past decade has been phenomenal.
- In April 2011, Forrester projected that it would be worth $160bn dollars by 2020, in reality it was 27% larger at $219bn.
- In 2022 it hit over $480bn in value and is projected to exceed $1tn by 2029.

https://www.fortunebusinessinsights.com/cloud-computing-market-102697

Where is the cloud?

Amazon US-East N. Virginia



Cloud types – Public cloud

- Amazon, MS Azure, Google Cloud are examples of public clouds.
- Any member of the public can sign up and start provisioning compute resources within minutes
- They are highly scalable and allow an organisation to grow its infrastructure rapidly

Cloud types – Private cloud

- Private cloud
  - Computing resources are dedicated to a single customer and not shared with other customers. Considered to be more secure.
  - AWS do offer a virtual private cloud
  - https://aws.amazon.com/vpc/
  - Organisations can also host their on cloud on-prem using software such as OpenStack.

Cloud types - Hybrid

- Finally a hybrid cloud is simply a mix of public cloud resources and private resources. An organisation may choose this option if there is a mixture in the criticality of their data or computational requirements.
- Data that doesn't require heightened security can be pushed on to the public cloud whereas that which does can be hosted on the private cloud.

Cloud services

- **Software as a Service** (SaaS) - provides users with—essentially—a cloud application, the platform on which it runs, and the platform's underlying infrastructure.
- **Platform as a Service** (PaaS) - provides users with a platform on which applications can run, as well as all the IT infrastructure required for it to run.
- **Infrastructure as a Service** (IaaS) - provides users with compute, networking, and storage resources.

Virtual Servers

- Infrastructure as a Service (IaaS)
  - Amazon, Google, Microsoft
- Create virtual machines
  - t1.micro, m1.small, c1.medium, m1.large…
- Customise instances and add greater amounts of storage.
- Each instance can be booted up with a different AMI, you can even create your own!
- Xen Hypervisor (Sun, AMD, IBM, Dell, Intel)
- Storage area networks provide the storage

Advantages/Disadvantages of cloud computing

- When compared to hosting in-house cloud computing has a number of benefits
  - Elasticity – if your application becomes very popular you can procure new resources in minutes
  - Reduced capital expenditure
  - Economies of scale
- There are also some drawbacks
  - Security/privacy
  - Cost
  - Migration issues

SCRUM – ROLES AND CEREMONIES

Dr. Enda Barrett

OÉ Gaillimh
NUI Galway

Scrum Framework

Roles

- Product owner
- ScrumMaster
- Team

Ceremonies

- Sprints
- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

Roles

- Product owner
- ScrumMaster
- Team

Ceremonies

- Sprints
- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

Product Owner

- Define the features of the product
  - Tries to remove conjecture, “I know the customer wants this” as opposed to “I believe this would be a good feature”
- Decide on release date and content
  - Usually responsible for press releases
- Be responsible for the profitability of the product (ROI)
- Prioritize features according to market value
  - Conduct market research, feasibility studies
- Adjust features and priority every iteration, as needed
- Accept or reject work results.

Scrum Master

- Represents management to the project
  - Often one of the engineers
- Responsible for enacting Scrum values and practices
- Removes impediments
- Ensure that the team is fully functional and productive
- Enable close cooperation across all roles and functions
- Shield the team from external interferences

Scrum Team

- Typically 5-10 people
- Cross-functional
- QA, Programmers, UI Designers, etc.
- Members should be full-time
- May be exceptions (e.g., System Admin, etc.)
- Teams are self-organizing
- Membership can change only between sprints

Roles

- Product owner
- ScrumMaster
- Team

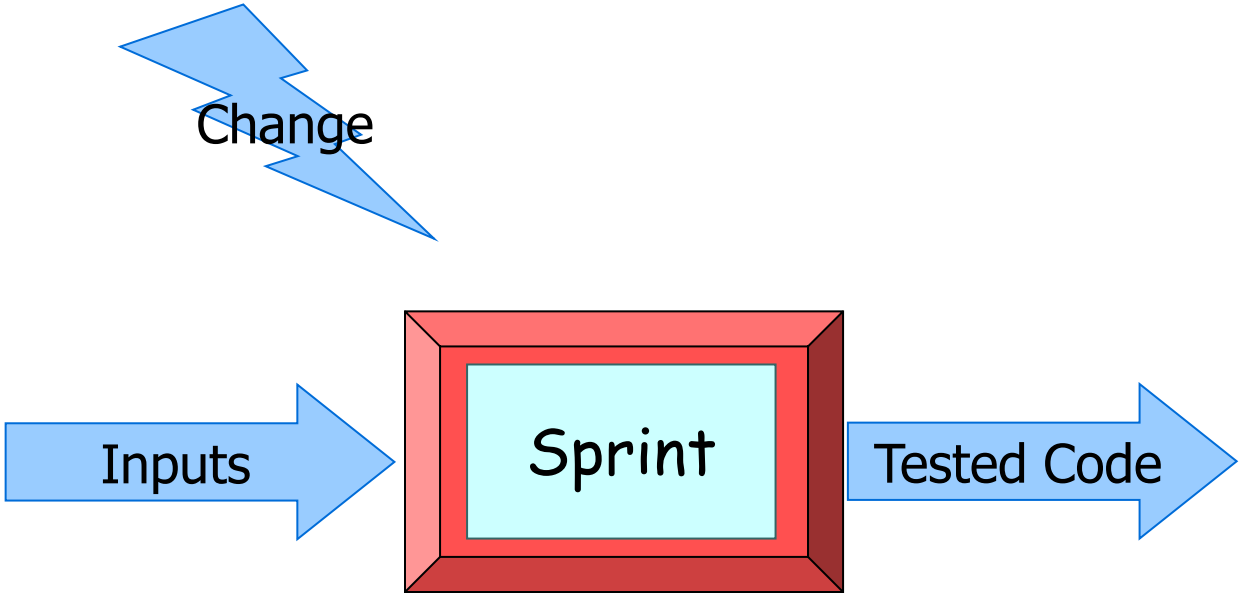
Ceremonies

- Sprints
- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

Sprints

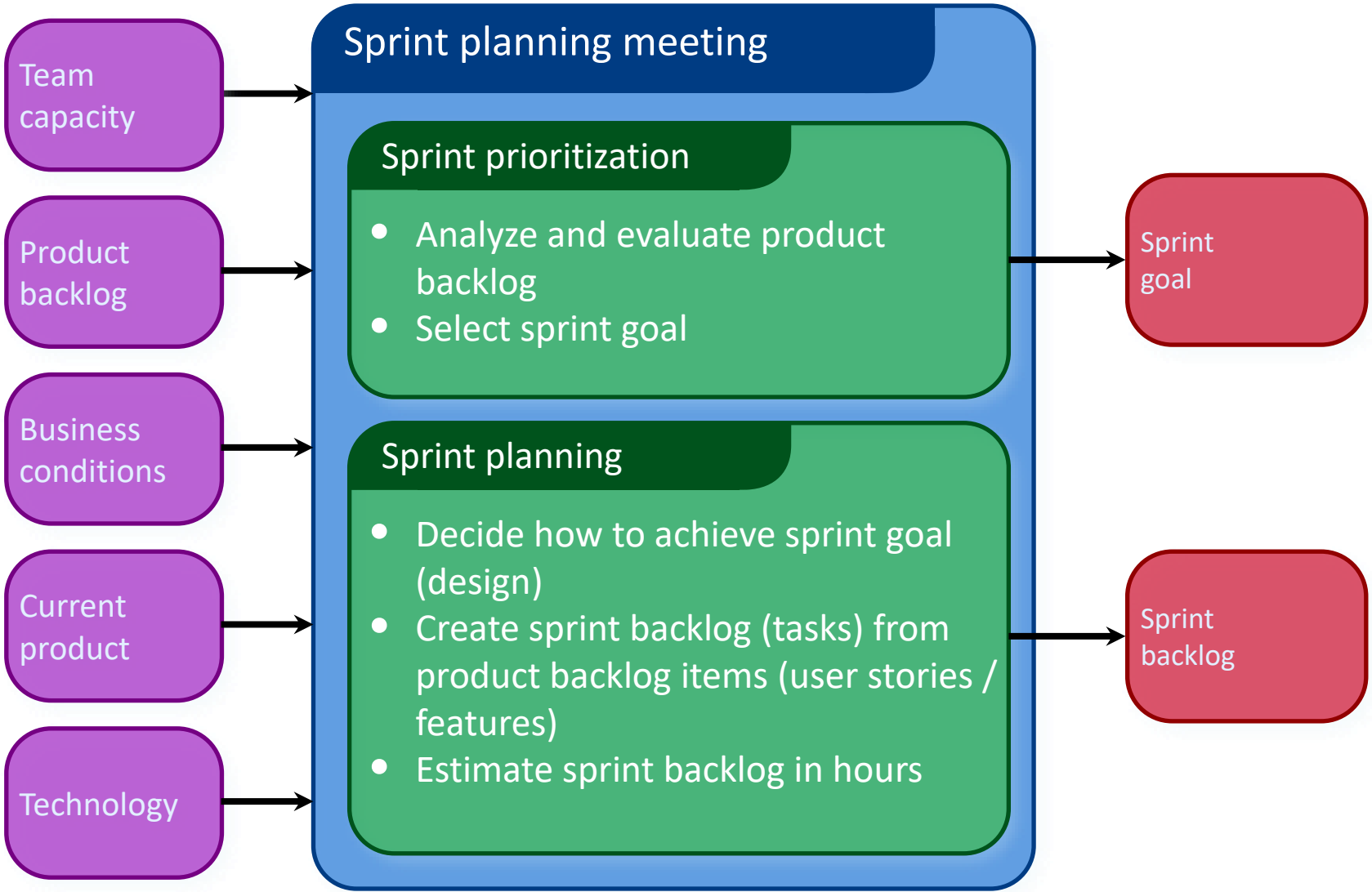
- Scrum projects make progress in a series of "sprints"
- Target duration is one month
  - +/- a week or two (2 - 6 weeks max)
    - But, a constant duration leads to a better rhythm
- Product is designed, coded, and tested during the sprint
  - The output is a build which may or may not be a release
- Move onto the next sprint…

No changes during sprint



- Plan sprint durations around how long you can commit to keeping change out of the sprint

Sprint planning



Sprint planning

- Team selects items from the product backlog that they can commit to completing
- Sprint backlog is created
  - Tasks are identified and each is estimated (1-16 hrs)
  - Team collaboratively does this

As a vacation planner, I want to see photos of the hotels.

Code the middle tier (8 hours)
Code the user interface (4)
Write test fixtures (4)
Code the foo class (6)
Update performance tests (4)

What is the Product Backlog

- A list of all desired work on the project (the requirements)
  - Usually a combination of
    - story-based work ("let user search and replace")
    - task-based work ("improve exception handling")
- List is prioritized by the Product Owner
  - Typically a Product Manager, Marketing, Internal Customer, etc.
  - Priority groupings (high, medium, low … etc)
  - Reprioritised at the start of each sprint
  - Spreadsheet (usually)

To create a Sprint Backlog you must have a (Sprint) goal

Life Sciences

Support features necessary for population genetics studies.

Database Application

Make the application run on SQL Server in addition to Oracle.

Financial services

Support more technical indicators than company ABC with real-time, streaming data.

From Sprint Goal to Sprint Backlog

- Scrum team takes the Sprint Goal and decides what tasks are necessary
- Team self-organizes around how they will meet the Sprint Goal
  - Manager does not assign tasks to individuals
- Managers don't make decisions for the team

- Sprint Backlog is created

Sprint backlogs during the sprint

- Changes
  - Team adds new tasks whenever they need to, in order to meet the Sprint Goal
  - Team can remove unnecessary tasks
  - But: Sprint Backlog can only be updated by the team

- Estimates are updated whenever there's new information

Roles

- Product owner
- ScrumMaster
- Team

Ceremonies

- ~~Sprints~~
- ~~Sprint planning~~
- Sprint review
- Sprint retrospective
- ~~Daily scrum meeting~~

Sprint review meeting

- Team presents what it accomplished during the sprint
- Typically takes the form of a demo of new features or underlying architecture
- Informal
  - Two hour prep time
  - No slides
- Participants
  - Customers
  - Management
  - Product Owners
  - Engineering team

Sprint Retrospective meetings

- Typically 15–30 minutes
- Done after every sprint
- Feedback meeting – time to reflect on how things are going…

- Many participants
  - ScrumMaster
  - Product owner
  - Team
  - Possibly customers and others

Start/Stop/Continue

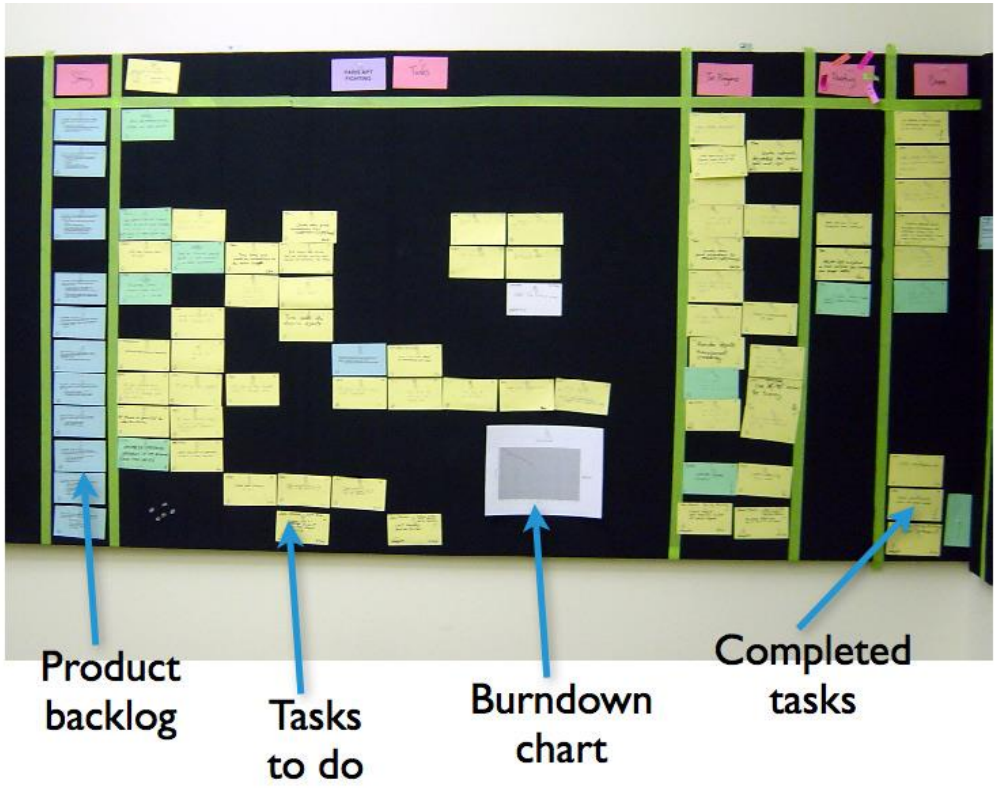
Whole team gathers and discusses what they'd like to:

Start doing

Stop doing

Continue doing

War room


Product backlog
Tasks to do
Burndown chart
Completed tasks

Pros/Cons of Agile Methods

- *Advantages*
  - *Completely developed and tested features in short iterations*
  - *Simplicity of the process*
  - *Clearly defined rules*
  - *Increasing productivity*
  - *Self-organizing*
  - *Each team member carries a lot of responsibility*
  - *Improved communication*
  - *Combination with Extreme Programming*
- *Drawbacks*
  - *"Undisciplined hacking" (no written documentation)*
  - *Violation of responsibility*
  - *Current mainly carried by the inventors*
  - *Employee Burnout & Fatigue.*

SOFTWARE DEVELOPMENT PARADIGMS –AGILE METHODS

Dr. Enda Barrett

OÉ Gaillimh
NUI Galway

Agile software development

- What is agile software development?
  - Scrum – Software Project Management Methodology
  - XP – Software Development Methodology

Software Development Lifecycle

- The software lifecycle is an abstract representation of a software process. It defines the steps, methods, tools, activities and deliverables of a software development project. The following **lifecycle phases** are considered:
  - 1. requirements analysis
  - 2. system design
  - 3. implementation
  - 4. integration and deployment
  - 5. operation and maintenance

SDLC Limitations

- Classical project planning methods have a lot of disadvantages
  - Huge efforts during the planning phase {Requirements + Design}
  - Poor requirements conversion in a rapidly changing environment
  - Treatment of staff as a factor of production

Agile Man

Agile Motivations

- Agile proponents argue:
  - Software development processes relying on lifecycle models are too heavyweight or cumbersome
  - Too many things are done that are not directly related to the software product being produced, i.e. design, models, requirements docs, documentation that isn't shipped as part of the product
  - Difficulty with incomplete or changing requirements
  - Short development cycles (Mobile Apps)
  - More active customer involvement needed

What is Agile?

- Agile methods focus on
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan
- Several agile methods
  - No single agile method
  - Scrum
  - XP
- No single definition
- Agile Manifesto closest to a definition
  - Set of principles
  - Developed by Agile Alliance (http://www.agilealliance.org/)

Agile methods

- Agile methods:
  - Scrum
  - Extreme Programming (XP)
    - Continuous Integration
    - Test Driven Development (TDD)
    - …
- Agile Alliance (www.**agilealliance**.org)
  - A non-profit organization promotes agile development

Scrum in 100 words

- Scrum is an agile project management methodology for managing product development.
- It allows us to rapidly and repeatedly inspect actual working software (every two weeks to one month).
- The business sets the priorities. The teams self-manage to determine the best way to deliver the highest priority features.
- Every two weeks to a month anyone can see real working software and decide to release it as is or continue to enhance for another iteration.

History of Scrum

- 1995:
  - Analysis of common software development processes found that they are not suitable for unpredictable and non-repeatable processes
  - Design of a new method: Scrum by Jeff Sutherland & Ken Schwaber
  - Enhancement of Scrum by Mike Beedle & combination of Scrum with Extreme Programming
- 1996:
  - Introduction of Scrum at the (Object-Oriented Programming, Systems, Languages & Applications) OOPSLA conference
- 2001:
  - Publication “Agile Software Development with Scrum” by Ken Schwaber & Mike Beedle
  - Gained in popularity steadily ever since
- Founders are members in the Agile Alliance

Characteristics of Scrum

- Self-organizing teams
  - No need for project manager (in-theory)
- Product progresses in a series of month-long "sprints"…could be biweekly also
- Assumes that the software cannot be well defined and requirements will change frequently
- Requirements are captured as items in a list of "product backlog"
- No specific engineering practices prescribed
  - XP, TDD, FDD…
- Best approach is to start with Scrum and then invent your own version using XP, TDD, FDD

Daily Scrum/Standup

- Parameters
  - Daily
  - 15-minutes
  - Stand-up
  - Not for problem solving
  - Only team members, ScrumMaster, Product Owners should talk
  - Should help to avoid additional unnecessary meetings
  - Commitment in front of peers to complete tasks

Answer three questions

1. What did you do yesterday?
2. What will you do today?
3. Is anything in your way?

Daily SCRUM/Standup

- Is NOT a problem solving session
- Is NOT a way to collect information about WHO is behind the schedule
- Is a meeting in which team members make commitments to each other and to the Scrum Master
- Is a good way for a Scrum Master to track the progress of the team

AGILE METHODS – EXTREME PROGRAMMING

Dr. Enda Barrett

OÉ Gaillimh
NUI Galway

Overview

- **XP**
  - **General concepts**
- Specific XP concepts
  - Pair programming
  - Test Driven Development
  - Continuous Integration

Scrum Summary

- Scrum is a **project management** methodology

Characteristics of Scrum

- Self-organizing teams
  - No need for project manager (in-theory)
- Product progresses in a series of month-long "sprints"…could be biweekly also
- Assumes that the software cannot be well defined and requirements will change frequently
- Requirements are captured as items in a list of "product backlog"
- No specific engineering practices prescribed
  - XP, TDD, FDD…
- Best approach is to start with Scrum and then invent your own version using XP, TDD, FDD

Daily Scrum/Standup

- Parameters
  - Daily
  - 15-minutes
  - Stand-up
  - Not for problem solving
  - Only team members, ScrumMaster, Product Owners should talk
  - Should help to avoid additional unnecessary meetings
  - Commitment in front of peers to complete tasks

Answer three questions

1. What did you do yesterday?
2. What will you do today?
3. Is anything in your way?

Daily SCRUM/Standup

- Is NOT a problem solving session
- Is NOT a way to collect information about WHO is behind the schedule
- Is a meeting in which team members make commitments to each other and to the Scrum Master
- Is a good way for a Scrum Master to track the progress of the team

Scrum Framework

Roles

- Product owner
- ScrumMaster
- Team

Ceremonies

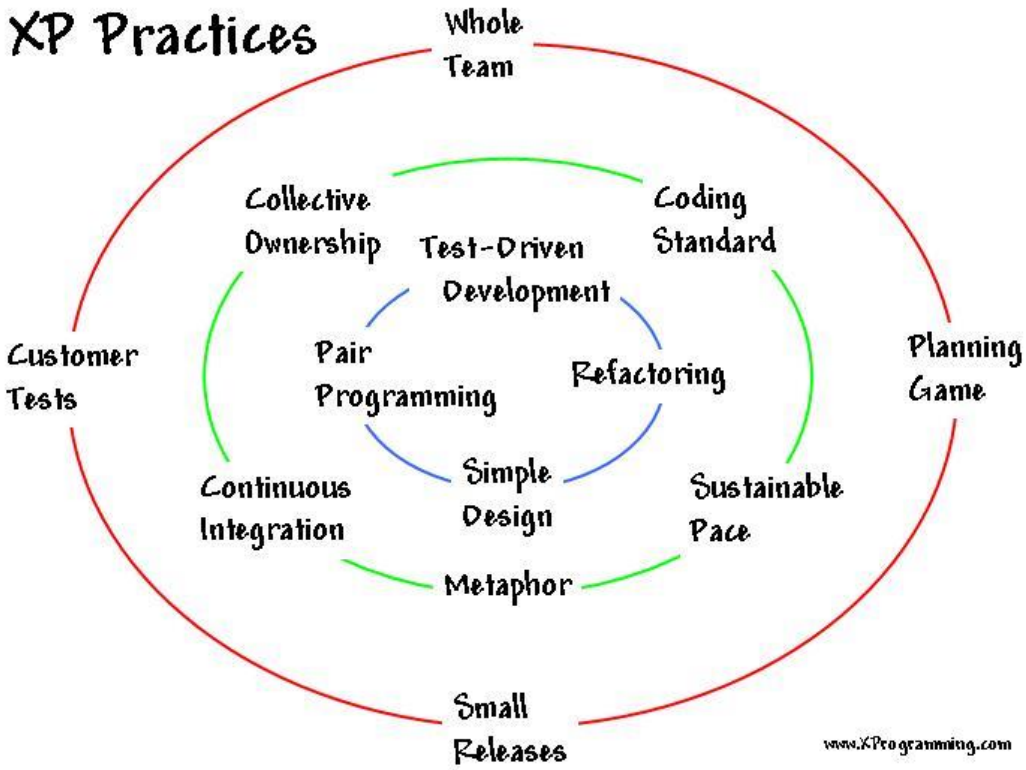
- Sprints
- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

We need an Agile Development method

- eXtreme Programming (XP)
  - One of the most popular agile **software development** methods

eXtreme Programming

- Pair programming
- Refactoring
- Test Driven Development
- Continuous Integration
- Metaphor
- Small releases
- Simple Design
- Customer tests

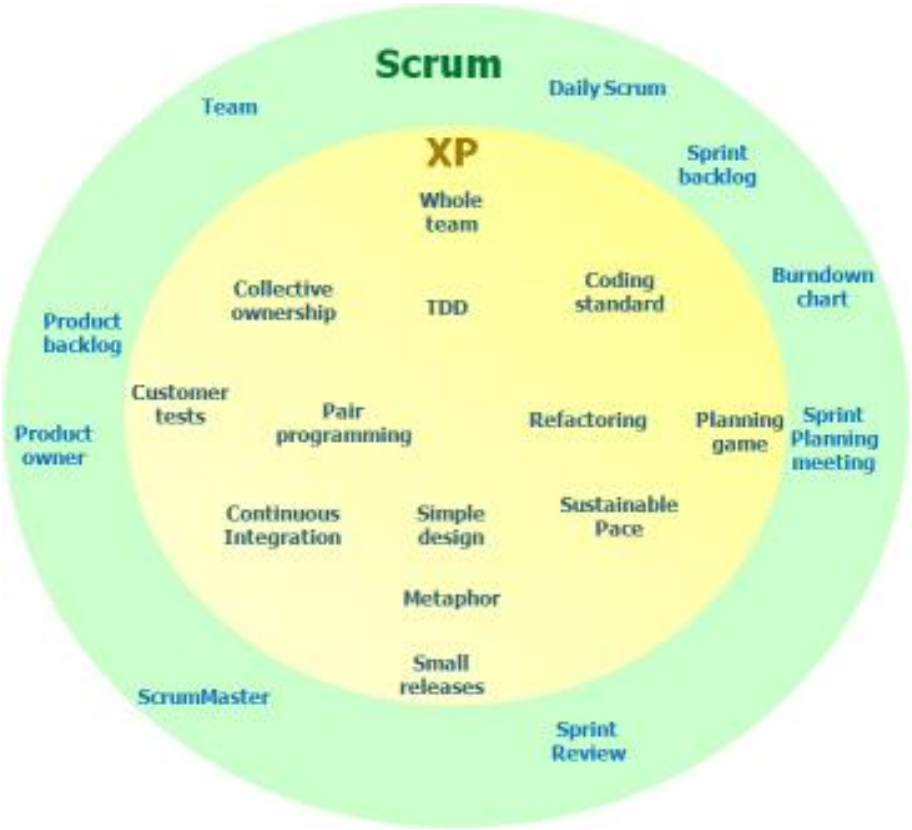


Complete Agile Process

Scrum + XP

Overview

How Scrum and XP can work together



Overview

- **XP**
  - **General concepts**
- Specific XP concepts
  - Pair programming
  - Test Driven Development
  - Continuous Integration

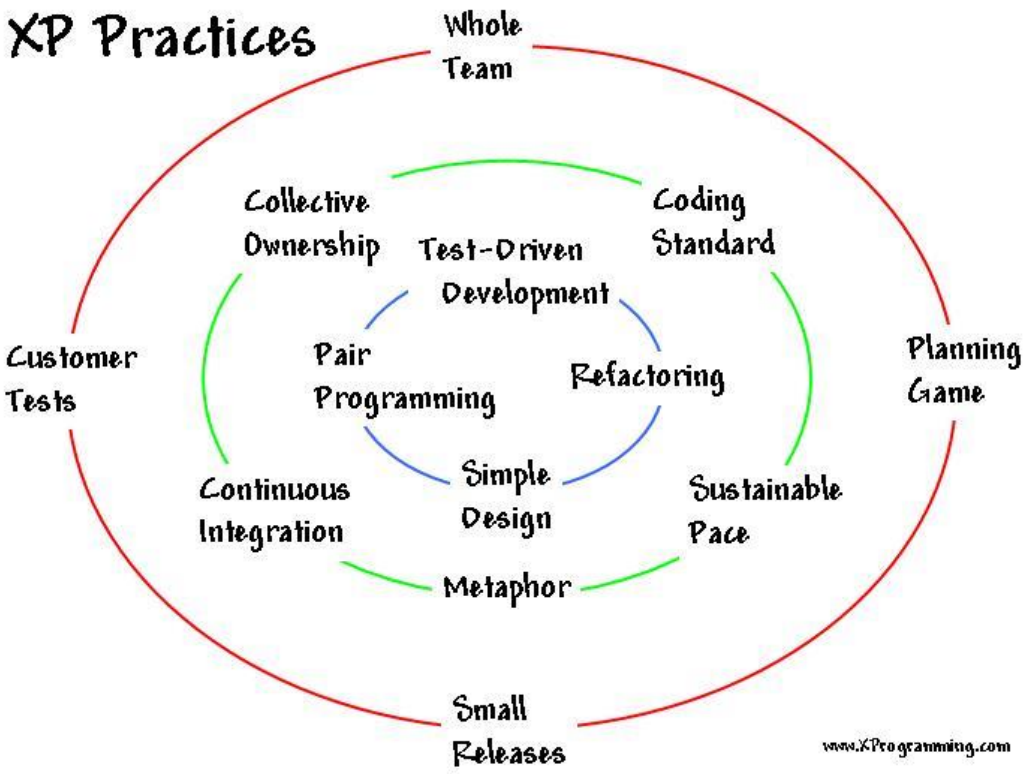
Principles of XP

Kent Beck

- **Communication**
  - Software development is inherently a team sport that relies on communication to transfer knowledge from one team member to everyone else on the team. XP stresses the importance of the appropriate kind of communication – face to face discussion with the aid of a white board or other drawing mechanism.
- **Simplicity**
  - Simplicity means "what is the simplest thing that will work?" The purpose of this is to avoid waste and do only absolutely necessary things such as keep the design of the system as simple as possible so that it is easier to maintain, support, and revise. Simplicity also means address only the requirements that you know about; don't try to predict the future.
- **Feedback**
  - Through constant feedback about their previous efforts, teams can identify areas for improvement and revise their practices. Feedback also supports simple design. Your team builds something, gathers feedback on your design and implementation, and then adjust your product going forward.
- **Courage**
  - Kent Beck defined courage as "effective action in the face of fear" (Extreme Programming Explained P. 20). This definition shows a preference for action based on other principles so that the results aren't harmful to the team. You need courage to raise organizational issues that reduce your team's effectiveness. You need courage to stop doing something that doesn't work and try something else. You need courage to accept and act on feedback, even when it's difficult to accept.
- **Respect**
  - The members of your team need to respect each other in order to communicate with each other, provide and accept feedback that honors your relationship, and to work together to identify simple designs and solutions.

Source:https://www.agilealliance.org/glossary/xp

Practices of XP



Further Reading and Descriptions

http://ronjeffries.com/xprog/what-is-extreme-programming/.

Whole Team

- All the contributors to an XP project sit together, members of one team. This team must include a business representative (Product Owner) – the "Customer" – who provides the requirements, sets the priorities, and steers the project.

https://ronjeffries.com/xprog/what-is-extreme-programming/#whole

Planning Game

- XP planning addresses two key questions in software development: predicting what will be accomplished by the due date, and determining what to do next.
- *Release Planning* is a practice where the Customer presents the desired features to the programmers, and the programmers estimate their difficulty.
- *Iteration Planning* is the practice whereby the team is given direction every couple of weeks. (Sprints)

https://ronjeffries.com/xprog/what-is-extreme-programming

Customer Tests

- As part of presenting each desired feature, the XP Customer defines one or more automated acceptance tests to show that the feature is working. The team builds these tests and uses them to prove to themselves, and to the customer, that the feature is implemented correctly.

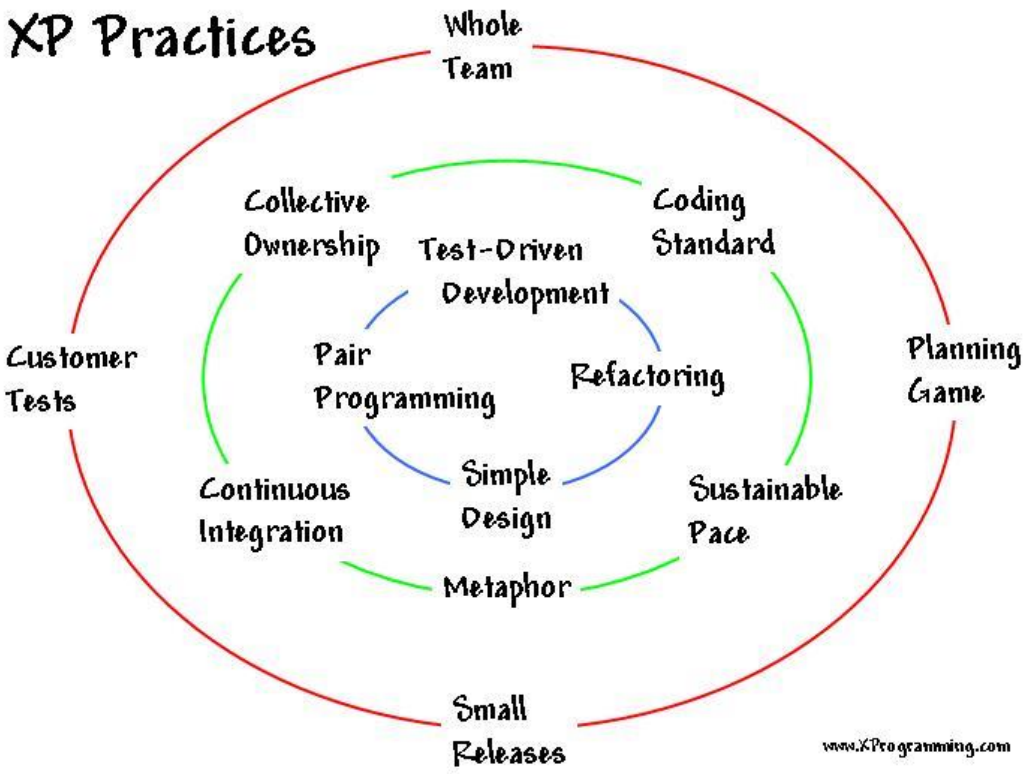
https://ronjeffries.com/xprog/what-is-extreme-programming/#whole

Small Releases

- XP teams practice small releases in two important ways:
  - First, the team releases running, tested software, delivering business value chosen by the Customer, every iteration.
  - Second, XP teams release to their end users frequently as well.

https://ronjeffries.com/xprog/what-is-extreme-programming/#whole

Practices of XP



Further Reading and Descriptions

http://ronjeffries.com/xprog/what-is-extreme-programming/.

Coding standards

- XP teams follow a common coding standard, so that all the code in the system looks as if it was written by a single – very competent – individual. The specifics of the standard are not important: what is important is that all the code looks familiar, in support of collective ownership.

http://ronjeffries.com/xprog/what-is-extreme-programming/.

Collective Ownership

- On an Extreme Programming project, any pair of programmers can improve any code at any time. This means that all code gets the benefit of many people's attention, which increases code quality and reduces defects.

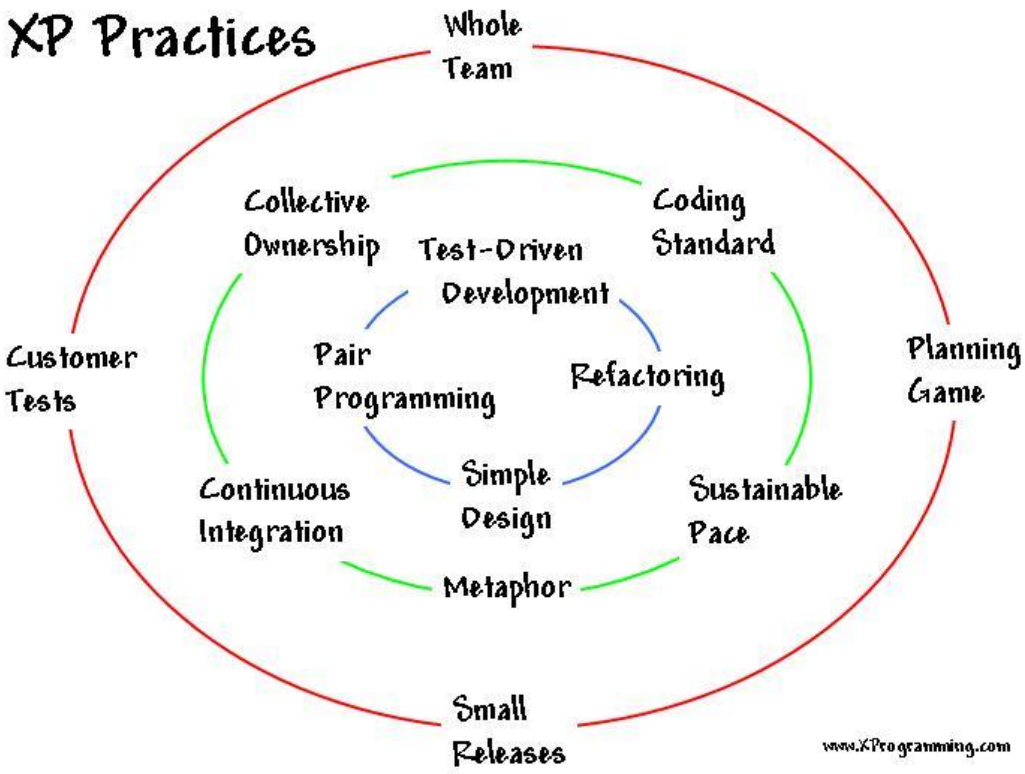
http://ronjeffries.com/xprog/what-is-extreme-programming/.

Metaphor

- Extreme Programming teams develop a common vision of how the program works, which we call the "metaphor". At its best, the metaphor is a simple evocative description of how the program works, such as "this program works like a hive of bees, going out for pollen and bringing it back to the hive" as a description for an agent-based information retrieval system.

http://ronjeffries.com/xprog/what-is-extreme-programming/.

Practices of XP



Further Reading and Descriptions

http://ronjeffries.com/xprog/what-is-extreme-programming/.

Refactoring

- The refactoring process focuses on removal of duplication (a sure sign of poor design), and on increasing the "co*hesion*" of the code, while lowering the "co*upling*". High cohesion and low coupling have been recognized as the hallmarks of well-designed code for at least thirty years. The result is that XP teams start with a good, simple design, and always have a good, simple design for the software.

http://ronjeffries.com/xprog/what-is-extreme-programming/.

Simple Design

- XP teams build software to a simple but always adequate design. They start simple, and through testing and design improvement, they keep it that way. An XP team keeps the design exactly suited for the current functionality of the system. There is no wasted motion, and the software is always ready for what's next.

http://ronjeffries.com/xprog/what-is-extreme-programming/.

Overview

- ~~XP~~
  - ~~General concepts~~

- Specific XP concepts
  - Pair programming
  - Test Driven Development
  - Continuous Integration

Pair Programming

"You'll never work alone"

Not without precedent

Pair programming

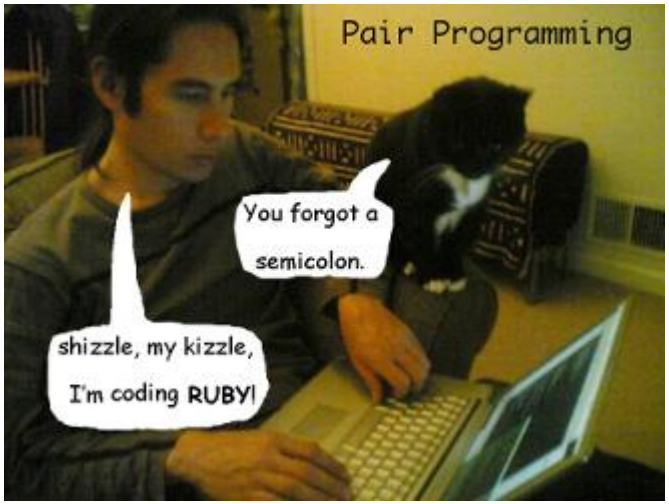
- Two developers working on the same task as a team
- One controls the keyboard one sits looking over their shoulder
- Driver
  - This person writes the code
- Navigator
  - This person reviews each line as it is typed

Advantages

- Higher code quality
  - Fewer bugs, code rewrites, integrations problems
- Expert novice pairing can help the novice to learn about the system and best practices
- Tends to produce more design alternatives and catches design defects earlier

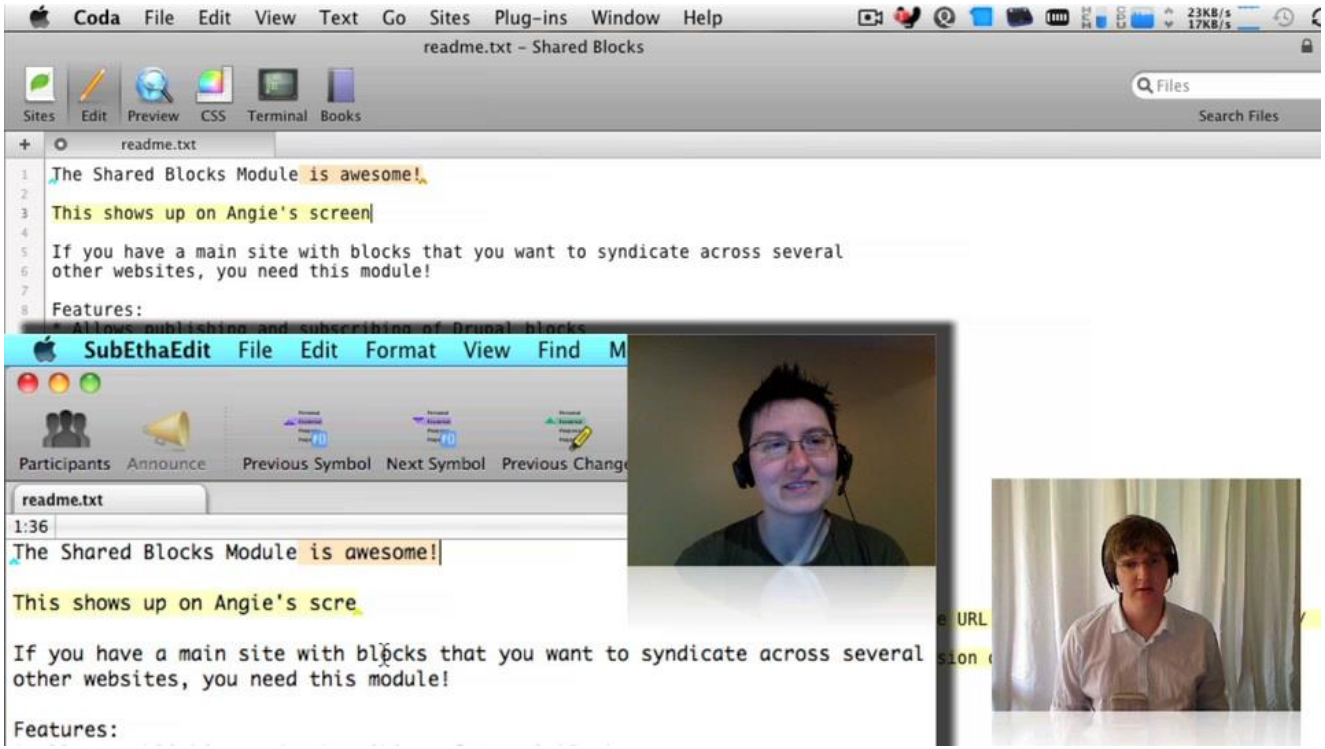
Disadvantages

- There is a high probability of disengagement
- "Watch the master" phenomenon
- Working relationship needs to be good
- Hard sell to management
  - Two people working on 1 feature



Remote pair programming

- Using communications technology
  - Screen sharing
  - IM clients, VOIP etc

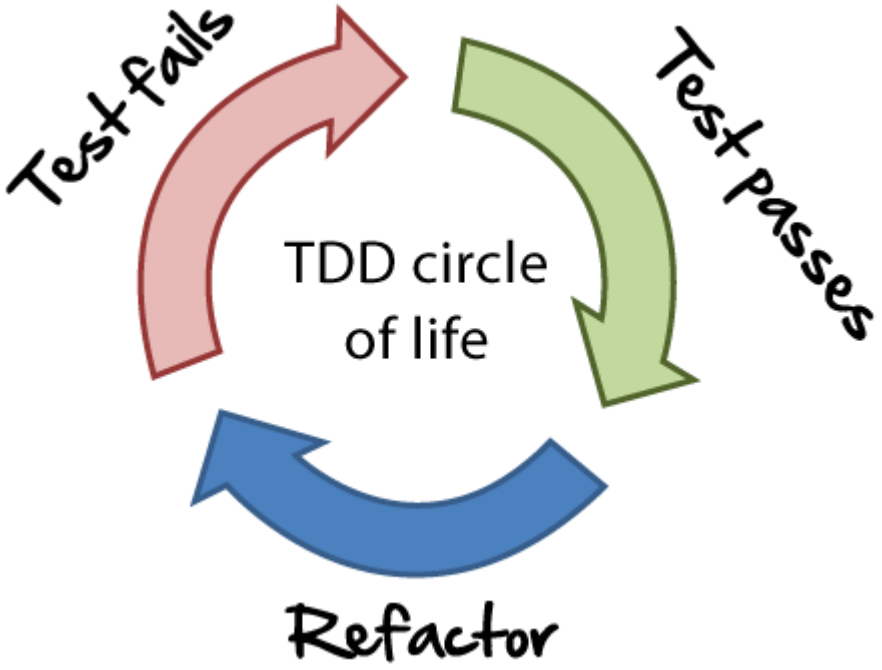


Overview

- ~~XP~~
  - ~~General concepts~~

- Specific XP concepts
  - ~~Pair programming~~
  - **Test Driven Development**
  - Continuous Integration

Test Driven Development (TDD)



TDD Cycle

- Create a test
  - Each new feature requires a test
- Run all tests
  - Make sure the new test fails
- Write the code
  - Doesn't have to be perfect, just pass the test
- Run all tests
  - If all tests pass, requirements are met
- Refactor code
  - TDD can result in duplication, this should be removed
- Repeat the process

Principles of TDD

- Never write new functionality without a failing test
- Continually make small incremental changes
- Keep the system running at all times
  - No one can make a change that breaks the system
  - Failures must be addressed immediately

Advantages

- Discourages “gold plating” of implementation
- Forces the developer to specify an end criteria
- Encourages loose coupling

Disadvantages

- Big time investment
- Complexity in writing appropriate test cases
- Design changes
- Mock code to pass tests
- Customer may not wish to get involved in creating acceptance tests

Interesting - IBM Study

- Study carried out by IBM focussed on a team that had been practising TDD for 5 years and delivered 10 releases of a software product

- Quality was the big winner, much improved, fewer defects/bugs etc

- Productivity did **decrease** but not dramatically

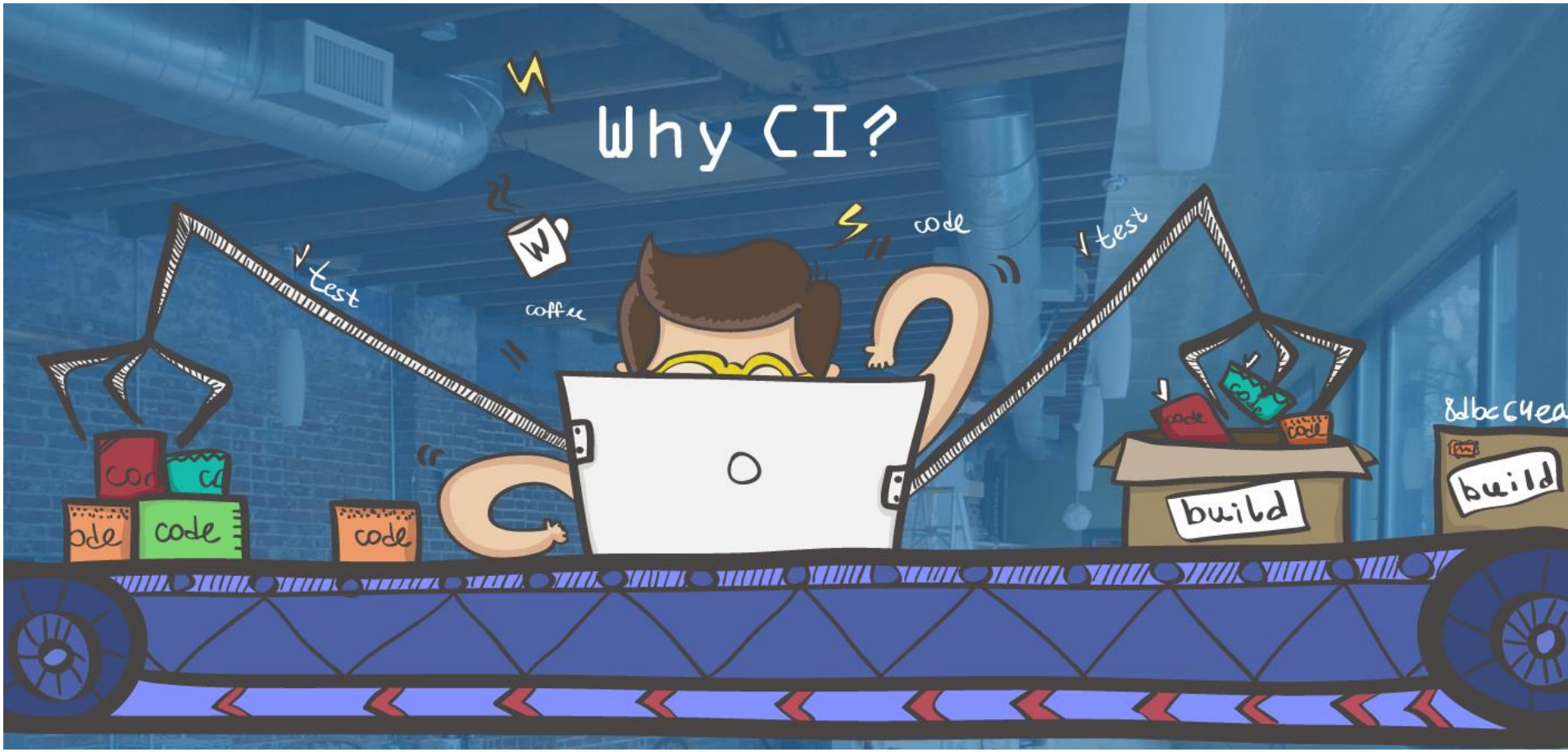
Sanchez, J., Laurie Williams, and E. Michael Maximilien. "A Longitudinal Study of the Use of a Test-Driven Development Practice in Industry." Proc. Agile. 2007.

Learning objectives

- ~~XP~~
  - ~~General concepts~~

- Specific XP concepts
  - ~~Pair programming~~
  - ~~Test Driven Development~~
  - **Continuous Integration**

Continuous Integration

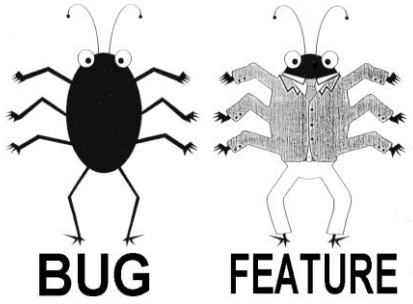


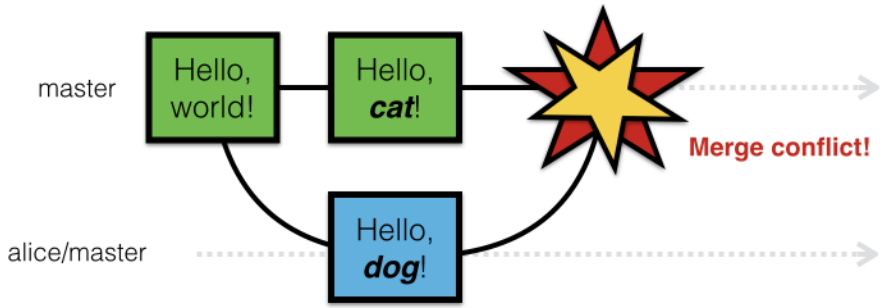
Continuous Integration (CI)

- CI is a practice where members integrate their changes frequently.
  - Often daily
- Each integration is verified by an automated build including tests to detect integration errors as early as possible.
  - Often upon commit, builds are run to make sure everything is okay

Development before CI

- Lots of bugs
- Infrequent commits
- Difficult integration
- Infrequent releases
- Testing happens late

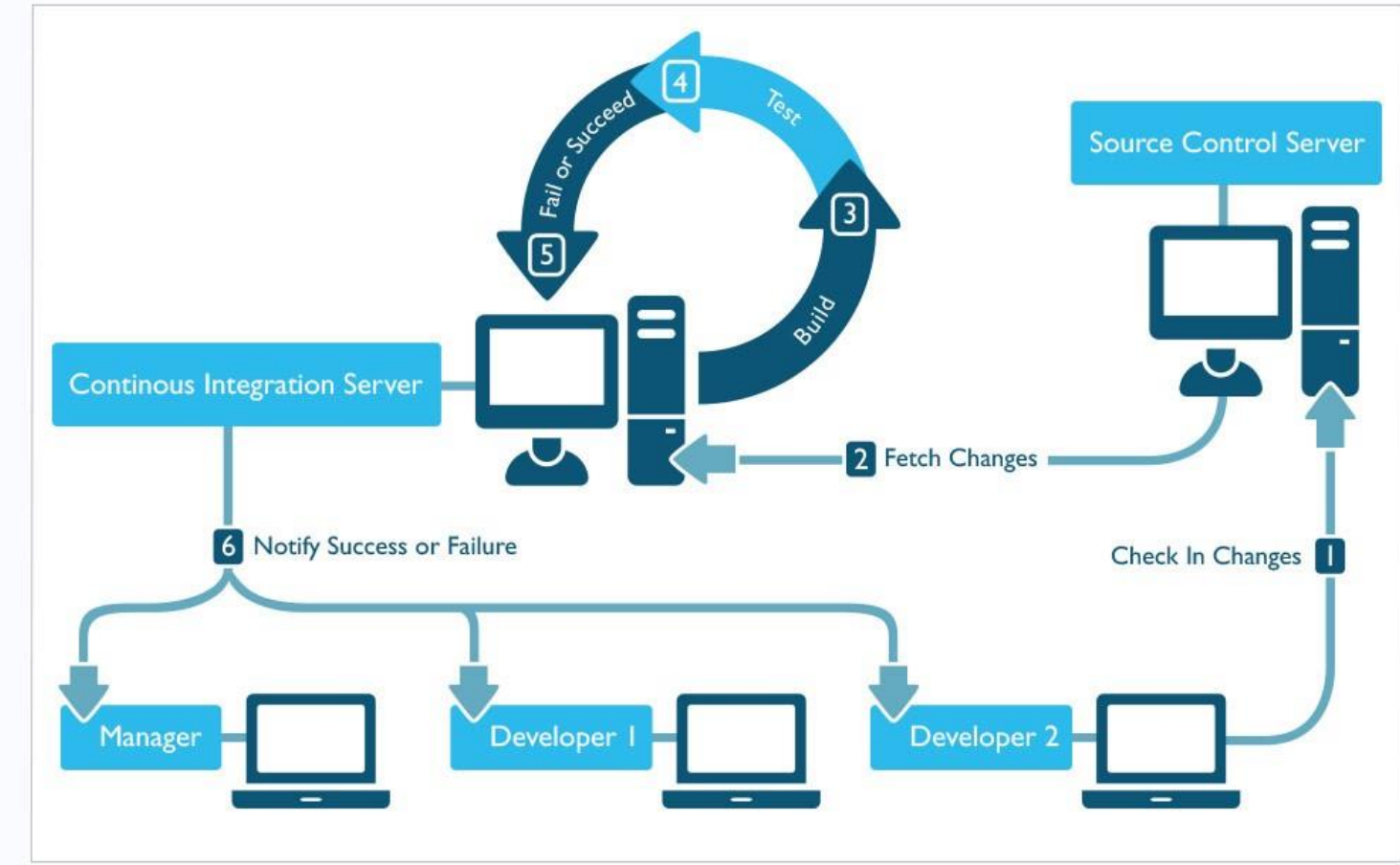




Benefits of CI

- Fail early/fast
  - Detect problems as early as possible
- Facilitates continuous deployments
  - Deploying every good build live to production
- Enables automated testing
  - Tests are run during the build process

Overview



https://insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html

Drawbacks of using CI

- Initial setup required
  - Can take a couple of weeks to get it running properly within an organisation
- Excellent tests must be developed
  - CI will run all the automated tests but this requires substantial up front development effort.

Popular CI software

CODESHIP

circleci

Bamboo

Solano Labs

snap

semaphore

AppVeyor

drone.io

SETTING UP OUR DEVELOPMENT ENVIRONMENTS

Dr. Enda Barrett – Enda.Barrett@nuigalway.ie

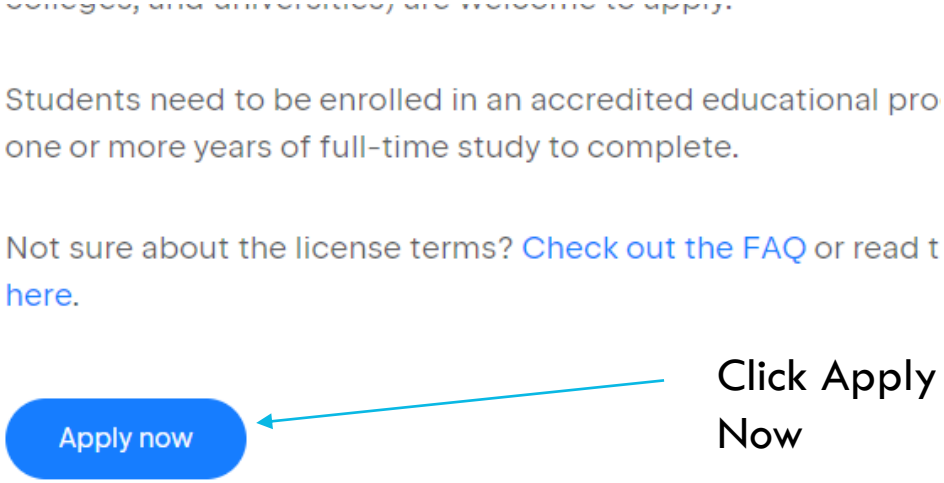
Ollscoil na Gaillimhe
University of Galway

Getting things set up

- Install an IDE
- Install software (NodeJS runtime environment)
- Quick overview of Firebase
- Claim cloud credits
- Create a new Google Account using your University email address
- Create a new Firebase project
- Create a new local project
- Using the Command Line
- Install Firebase CLI
- Initialise the projects to link with the cloud
- Deploy to the cloud

WebStorm - IntelliJ

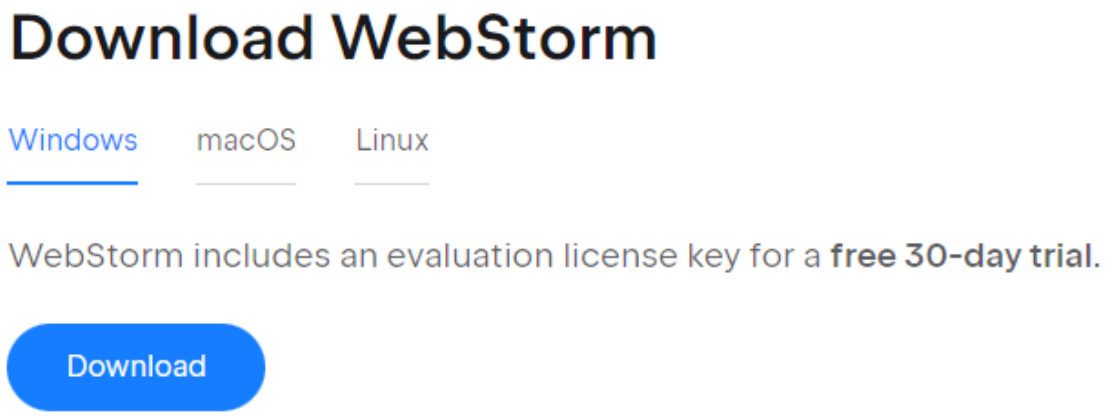
- Navigate to
  - https://www.jetbrains.com/community/education/#students



- Fill in the form details and you will get a 1 year license
  - There are lots of alternatives too if you prefer to use something else

Download WebStorm

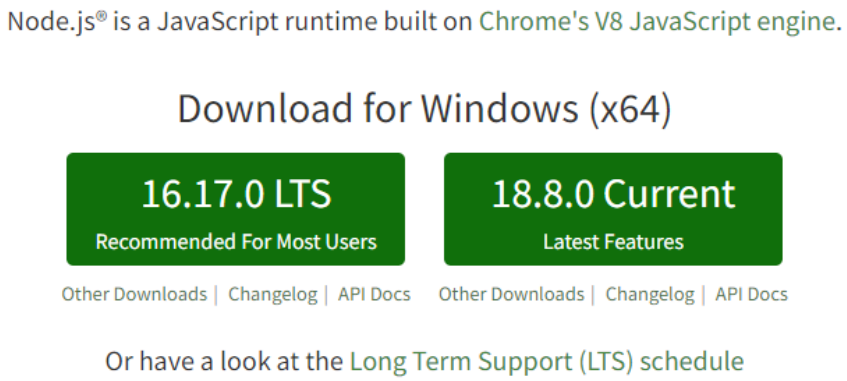
- https://www.jetbrains.com/webstorm/download/#section=windows



Install NodeJS

- Navigate to https://nodejs.org/en/

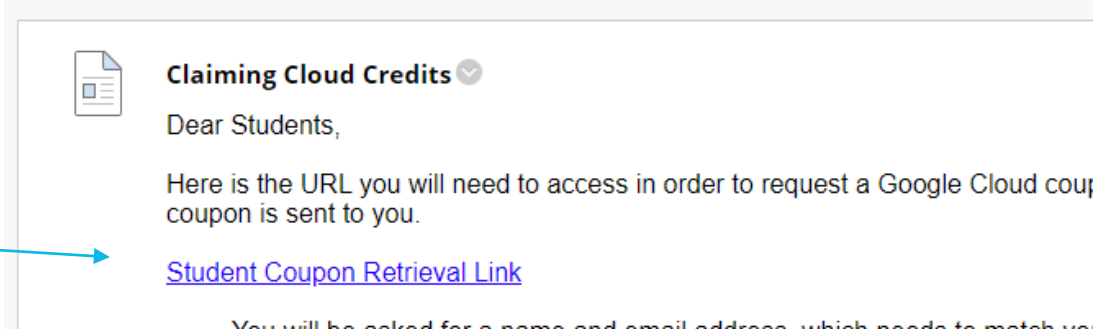
- Install the latest LTS (Long Term Support) version of NodeJS



Cloud Credits

- Google have kindly provided each student with $25 in cloud credits to use on their Firebase platform
- On Blackboard under Week 2, you will see this post

Click the link

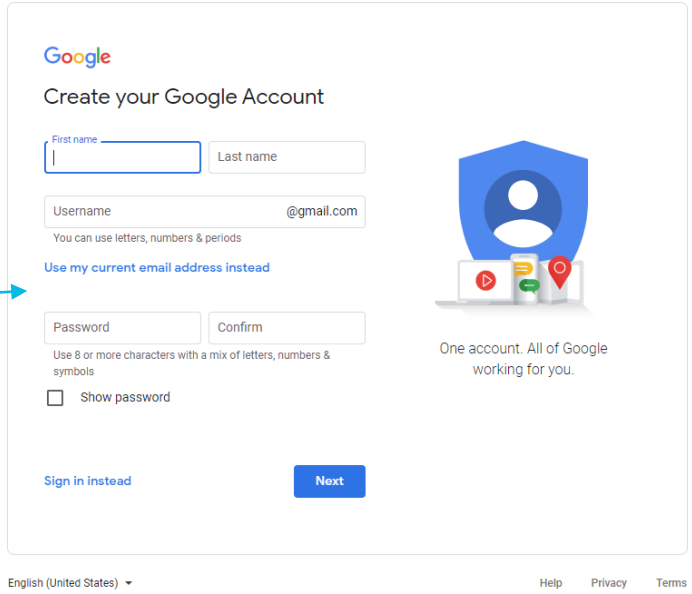


- You must use your university email address when claiming the credits

Create a Google Account

- Perform a quick search for Google Account Creation

Click on use my current email address



- Use your **university email address** for this. Same one you claimed the credits with!

CREATING A NEW FIREBASE PROJECT

Dr. Enda Barrett – Enda.Barrett@nuigalway.ie

Ollscoil na Gaillimhe
University of Galway

Firebase – Quick overview

- Originally Firebase was a (Mobile) Backend as a Service offer which is a subset of a Platform as a Service
- Whilst an IaaS provider does help you to move away from on-prem hosting, you do have a good bit of setup. Install a database on the virtual server, varying software packages, even an OS! You have to keep it updated and secure etc.
- With a PaaS offering such as Firebase you can focus on coding!
- It provides a suite of services
  - Storage
  - Database
  - Authentication
  - Machine Learning
  - Functions
  - Messaging
  - Hosting

Firebase

My sample project

Project Overview

Product categories

Build
- Authentication
- App Check
- Firestore Database
- Realtime Database
- Extensions
- Storage
- Hosting
- Functions
- Machine Learning
- Remote Config
- Cloud Messaging

Release & Monitor

Analytics

Engage

All products

Spark
No-cost $0/month
Upgrade

Receive email updates about new Firebase features, research, and events Sign up

My sample project

Spark plan

Get started by adding Firebase to your app

iOS+

Add an app to get started

Store and sync app data in milliseconds

Authentication

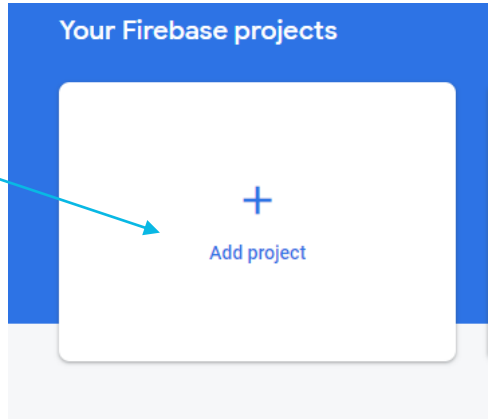
Authenticate and manage users

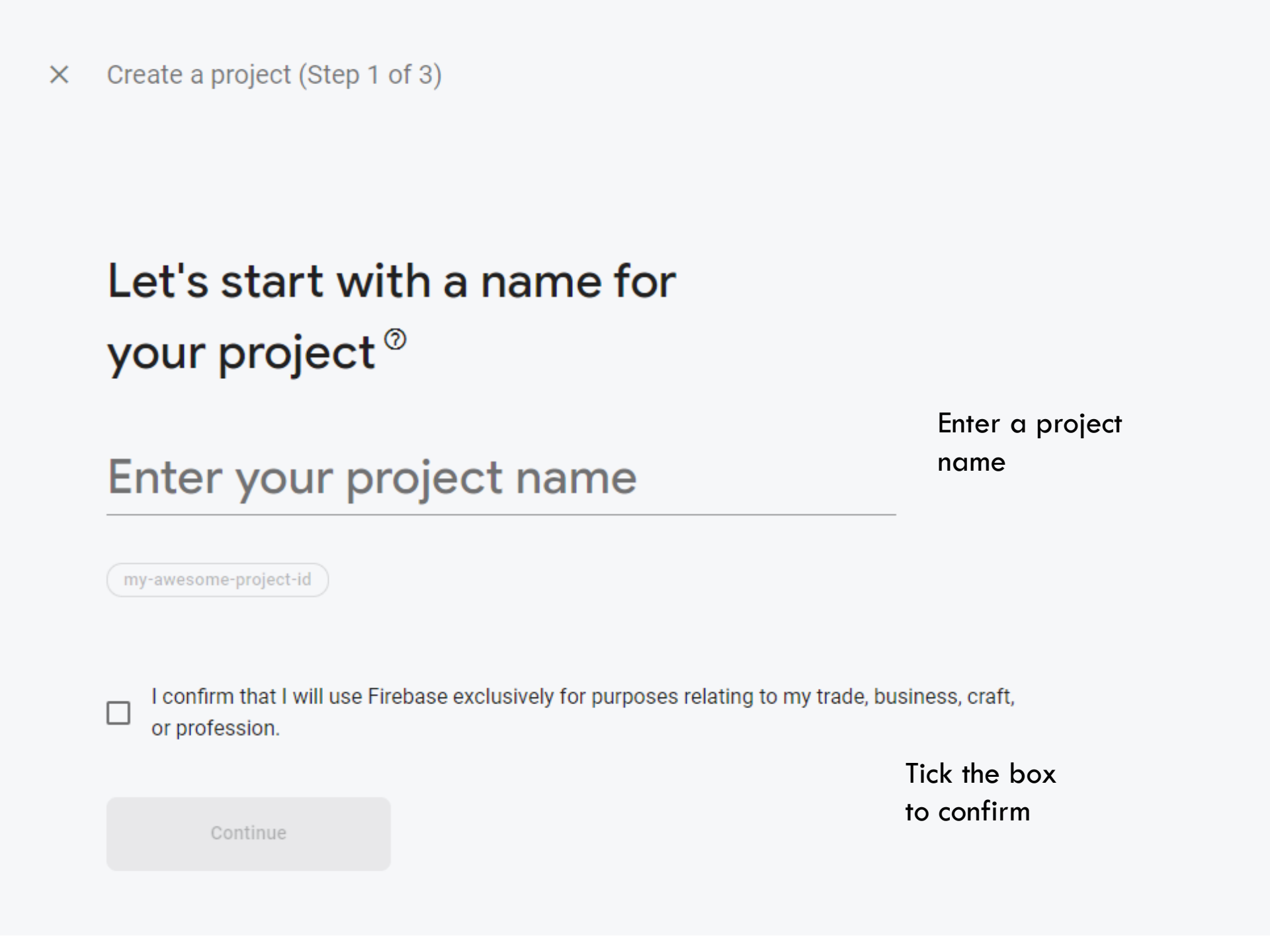
Cloud Firestore

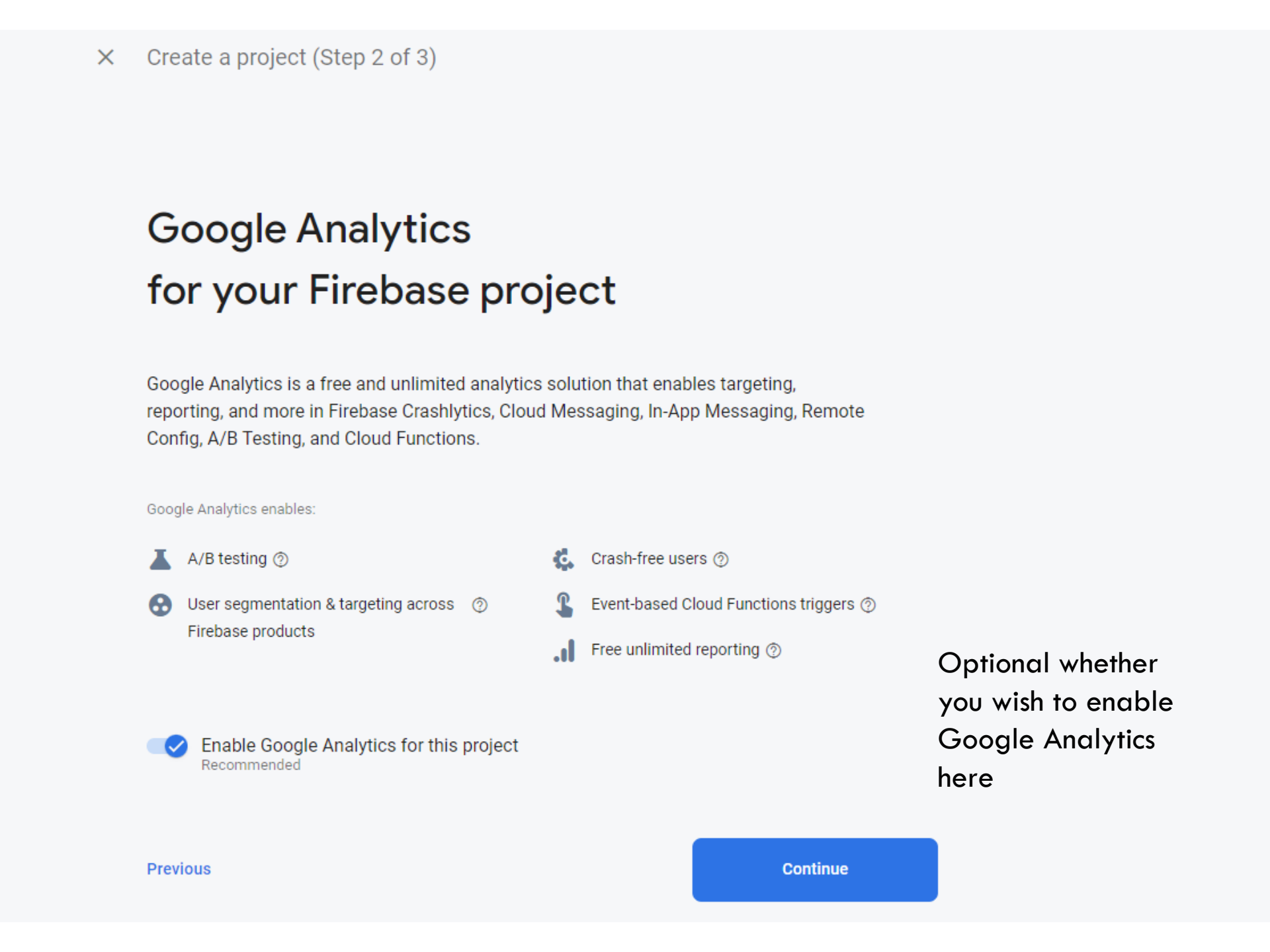
Realtime updates, powerful queries, and automatic scaling

Firebase project

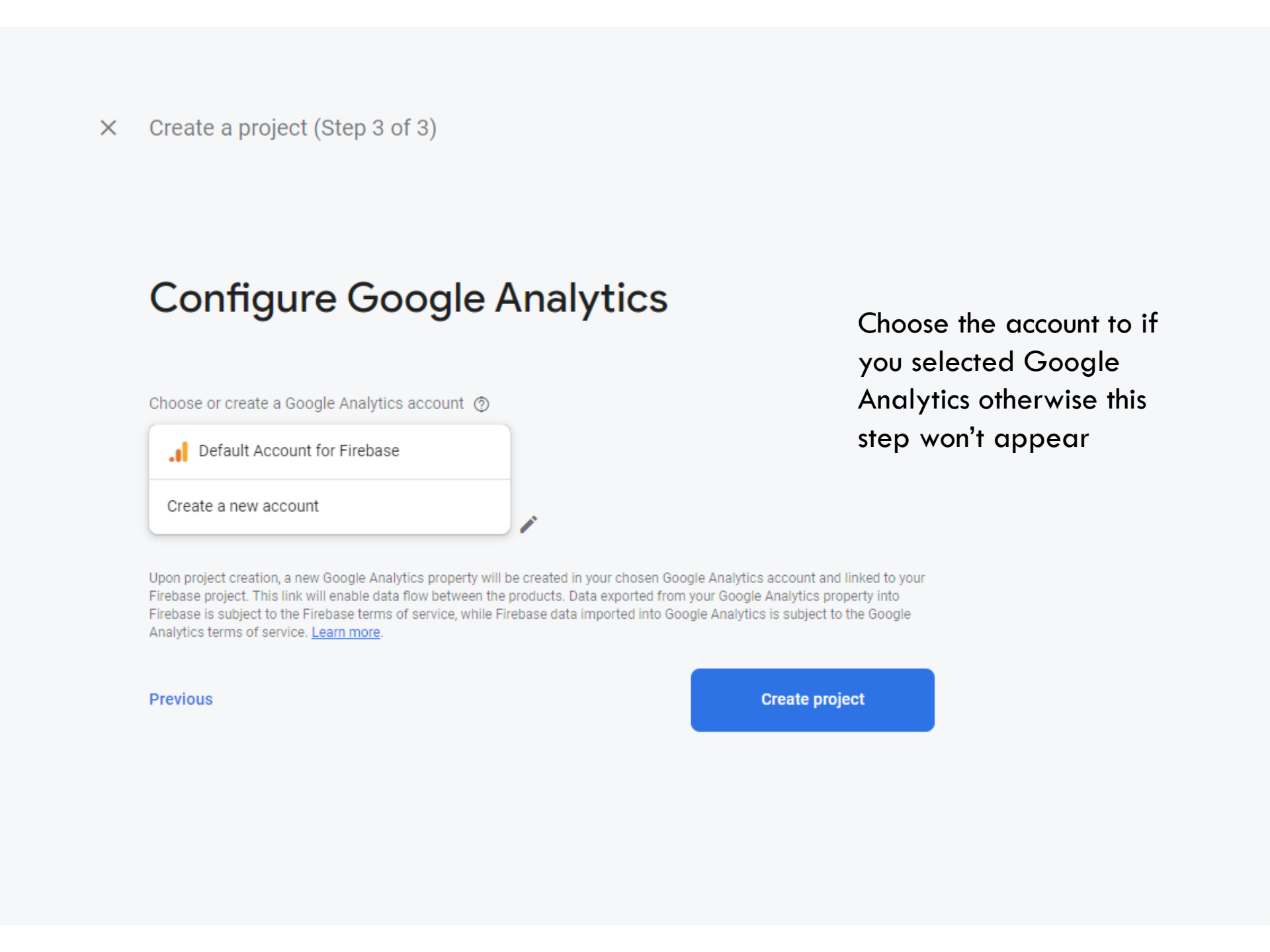
- We will need to create Firebase projects for our apps
- Login using your Google Account created with your university email address
- Create a new project by clicking here




Create a project (Step 1 of 3)
Let's start with a name for your project
Enter your project name
my-awesome-project-id
I confirm that I will use Firebase exclusively for purposes relating to my trade, business, craft, or profession.
Continue
Enter a project name
Tick the box to confirm



Optional whether you wish to enable Google Analytics here



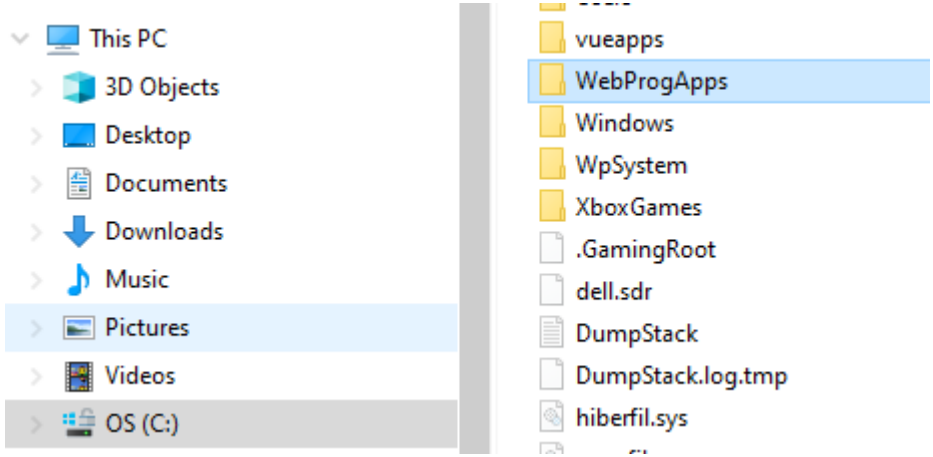
Choose the account to if you selected Google Analytics otherwise this step won't appear

Creating a local directory

- In order to write our applications we will code them locally on our computers, save our code files and then push them to the cloud.
- In order to work locally we need to create a folder to store our code.
- It is best to place this on root the of your machine or some place that is easy to access from a **command line** perspective.
- In windows create a folder on C:\WebProgApps

May look something like this

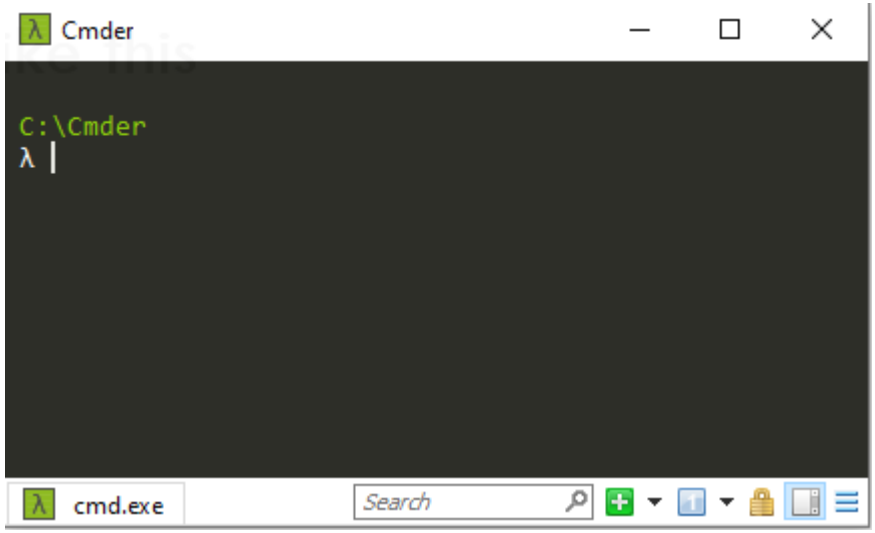
- If you have a windows machine you can add a folder like this.



- On a MAC you can use Finder to the same, if using Linux you already know this!

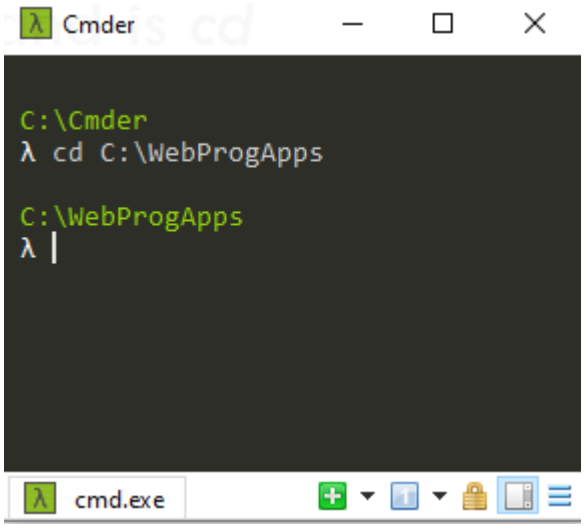
What is the Command Line Interface

- A mechanism through which you can execute commands to the OS. It's how we used to operate computers before their was a nice GUI.
- Depending on your OS (Windows, MAC or Linux) you search for Command Prompt(Windows), Terminal (MAC or Linux).
- It will look something like this



Navigate to the root

- Depending on the OS the default directory that the command line will begin with once opened can be different.
- We wish to navigate to the folder we have created on the root to get it set up for development.
- To change directory the command is *cd*
- Type cd C:\WebProgApps and press Enter on the keyboard

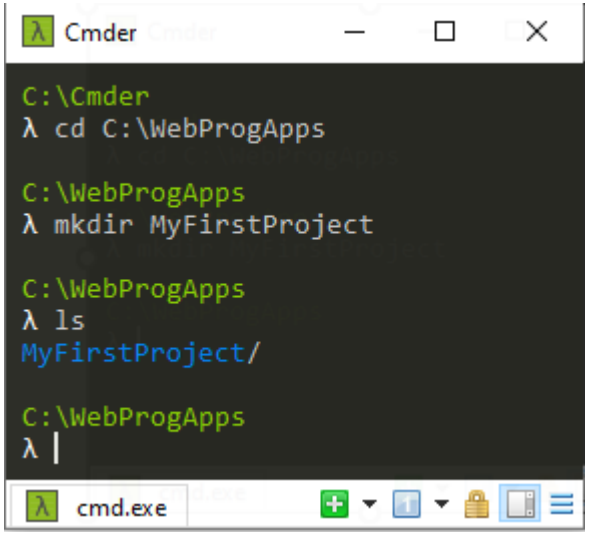


Creating a directory

- You can avoid having to use the GUI to create a directory and use the mkdir command instead.
- If you called your project on Firebase MyFirstProject then we can use the same name for the local folder which will contain the code for this project
- Type mkdir MyFirstProject

- Check the folder it should be there using

  dir (windows) or ls (MAC)



Install the Firebase CLI

- Firebase CLI - https://github.com/firebase/firebase-tools
- A suite of tools for managing, visualising and deploying Firebase projects
- Install it onto our machines and then we can use it to configure our projects, link them to the cloud backend and then deploy them onto the web!
- Two ways to install it, you can use the standalone binaries which are available in the link below or you can use ***npm*** *(Node Package Manager)*

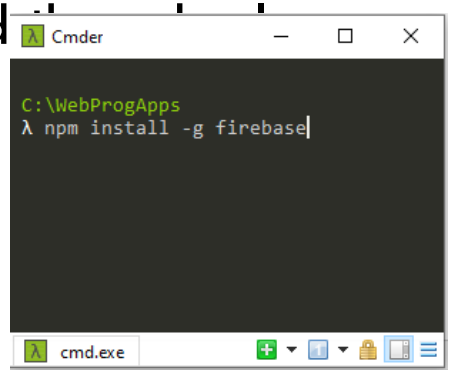
https://firebase.google.com/docs/cli

NPM – Node Package Manager

- Node Package Manager, is the default package manager for the NodeJS runtime.
  - https://www.npmjs.com/
- It contains thousands of Open Source projects containing code which we call packages.
- We can install packages onto our machine by using the NPM command. These can be executed as standalone programs on our machines (Firebase CLI) or included in our own programs that we write.
- When we installed NodeJS (back in Slide 4), NPM was also included. Thus we can use it by typing the command npm

Install Firebase

- Step 1 – Install the Firebase CLI
  - npm install -g firebase
  - npm install -g firebase-tools
  - The g argument is for global – all projects on the machine will have access to the CLI
  - If you are using a MAC you will need to include sudo
    - sudo npm install –g firebase
    - sudo npm install –g firebase-tools
- Once you have typed in the command press Enter on the keyboard
- Install it onto our machines and then we can use it to configure our projects, link them to the cloud backend and [illegible] them onto the web!

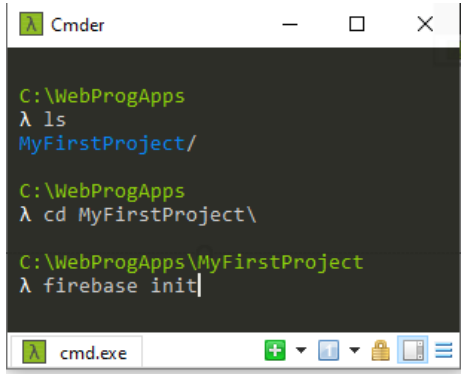


Logging into Firebase

- In order to perform the next sequence of steps we need to log into Firebase using the command line.
- Enter the command firebase login and hit Enter
- This will open up a web browser where you can login using your Google Account created with your university email address.

Initialise our project folder

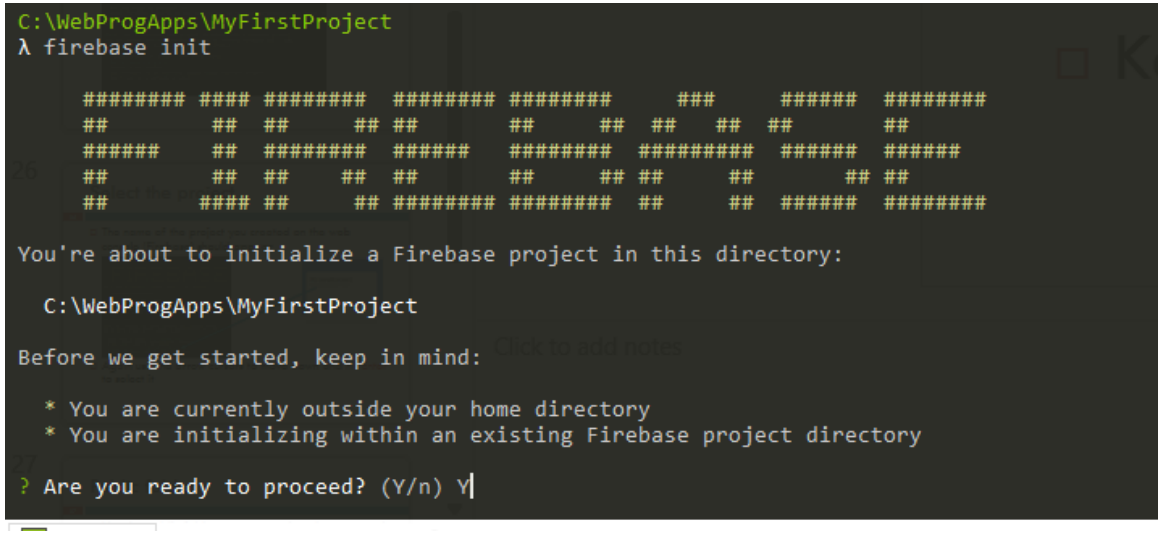
- In order to link our local project folders with the online cloud project we need to initialise it.
- Make sure you are in the correct directory on the command line



- Enter the command firebase init and press Enter

Sequence of init steps

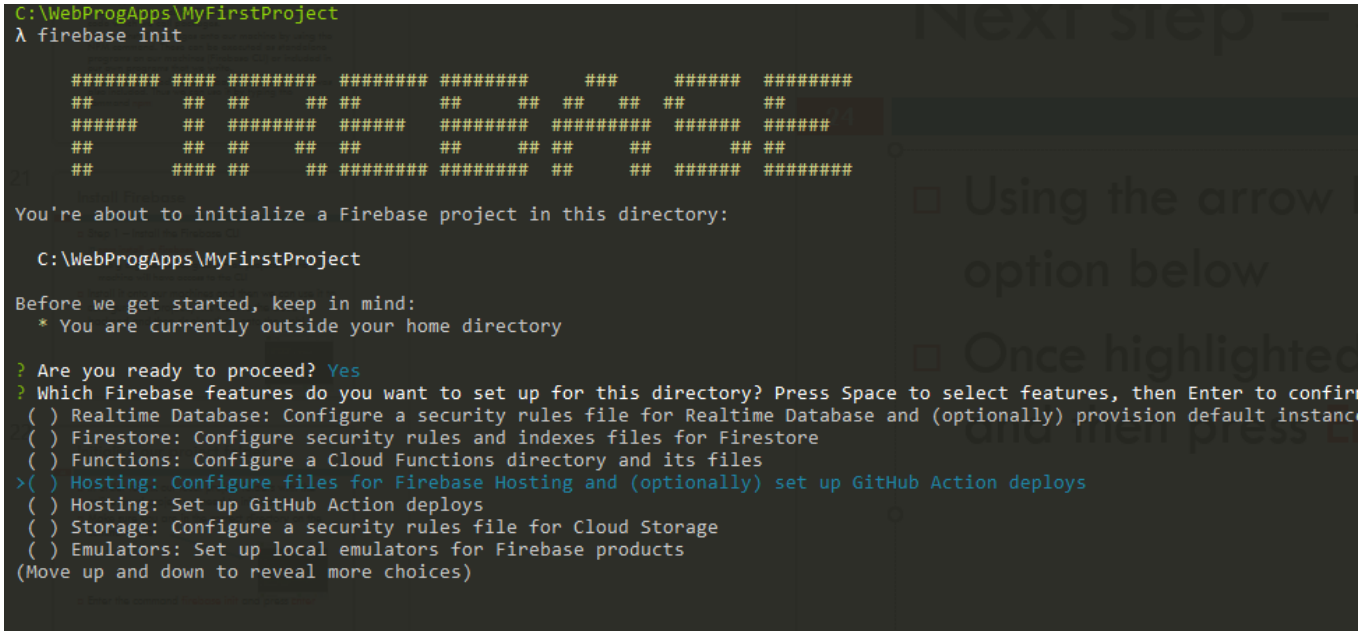
- If everything has worked correctly you should now see the following



- Key in Y and pres Enter

Next step – Setup hosting

- Using the arrow keys move down to the highlighted option below
- Once highlighted select it using the spacebar key and then press Enter



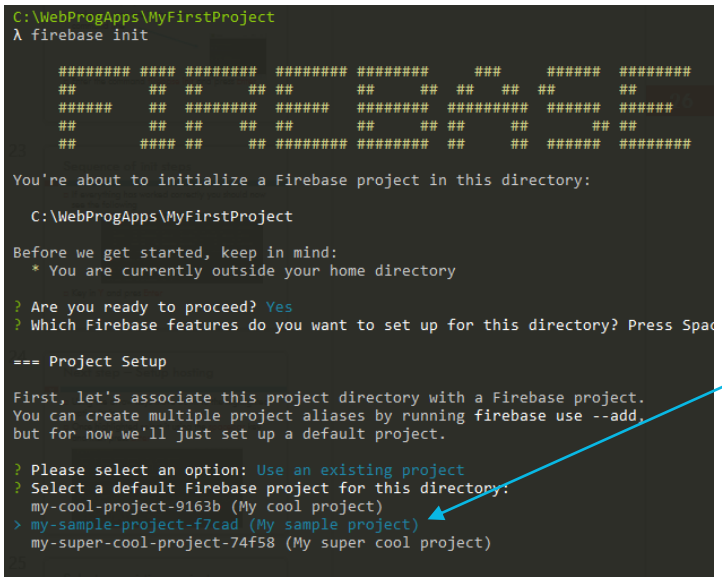
Select an existing project

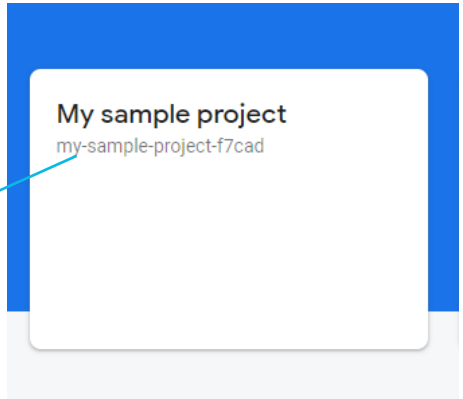
- Select use an existing project by hitting Enter

```
  C:\WebProgApps\MyFirstProject

Before we get started, keep in mind:
  * You are currently outside your home directory

? Are you ready to proceed? Yes
? Which Firebase features do you want to set up for this directory? Press Space to

=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

? Please select an option: (Use arrow keys)
> Use an existing project
  Create a new project
  Add Firebase to an existing Google Cloud Platform project
  Don't set up a default project
```

Select the project

- The name of the project you created on the web console (Firebase) should appear





- Again use the arrow cursors to move down and hit Enter to select it

Public folder

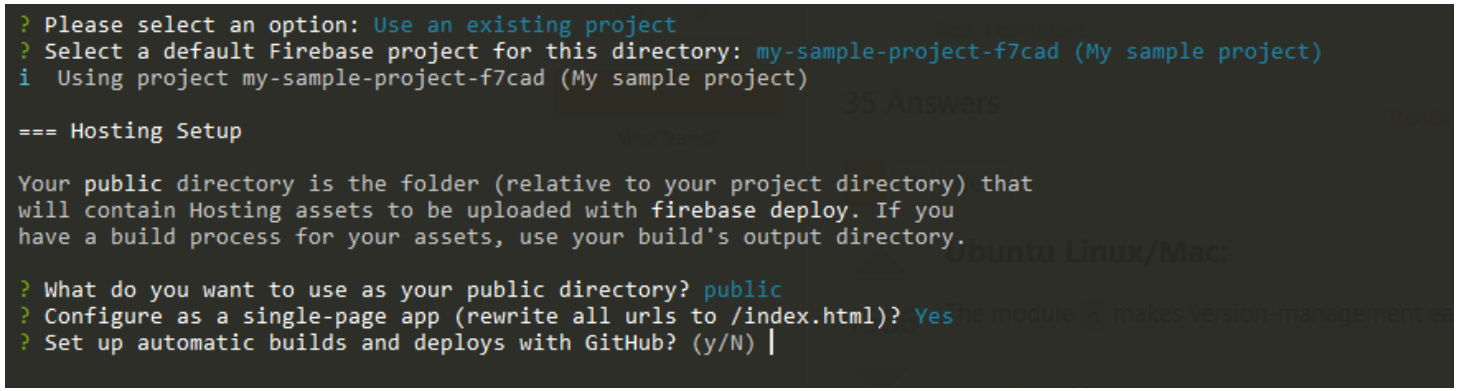
- Use the public folder as prompted on the command line. This is where we will place all of frontend code.

```
? Please select an option: Use an existing project
? Select a default Firebase project for this directory: my-sample-project-f7cad (My sample project)
i  Using project my-sample-project-f7cad (My sample project)

=== Hosting Setup

Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.

? What do you want to use as your public directory? (public)
```

- Press the Enter key on your keyboard

URL rewrite

- The next option is a technical setting which configures the app as a Single Page App (SPA) where requests are directed at index.html
- Select Y here and press Enter



Automatic builds and deploys

- There is no need to setup automatic builds and deploys at that this point

- Select N for this and press Enter

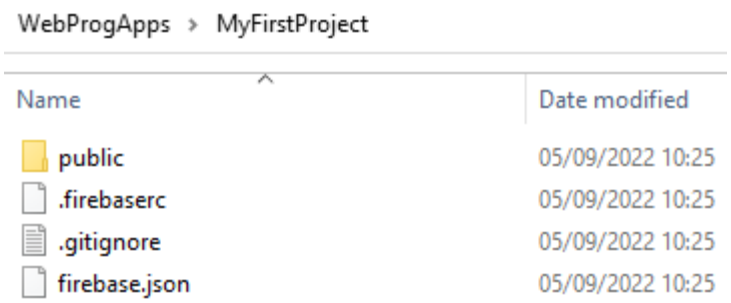
```
? Please select an option: Use an existing project
? Select a default Firebase project for this directory: my-sample-project-f7cad (My sample project)
i  Using project my-sample-project-f7cad (My sample project)

=== Hosting Setup

Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.

? What do you want to use as your public directory? public
? Configure as a single-page app (rewrite all urls to /index.html)? Yes
? Set up automatic builds and deploys with GitHub? (y/N) N
```

Firebase initialisation should be complete


```
C:\WebProgApps\MyFirstProject
λ firebase init

     ######## #### ########  ######## ########     ###     ######  ########
     ##        ##  ##     ## ##       ##     ##  ##   ##  ##       ##
     ######    ##  ########  ######   ########  #########  ######  ######
     ##        ##  ##    ##  ##       ##     ## ##     ##       ## ##
     ##       #### ##     ## ######## ########  ##     ##  ######  ########

You're about to initialize a Firebase project in this directory:

  C:\WebProgApps\MyFirstProject

Before we get started, keep in mind:
  * You are currently outside your home directory

? Are you ready to proceed? Yes
? Which Firebase features do you want to set up for this directory? Press Space to select features, th

=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

? Please select an option: Use an existing project
? Select a default Firebase project for this directory: my-sample-project-f7cad (My sample project)
i  Using project my-sample-project-f7cad (My sample project)

=== Hosting Setup

Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.

? What do you want to use as your public directory? public
? Configure as a single-page app (rewrite all urls to /index.html)? Yes
? Set up automatic builds and deploys with GitHub? No
+  Wrote public/index.html

i  Writing configuration info to firebase.json...
i  Writing project information to .firebaserc...
i  Writing gitignore file to .gitignore...

+  Firebase initialization complete!

C:\WebProgApps\MyFirstProject
λ
```

Files in the folder

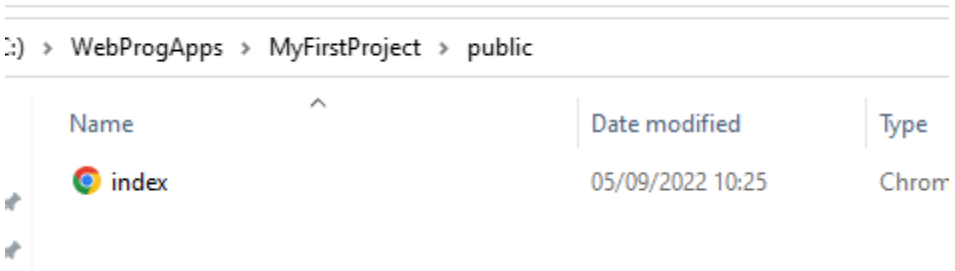
- If you look in your app folder on your machine you should now see a bunch of new files have been created, a public folder and an index.html file within the public folder



Deploying to the cloud

- The final step is to deploy to the cloud.

- We haven't written any code yet or created our first web page, but during the init process Firebase created a page index.html which you can find in the public folder.



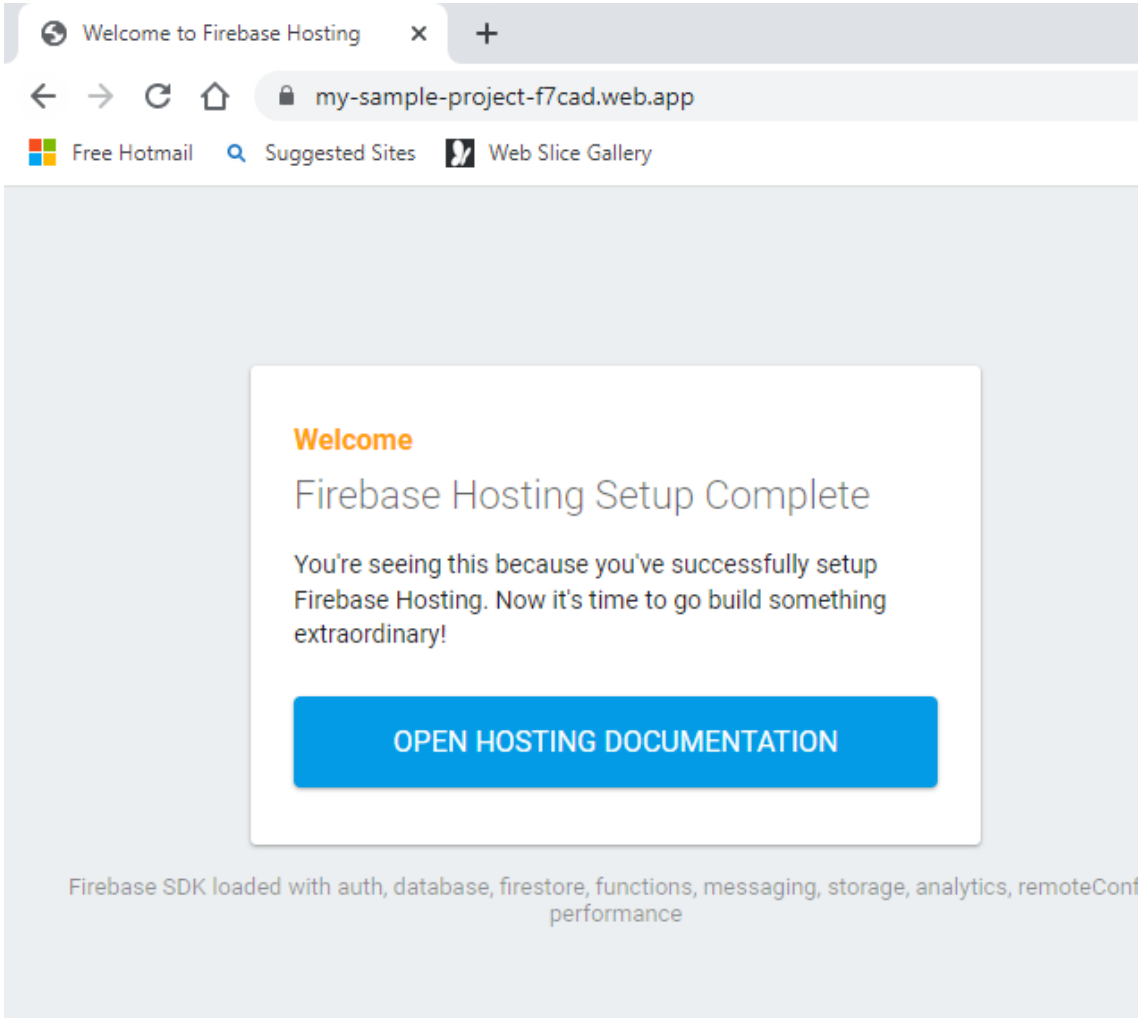
Firebase deploy

- The command to deploy our apps to the cloud is
  - firebase deploy

```
C:\WebProgApps\MyFirstProject
λ firebase deploy

=== Deploying to 'my-sample-project-f7cad'...

i  deploying hosting
i  hosting[my-sample-project-f7cad]: beginning deploy...
i  hosting[my-sample-project-f7cad]: found 1 files in public
+  hosting[my-sample-project-f7cad]: file upload complete
i  hosting[my-sample-project-f7cad]: finalizing version...
+  hosting[my-sample-project-f7cad]: version finalized
i  hosting[my-sample-project-f7cad]: releasing new version...
+  hosting[my-sample-project-f7cad]: release complete

+  Deploy complete!

Project Console: https://console.firebase.google.com/project/my-sample-project-f7cad/overview
Hosting URL: https://my-sample-project-f7cad.web.app

C:\WebProgApps\MyFirstProject
λ
```

Copy and paste the hosting URL and pop it into the address bar of your browser

It works!


Welcome to Firebase Hosting
my-sample-project-f7cad.web.app
Free Hotmail
Suggested Sites
Web Slice Gallery
Welcome
Firebase Hosting Setup Complete
You're seeing this because you've successfully setup Firebase Hosting. Now it's time to go build something extraordinary!
OPEN HOSTING DOCUMENTATION
Firebase SDK loaded with auth, database, firestore, functions, messaging, storage, analytics, remoteConf
performance

CREATING OUR FIRST WEB PAGE WITH HYPERTEXT MARKUP LANGUAGE (HTML)

Dr. Enda Barrett
Enda.Barrett@universityofgalway.ie

OÉ Gaillimh
NUI Galway

HTML

- Stands for Hyper Text Markup Language (HTML)
- Notation for describing document structure and formatting
- A html file has a .html or .htm extension
- It is rendered by a browser

Simple HTML Document

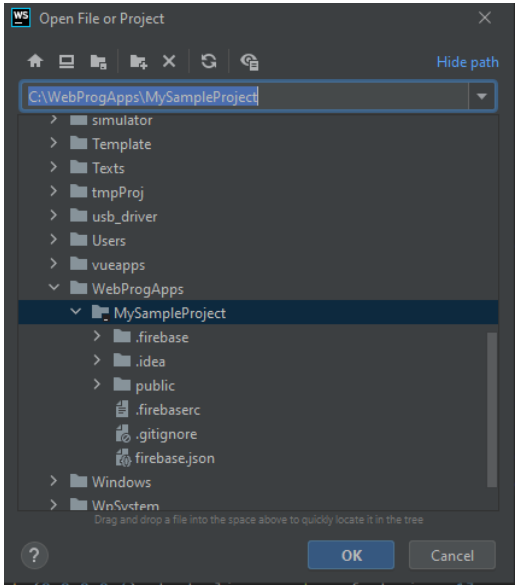
- Document starts with: `<html>`
- Document ends with: `</html>`
- Text between `<head>` and `</head>` is header information which is not rendered by the browser
  - `<title>` and `</title>` displays the title of the document

- Everything between `<body>` and `</body>` is rendered and displayed by the browser
- Contains actual text to display and tags defining its style, layout etc. plus additional elements e.g. images

Open Project in WebStorm

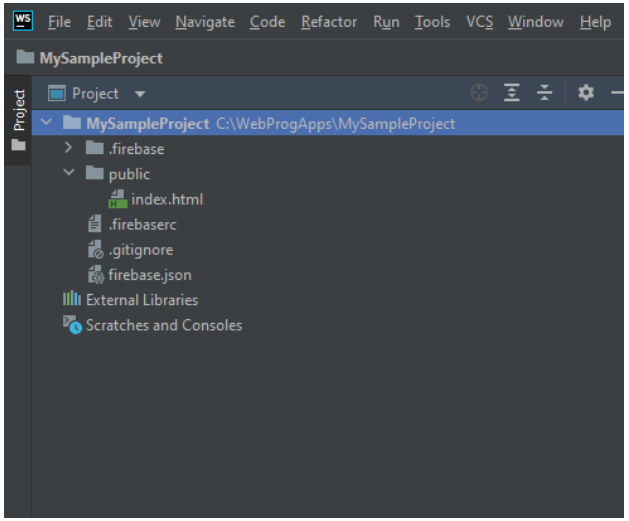
- Using WebStorm (or preferred IDE) open up the firebase project that you created in the previous lecture.

- In WebStorm select File>Open
- This will pop up a dialog where you can select your project (MySampleProject)



Once opened in WebStorm

- You will see a view (see screenshot below) where the project is on the left hand-side including the files and folders
- If you expand the public folder you will see the index.html file.
- Double click it to start editing
- This file is our first HTML page!



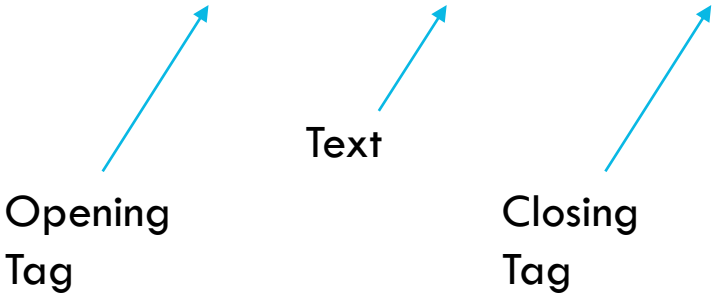
Exercise 1 - HTML – Hello World


```
<!DOCTYPE html>
<html>
<head>
  <title>Simple Page</title>
</head>
<body>
<h1>Hello World!</h1>
</body>
</html>
```

- Delete all the existing markup from index.html
- Take the following markup and insert it into index.html
- Surround the “Hello World” text with a heading 1 tag – (search online).
- Save the file
- Deploy it to Firebase (firebase deploy) command line

HTML Tags

- HTML contains text to display and markup tags
  - <h1>Hello World</h1>



- The tags tell the browser how to display the contents of a page: colours, formats, positions, etc.

HTML Tags

- Tags denote markup **elements**
- Each tag is surrounded by angle brackets < >
- Tags normally come in pairs:
  - the **opening tag** and the **closing tag**
- Tags are **not** case sensitive
  - <html> and <HTML> are functionally the same
  - Recommended to use lowercase
- Text between the tags is the **element content**
- `<h1> ... </h1>` Section 1
- …
- `<h6> ... </h6>` Section 1.1.1.1.1.1

Tag Attributes

- Tags can have associated attributes (or properties) that provide extra information to the browser
- Attributes consist of **`name="value"`** pairs
- Attributes are always added to the **opening tag**
- For example, to colour the text of heading 1 tag to red:
  - *Set the attribute style to the value of **color:red***
  - `<h1 style="color:red"> Section 1 </h1>`

Section 1

Tag Attributes cont'd

- Attribute values normally enclosed in quotes
  - Double quotes are the most common
  - `style="color:red"`
  - Single quotes are also allowed
  - `style='color:red'`

Be careful with copy and paste

- NB: If the value contains one type of quote, then the other type should be used to enclose the value
  - `name="John's Place"`

Common Tags - Text

- Bold

  `<b>Bold Text</b>` **Bold Text**

- Italic

  `<i>Italic Text</i>` *Italic Text*

- Paragraph

  `<p>Text</p>` Defines a paragraph

Exercise 2

1. Extend your web page to include two paragraphs of dummy text
2. Italicise any words beginning with *s*
3. Bold any words beginning with **a**

- Text
- "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat."

- "Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

Common Tags - Continued

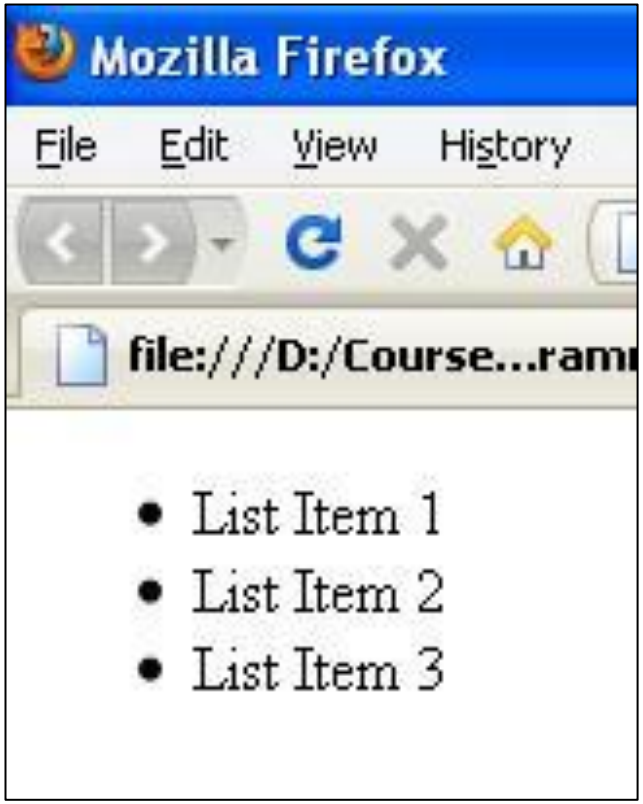
- Line Break
  - Forces a line break wherever it's placed
  - Putting a carriage-return into the HTML code will not produce a visible line break!
  - This
is a para
graph with line breaks
- Horizontal Rule <hr>
  - Can take width attribute <hr width="75%">
- Special Entities (start of a character reference)
  - Ampersand &
  - Copyright © ©

HTML Lists

- **U**nordered **L**ists `<ul> </ul>`
  - Each **L**ist **I**tem appears as Bullet `<li> </li>`
- **O**rdered **L**ists `<ol> </ol>`
  - Bullets replaced with numbers or letters
  - Type of order defined with type attribute
- **D**efinition **L**ists `<dl> </dl>`
  - Lists a **d**efinition **t**erm `<dt> </dt>`
  - and the **d**efinition **d**escription `<dd> </dd>`

Unordered Lists

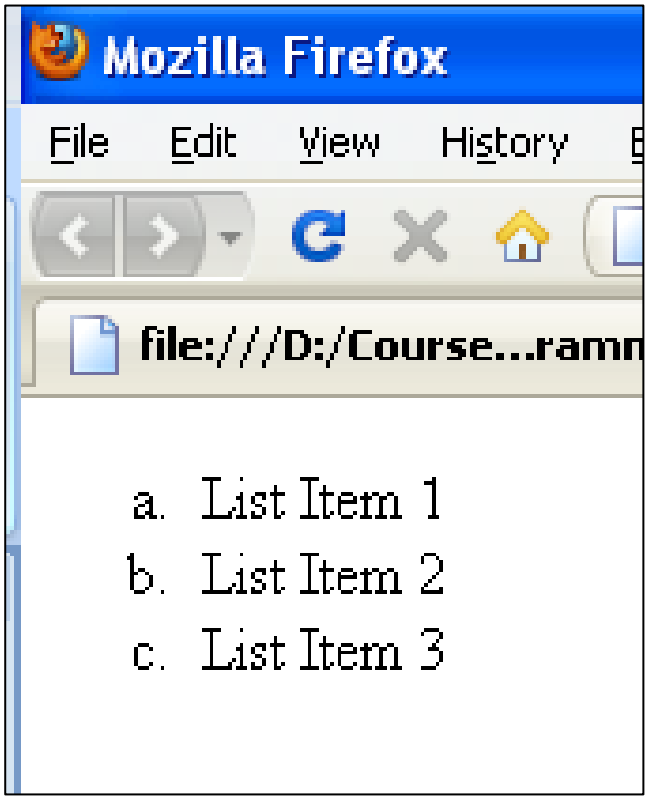

```
<ul>
        <li>List Item 1</li>
        <li>List Item 2</li>
        <li>List Item 3</li>
</ul>
```



- Bullet is default type
- Other types can be specified
  - "disc", "square", "circle", etc

Ordered Lists


```
<ol type="a">
      <li>List Item 1</li>
      <li>List Item 2</li>
      <li>List Item 3</li>
</ol>
```



- Numeric is default type
  - Other types can be specified
    - "1", "A", "a", "I", "i"

HTML Links (Hyperlinks)

- HTML Links created with **Link** Tags
  - `<a> </a>`
  - *`<a href="http://www.randomsite.com/">Visit Random Site.com</a>`*
- Hypertext Reference attribute gives the URL
  - `href="http://www.randomsite.com/"`
- Target attribute defines where the link opens
  - `target="_blank"` *Opens link in new window*
  - `target="_self"` *Opens link in same window*
  - `target="mywindow"` Opens in a named (possibly new) window

Exercise 3

1. Re-create the below simple FAQ for a fictional course
2. Use a heading 2 tag for the questions.
3. Use an ordered list for each question
4. URL for the Q1 link is https://www.universityofgalway.ie/science-engineering/school-of-computer-science/currentstudents/timetables/
5. URL for the Q2 link is https://www.universityofgalway.ie/science-engineering/school-of-computer-science/people/

1. **Where are the timetables for the course?**

   The course timetable can be found at the following link

2. **What are the contact details for academic faculty?**

   The programme director's info can be found at this location

Images in HTML

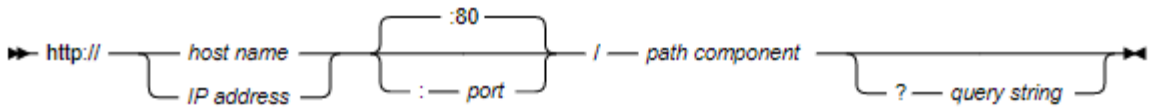
- Images are added with the empty tag <img>
- <img> requires a **source (src) attribute** with the URL of the image
  - <img src="http://www.randomsite.com/image.jpg">
- Image (and other) filenames are case sensitive on Linux and Mac servers
  - .jpg != .JPG
  - .png != .PNG

Absolute and Relative References

- URL:The path to a resource
- Any URL path in HTML can be given as either via an Absolute or a Relative Reference
- Both types are valid, interchangeable and useful in different situations
  - e.g. “a” tags and “img” tags can use both Reference Types

Structure of a URL (Absolute)

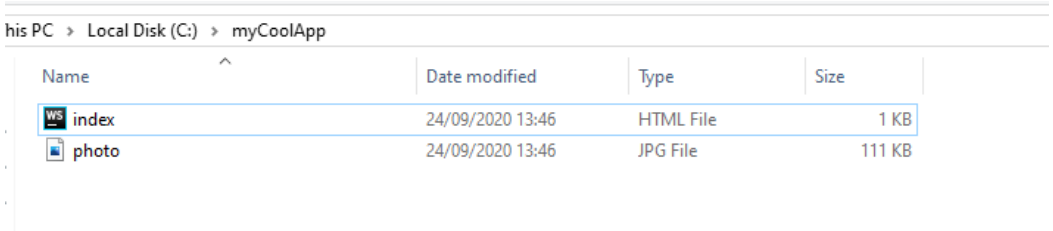
Figure 1. Syntax of an HTTP URL



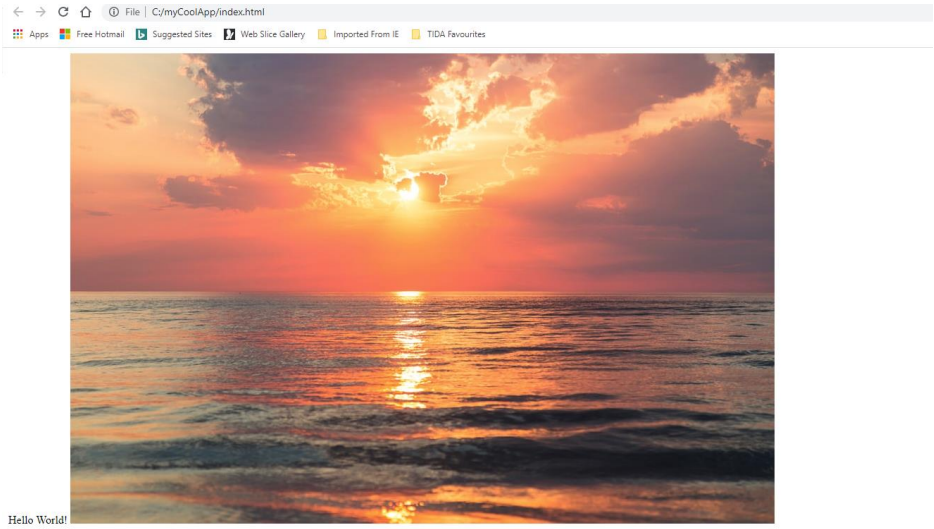
- http://www.example.com/software/index.html
- http://www.example.com:1030/software/index.html

Relative Reference

- If you wish to reference an image in the same folder you can do it via relative referencing




```
<!DOCTYPE html>
<html>
<head>
  <title>Simple
Page</title>
</head>
<body>
Hello World!
<img
src="photo.jpg"></img>
</body>
</html>
```



Exercise 4

1. Add an extra question to your FAQ page with an image of the CS building.
2. When you click on the image it opens a new tab to the CS homepage https://cs.universityofgalway.ie/.
3. Add a picture of the CS building, by including an absolute reference to an image already hosted online.
4. Also include an internal image of the building but this time relatively src it.

Final page should look something like this

1. **Where are the timetables for the course?**

   The course timetable can be found at the following link

2. **What are the contact details for academic faculty?**

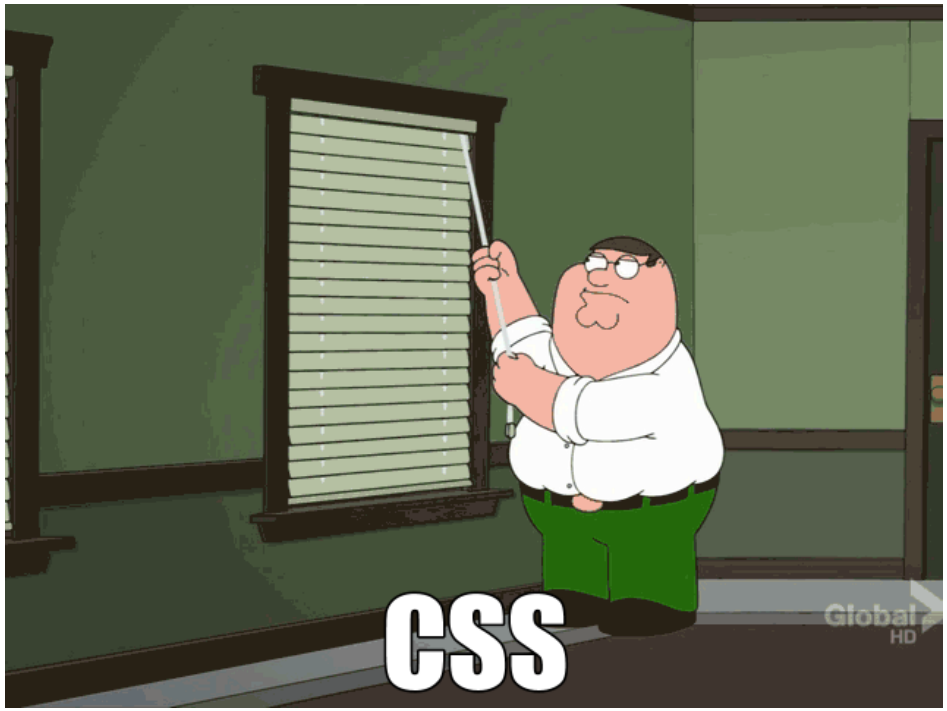
   The programme director's info can be found at this location

3. **What does the CS building look like?**

CASCADING STYLE SHEETS

What is CSS?

- Cascading Style Sheets
- Contains the rules for the presentation of HTML

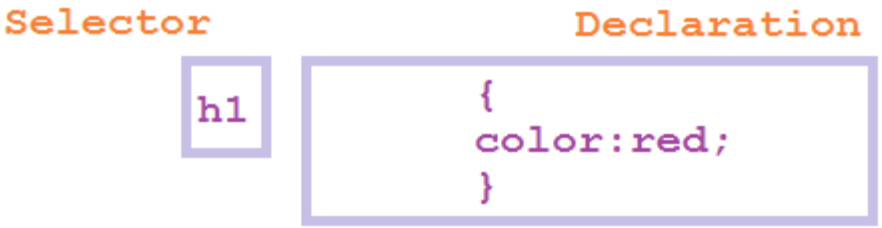


CSS Continued

- CSS allows the developer to define a set of styles
- These styles can be used across multiple pages
- Helps to create a uniform and consistent look across a web application
- Makes the formatting the content much easier
- Allows for more advanced techniques using div tags

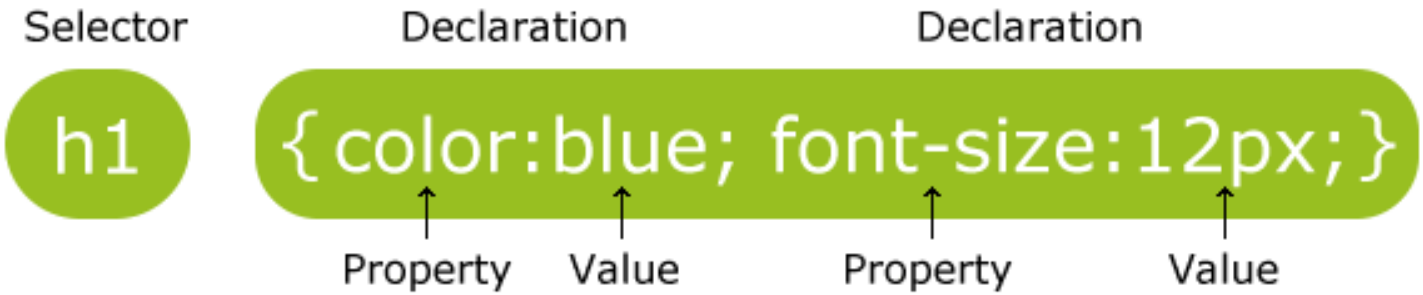
How does it work?

- CSS works by allowing you to associate rules to the HTML tags
- These rules govern how these elements should be rendered
- A rule is defined by a selector followed by a declaration block



Syntax continued

- Example of styling a Heading 1 tag
  - Colour it to blue
  - Set the text size to 12 pixels



Syntax continued


```
h1
{
 color: blue;
 font-size: 12px;
}
```

Note:

`color` **not** `colour`

Braces { } separate the selector from the declarations

Each declaration ends in semi-colon ;

Syntax continued


```
h1
{
color: blue;
font-size:
12px;
}
```

Note:

- **No space** between value and units (12 and px)
- **No quotes** for values

Sources of styles

- Inline styles
  - `<h1 style="color:red">Hello World</h1>` Hello World
- Embedded style tags
  -
```
<style>
        h1 {color:blue}
</style>
```
Hello World
- Linked styles
  - `<link href="/stylesheets/main.css" rel="stylesheet">`

Be careful with the styles

- Precedence or order over the styling
  - Browser default style (weakest)
  - User style sheet
  - Author stylesheet
  - Author embedded styles
  - Author inline styles (strongest)

- !important

Selecting a tag or element to style

- There are two main ways of doing this, class based selector and ID based

  - ID based (#)

```
<div id="content">
        My Text Content
 </div>
```

```
#content{
        width: 200px;
        }
```

  - Class based (.)

```
<div class="content">
        My Text Content
 </div>
```

```
.content{
        width: 200px;
        }
```

Font related CSS

- https://www.w3schools.com/cssref/pr_font_font.asp

Definition and Usage

The `font` property is a shorthand property for:

- font-style
- font-variant
- font-weight
- font-size/line-height
- font-family

```
<p class="p1">
        My paragraph of text
</p>
```

```
.p1 {
  font-family: "Times New Roman";
  font-weight: "600";
}
```

W3Schools have great docs on this

Exercise 1 – Adding styles

- Step 1: Add a file to the public folder of "MySampleApp" called styles.css
  - Add the following CSS to the stylesheet *styles.css*
    - A class based selector (".") called "links"
    - Set the font size to 16px
    - Set the font colour to blue
  - Add this linked stylesheet to your page (see slide 4)
- Step 2: Add 3 random hyperlinks to the web page
  - Associate this new class (created in Step 1) with all links on the page, does it work?
- Step 3: Using inline styling alter the colour of one of the links to change the font colour to green. Does it override the CSS in Step2?

The CSS Box model

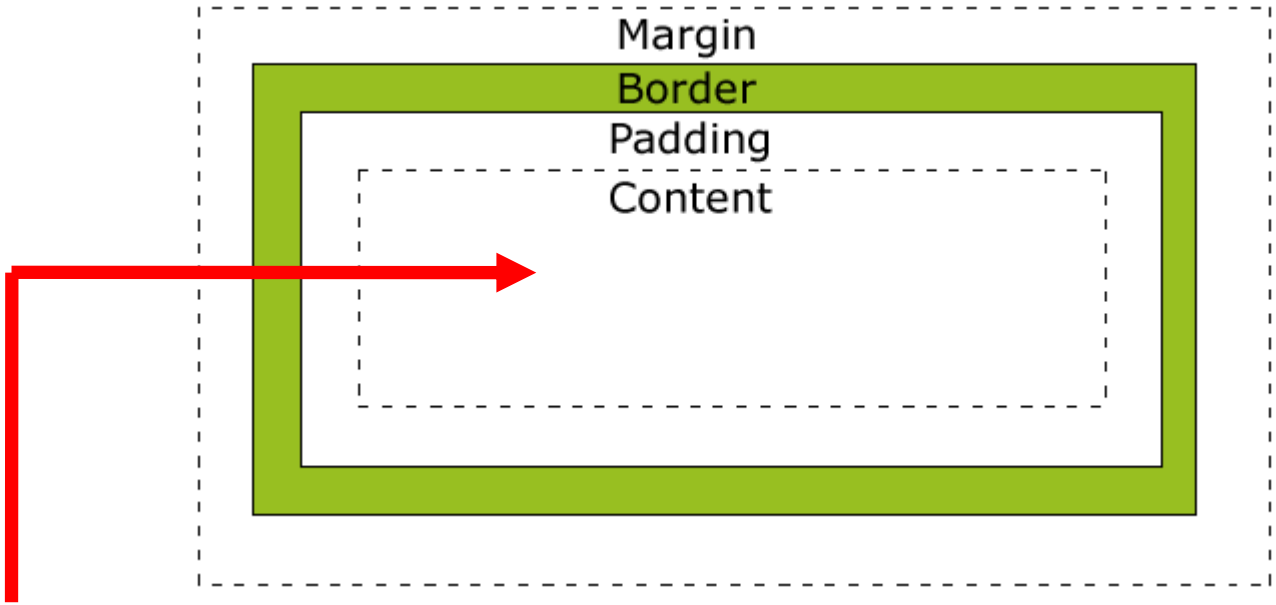
- All HTML elements are considered as boxes
  - Paragraphs, Headings, Tables etc.
- Box Model
  - A box that wraps around all the HTML elements
- Consists of Four Parts
  - Content
  - Padding
  - Border
  - Margin

CSS Box Model

- Content
- Padding
- Border
- Margin

- Padding, Border & Margin are ZERO by default

Content Area



- Content
  - The HTML element concerned
  - Text, Image, List, Table, etc

Content – Embedded <p> Example

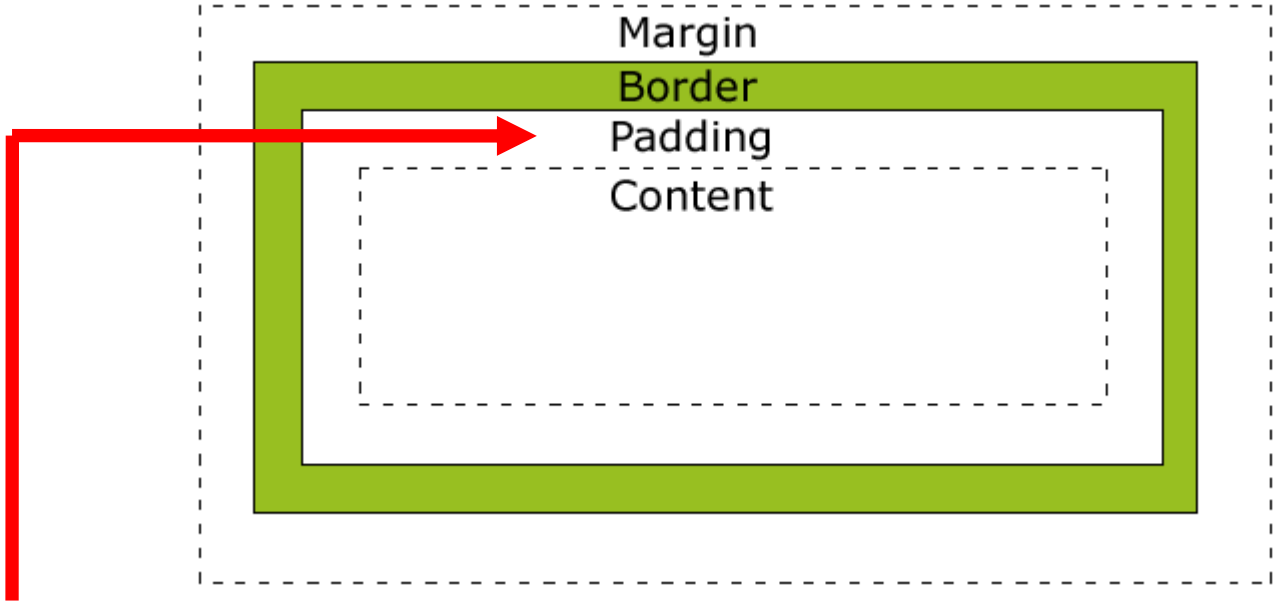

```
<html>
<head>
<style>
p
{
  background: #00FFFF;
}
</style>
</head>
```

```
<body>
<p>Text Goes Here!</p>
</body>
</html>
```

Result:

Text Goes Here!

Padding Area



- Padding
  - Empty space surrounding the Content
  - Uses the **same** background colour as the Content

Padding Content

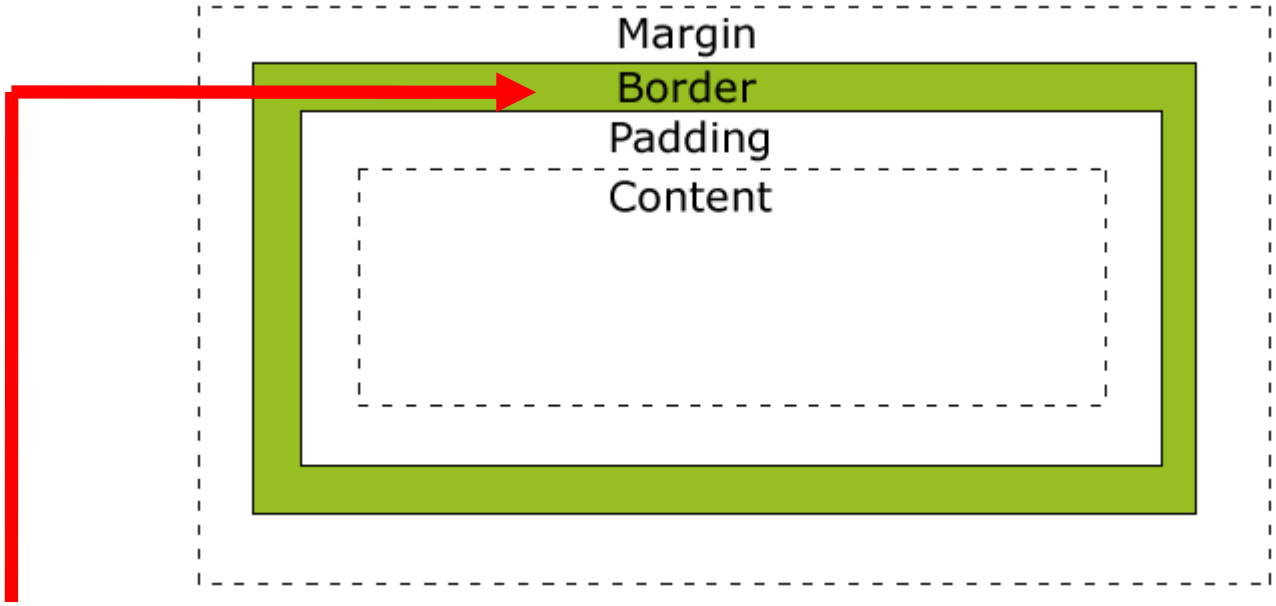

```
p
{
background: #00FFFF;
padding: 0px;
}
```

Text Goes Here!

```
p
{
background: #00FFFF;
padding: 20px;
}
```

Text Goes Here!

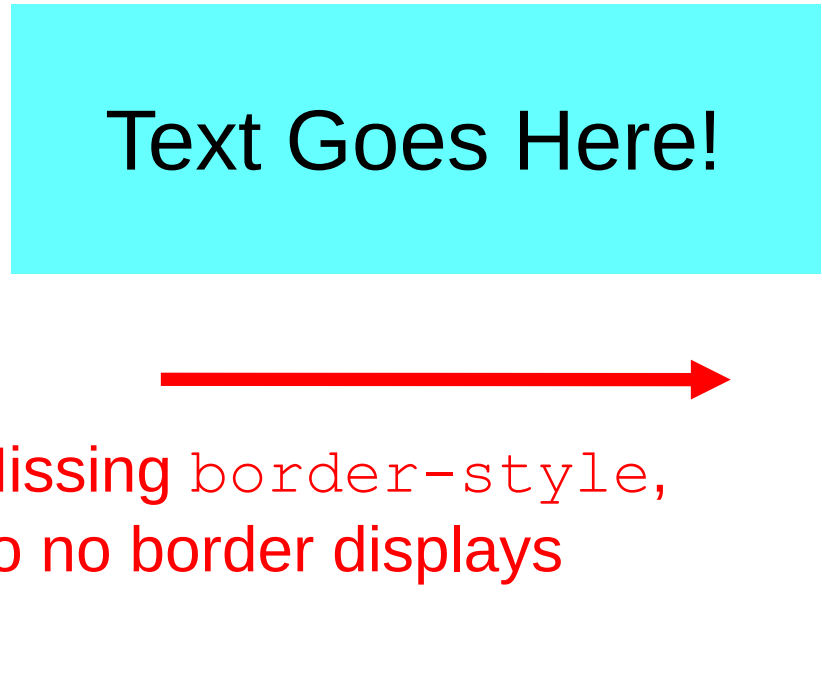
Border Area



- **Border**
  - The HTML element concerned
  - *border-style* **must** be set for border to take effect

Bordering Content


```
p
{
background: #00FFFF;
padding: 20px;
border: 20px;
border-color: #FF0000;

}
```

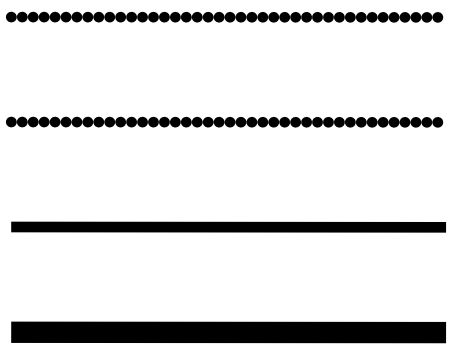


Bordering Content


```
p
{
background: #00FFFF;
padding: 20px;
border: 20px;
border-color:
#FF0000;
border-style: solid;
}
```

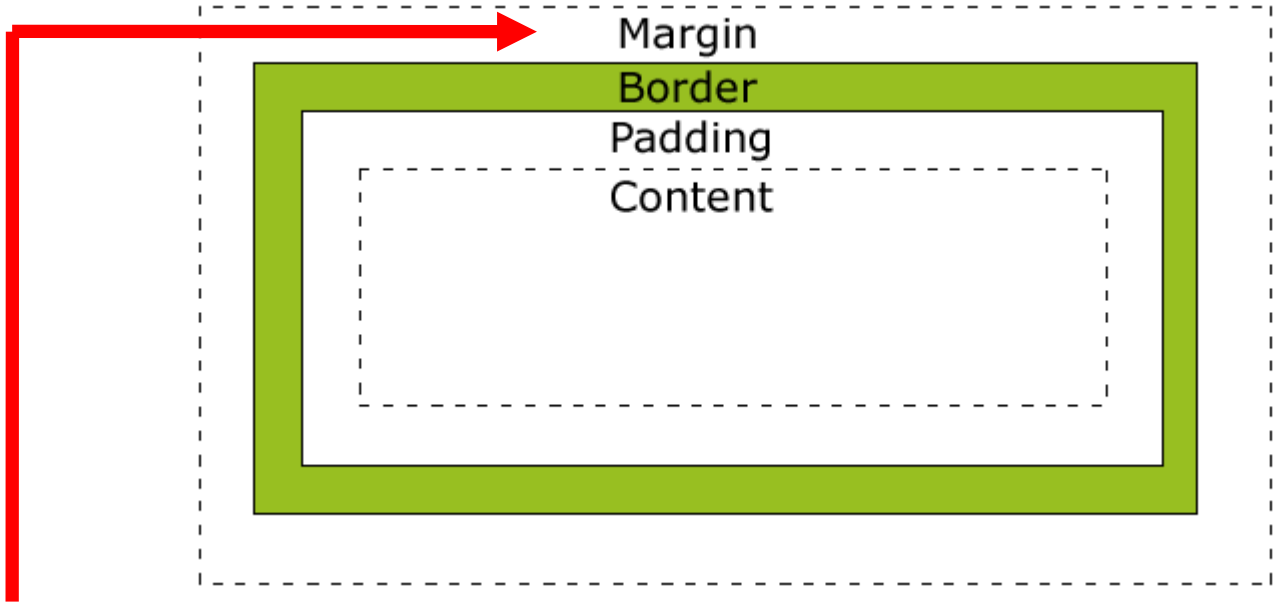
Border-style values

- none — *No border*
- dotted — *Dotted border*
- dashed — *Dashed border*
- solid — *Solid border*
- double — *Two solid borders*



- groove — *3D "grooved" border (engraved)*
- ridge — *3D "ridged" border (emboss)*
- inset — *3D "inset" border (lowered)*
- outset — *3D "outset" border (raised)*

Margin Area



- Margin
  - Transparent area that surrounds everything else
  - Used for spacing the element relative to others

Exercise 2

- Create a paragraph of text “Hello World”
- Add a background colour of your choosing using Hex values
- Add padding of 10px all around it
- Add a 1px dashed border around it of a colour of your choosing.
- Place a second paragraph below the first and style it the same way as the first however place a 10px margin between the two (either bottom of the first or top of the second).

Grouping and descendants

- Multiple selectors can be grouped in a single style declaration

```
p, .main{
        font-weight: bold;
}
```

- Select elements that are descendants

```
<div class="abc">
 <div>
  <p>
     My Text Content!
  </p>
 </div>
</div>
```

```
div.abc p{
        font-weight: bold;
}
```

CSS values

- *Text-align:center;*
- Numerical values: Numerical values are usually followed by a unit type.

  font-size:12px;
- 12 is the numerical value and px is the unit type in pixels
  - Absolute values :in, pc, px, cm, mm, pt
  - Relative values: em, ex, %
- Color values: color:#336699
  - Blue, red, green etc…

Colour Wheel [visible light]

- Combining different levels of RGB yields different colours
- Wheel on the right shows colour relationships
- Yellow is made of equal parts Re
  - Thus, HTML Colour for Yellow:
  - #FFFF00

The div tag

- Defines a division of a HTML page and often used as a container for other elements.
- Block level elements such as div's, paragraphs headings sit on top of each other, by default.

HTML

```
<body>
        <div id="div1"></div>
        <div id="div2"></div>
        <div id="div3"></div>
</body>
```

CSS

```
#div1 { width:300px;background:yellow;}
#div1 { width:300px;background:blue;}
#div1 { width:300px;background:orange;}
```

Inline elements

- Inline elements such as span and img sit side by side

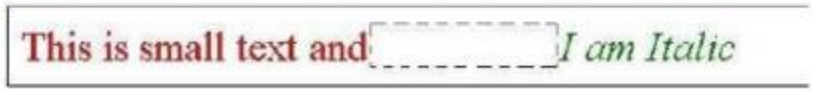

```
<div id="row1" >
   <span class="norm">This is
small text and </span>
   <span class="big">this is big</
span>
   <span class="italicText"> I am
Italic</span>
</div>
```

```
.norm {
        color:red;
}
.big {
        color:blue;
        font-weight:bold;
}
.italicText {
        color:green;
        font-style:italic;
}
#row1 {
        padding:10px;
        border:solid 1px #000;
}
```

Visibility

Visible : The element is visible (default).
Hidden : The element is invisible (but still takes up space)


```
.big {
        visibility:hidden;
}
```



Exercise 3 – Page styling

- Create two paragraphs of text
  - This is a paragraph of text
  - This is a paragraph of text
- Surround both paragraphs with a single div tag
- Create a style (using class-based selection) which centres the text of the paragraphs and changes the colour to #663399. Place the style in a separate styles.css file. Apply it to the div element.
- Within each paragraph use a span to change the word “paragraph” to yellow. Add the style to the styles.css file and using class-based selection apply it to both spans.

BOOTSTRAP

https://getbootstrap.com/

What is Bootstrap?

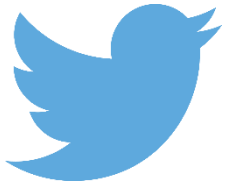
- Bootstrap is a CSS framework, it is CSS classes for structure, layout, components (buttons, navbar, modal etc), forms, written by other developers.
- This enforces a uniform layout, look and feel on the web application.
- Freely available to develop web sites and web applications.

What is Bootstrap cont.

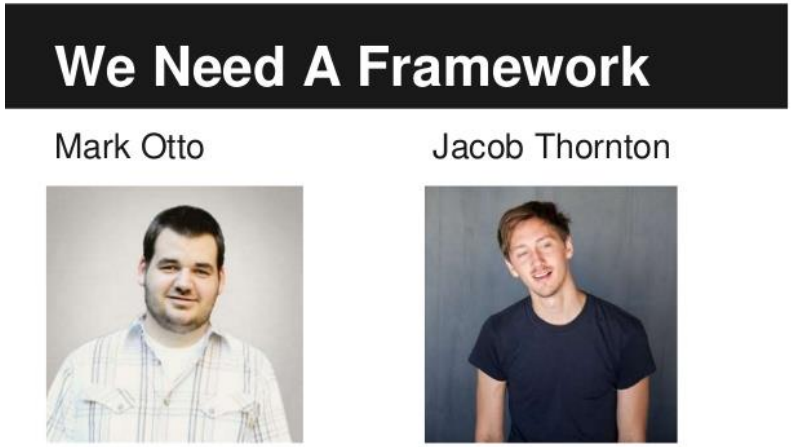
- JavaScript is also used in conjunction with the CSS classes, for things like animations, transitions, popups etc.
- The CSS within it is quite detailed, there are lots of classes with varying levels of hierarchy.
- It's fully customisable and the web is full of themes and templates for apps built using it
  - https://themes.getbootstrap.com/

Who developed it?

- It was developed by Twitter's Mark Otto and Jacob Thornton



- They wanted to standardise the frontend toolsets across twitter
- It was released as open source in August 2011 on Github



Adding Bootstrap to our web apps

- Two options
  - Download the files, i.e. CSS, JS, place them in the app folders
  - Use a Content Delivery Network URL

- https://getbootstrap.com/

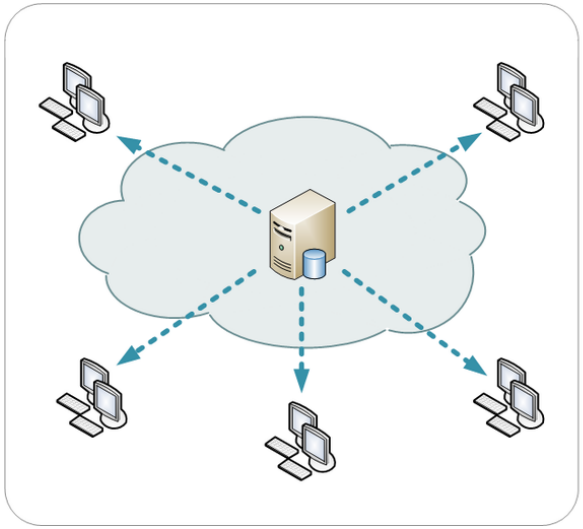
Content Delivery Networks

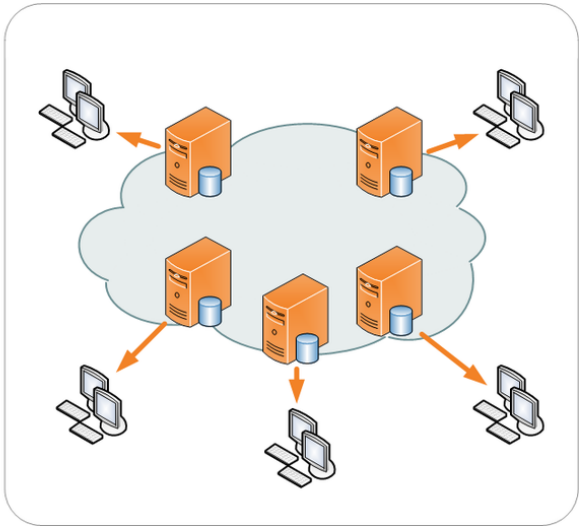
- A popular way to include frontend libraries and technologies is to use a CDN
- Instead of having a single server, CDNs involve using a collection of servers to serve content
- These servers are usually placed geographically close the user base to ensure that maximum performance is achieved
- Largely designed for delivering static content, images, videos, and web content such as text, graphics and scripts.

CDNs are popular

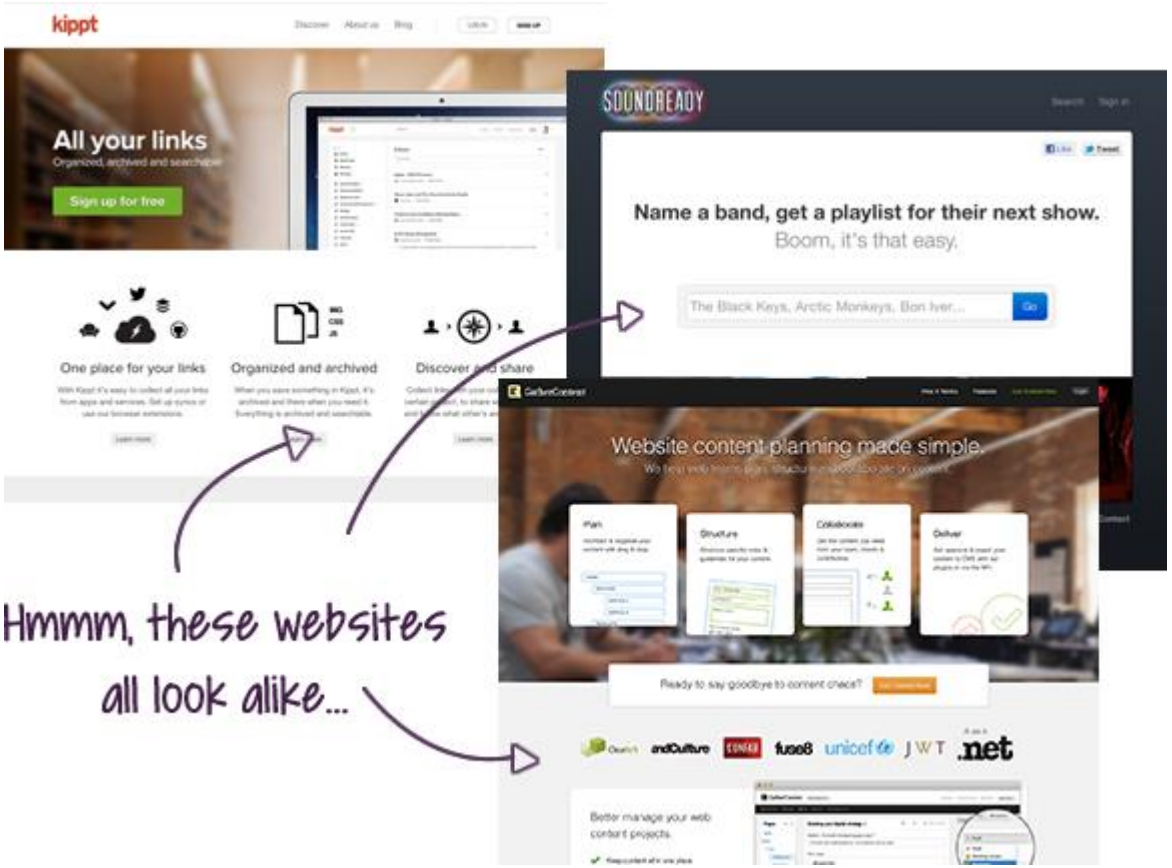
- Almost every frontend technology provider, Facebook, Google, Twitter will provide access to their libraries via a CDN

- This is very useful from a caching perspective, as browser caches should already have a hit for a regularly used CDN address, such as that from Bootstrap





How to recognise it?

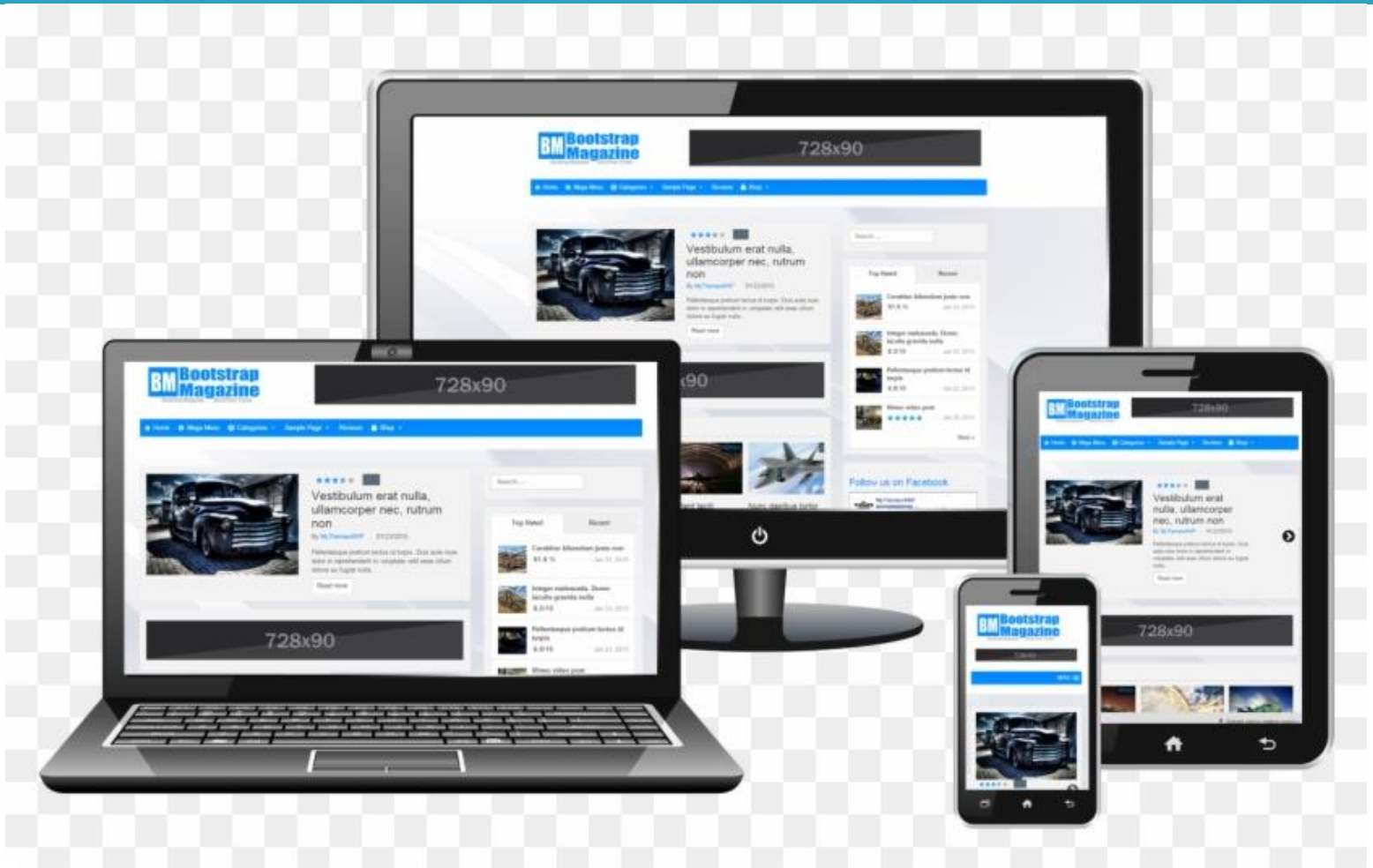


Supported by all browsers

Solid cross browser compatibility

All browsers have different
rendering engines, webkit, trident …

Responsive Design



https://www.pngfind.com/mpng/iJmbRw_bootstrap-responsive-design-laptop-tablet-mobile-psd-hd/

What is responsive layout?

- Produces an optimal viewing experience for the user independent of the device they are viewing it on
- Bootstrap is mobile first (software that has been developed to prioritise use on mobile platforms)
- If you view it on a mobile, tablet or larger screen it will scale accordingly
- As the viewport size increases it can scale up to 12 columns

https://www.tutorialrepublic.com/twitter-bootstrap-tutorial/bootstrap-responsive-layout.php

Get Bootstrap

- Navigate to https://getbootstrap.com/
- Scroll down to the Include via CDN

- Copy the CSS only link and paste it in between the <head></head> tags at the top of the page
- Take the JavaScript and pop it in below it.

Add Bootstrap to our Web pages

Index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title> Welcome to my cool new web page </title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
    <link rel='stylesheet' href='https://cdn.jsdelivr.net/npm/bootstrap@currversion…' />
    <script src='https://cdn.jsdelivr.net/npm/bootstrap@currversion…' />
  </head>
  <body>

  </body>
</html>
```

Add the bootstrap CSS URLs and JS URLs from getbootstrap.com

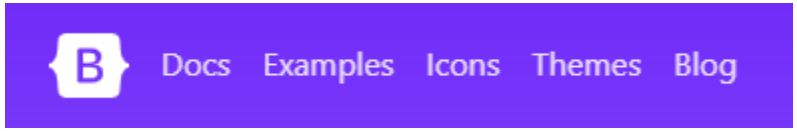
Bootstrap container class

- The container class is a fundamental building block of Bootstrap.
- They are required for the Grid system to work
- Thus it is best to ensure that all of your HTML elements marked up with Bootstrap classes are nested within a container

```
<div class="container">
…
</div>
```

Bootstrap Exercise 1 – Add bootstrap

- Add Bootstrap to your apps (see previous slide)
  - Test that it works by clicking on the Docs section (getbootstrap.com) and the choosing “Buttons” component
  - Paste the button samples into your HTML page (index.html)



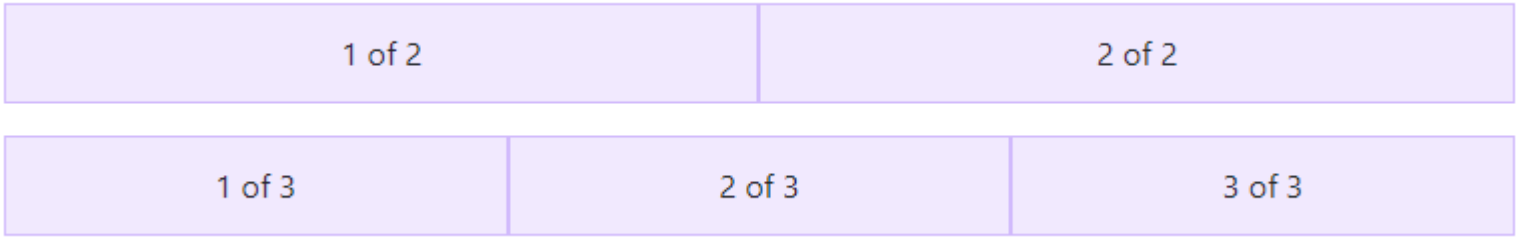
  - Does it look like the example in the docs, if not you haven’t included Bootstrap properly

  - Add a navigation bar to your app (consider where you might place this)

Bootstrap grid system

- For laying out content on your pages Bootstrap supports a grid system which structures the page into rows and columns

https://getbootstrap.com/



- It supports a concept of Rows and Columns, like a spreadsheet.
- Rows contain columns, columns contain the content!
- The total number of columns is 12.

Bootstrap grid system

- It uses div tags and with specific classes associated with each div.
- The rows and columns are all div tags
- It is built with Flexbox (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox)

How the “Grid” system works

- It supports six responsive breakpoints, which allows it to adjust the views for different screen sizes, **small, medium, large** etc.
- It has predefined padding and horizontal gutter widths between columns which are customizable but keep the content nicely spaced and uniformly structured with each other.

How the grid system works

- Rows must be placed within a **.container** class for proper alignment and padding.
- Use rows to create horizontal groups of columns.
- Content should be placed within columns, and only columns may be immediate children of rows.
- Predefined grid classes like **.row** and **.col** are available for quickly making grid layouts.
- Columns create gutters (gaps between column content) via padding. That padding is offset in rows for the first and last column via negative margin on **.rows.**
- Grid columns are created by specifying the number of twelve available columns you wish to span. For example, three equal columns would use three **.col-4** divs.

Bootstrap column classes

	xs <576px	sm ≥576px	md ≥768px	lg ≥992px	xl ≥1200px	xxl ≥1400px
Container `max-width`	None (auto)	540px	720px	960px	1140px	1320px
Class prefix	`.col-`	`.col-sm-`	`.col-md-`	`.col-lg-`	`.col-xl-`	`.col-xxl-`
# of columns	12					
Gutter width	1.5rem (.75rem on left and right)					
Custom gutters	Yes					
Nestable	Yes					
Column ordering	Yes					

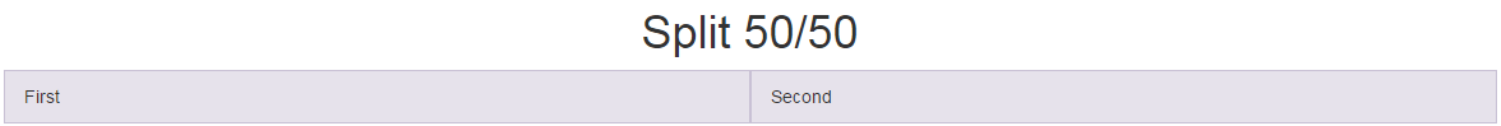
https://getbootstrap.com/docs/

Example

- Lets say you want to split the screen 50/50 for large screen devices

```
<div class="container">
    <h1>Split 50/50</h1>
    <div class="row show-grid">
        <div class="col-lg-6">First</div>
        <div class="col-lg-6">Second</div>
    </div>
</div>
```

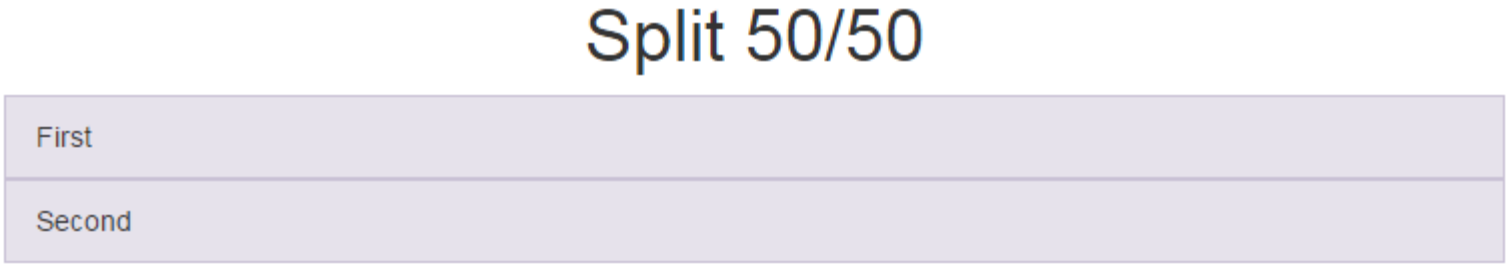
Remember everything is 12 columns wide
Note that this will affect breakpoints above it also



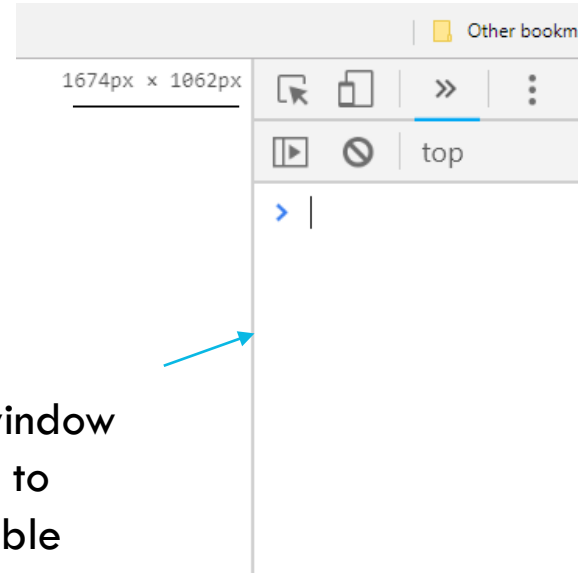
- On large screen sizes the columns will be placed side by side

Resize the screen

- When the screen is resized i.e. less than 992px it will stack the two columns



- You can find out the pixels by enabling the Chrome debugger



Keep them 50/50 on all devices

- If you want to keep them 50/50 on all devices you can specify the lowest size and it will scale

Practice it now!

Exercise 2 - Bootstrap – Grid

Provide the HTML and corresponding Bootstrap classes to split a row into two separate sections, each sized 6 columns wide on large and medium devices or above. In the case of smaller devices, the columns should stack on top of each other.

JAVASCRIPT

Conditionals and Loops

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Boolean Variables and Expressions

- Boolean literal may be either **true** or **false**
  - **let** isLoggedIn = true;
  - Note that true is a **value** not a string
  - *Do not enclose in quotes!*
  - Boolean expression is one which resolves to either true or false

Conditional Statements

- **if** statement – execute code only if some condition is true
- **if…else** statement – execute some code if the statement is true and another piece of it is false
- **if…else if… else** statements – used to execute one of many blocks of code

Conditional Statements JS


```
// If statement
if(true)
{

}

//If else statement
if(true)
{

}
else{

}
```

```
// If else if else
if(true)
{

}
else if(false){

}
else{

}
```

Relational operators

- Less than <
- Greater than >
- Less than or equal to <=
- Greater than or equal to >=
- Equal to ==
- Not equal to !=

Exercise 4: Conditional Example

- Write code that checks whether a given salary is well paid or poorly paid using conditionals.
  - Create a variable called *salary* and set it to 5000
  - Create an if statement which checks if *salary* is greater than 100000 and alerts "Wealthy Salary"
  - If less than 10000 alert "Poor Salary"
- What happens if you input 30000?

Logical Operators

- AND (&&)
  - Compares two Boolean values and equates to true if they are both true
- OR (||)
  - Compares two Boolean values and equates to true if one or the other (or both) is true
- NOT(!)
  - Negates the value of a Boolean value

Exercise 5: Conditional Example

- Extend exercise 4 to check if the salary is within 100,000 and 10,000 if so then alert “Normal Salary”

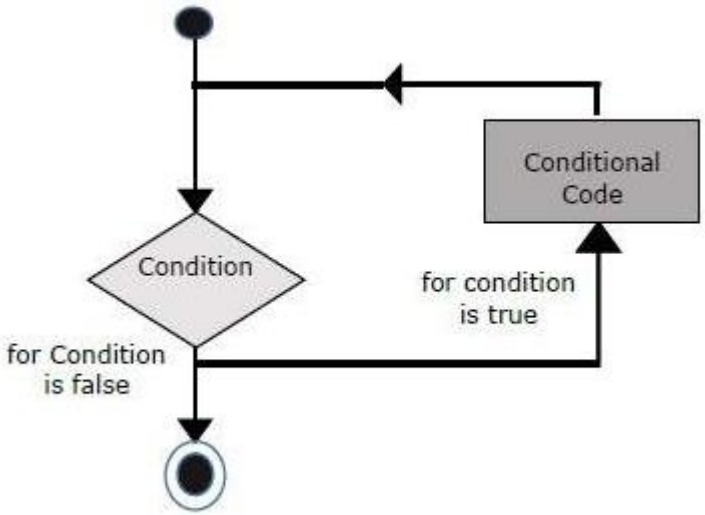
Loops in JavaScript

- Repeat a block of code
  - While some condition holds true
- For loops
  - Execute code a specific number of times
- While loops
  - Execute code an undetermined number of times

For Loop

- The For Loop repeats a block of code a certain number of times

```
//Repeat while i is still
less than 10
for(let i=0; i<10; i++)
{
    // Execute code
    // i++ at the end
}
```



https://www.tutorialspoint.com/javascript/javascript_for_loop.htm

While Loop

- Executes code an undetermined number of times
- Loops while the condition remains true
- Condition must be true for code to execute

```
while(i<j)
{
    // Execute this code
}
```

Break and Continue

- break;
  - Exits a loop immediately when it is encountered
- continue;
  - Stops the current execution of a loop
  - Does not exit the loop
  - Goes to the top of the loop for the next iteration
- Both can be used on any loop construct

Exercise 6: Loops

- Create a loop, either for or while, that alerts the even numbers between 1 and 10;

INTRODUCTION TO JAVASCRIPT

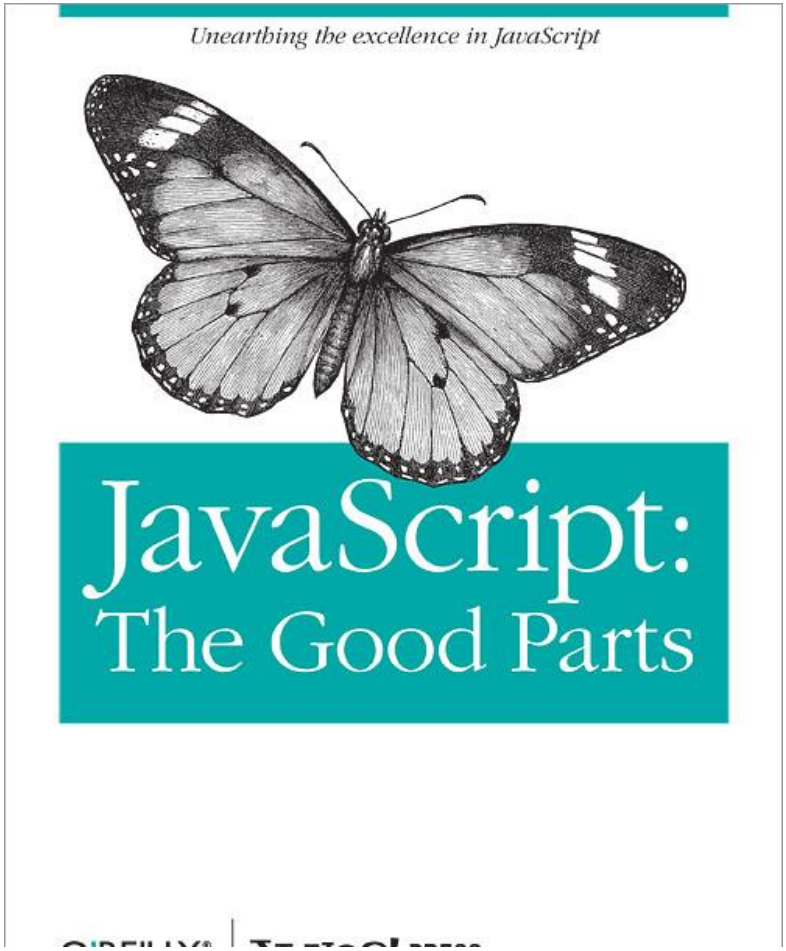
Client-side JS

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

JavaScript

- Best text book for learning JS

- Douglas Crockford

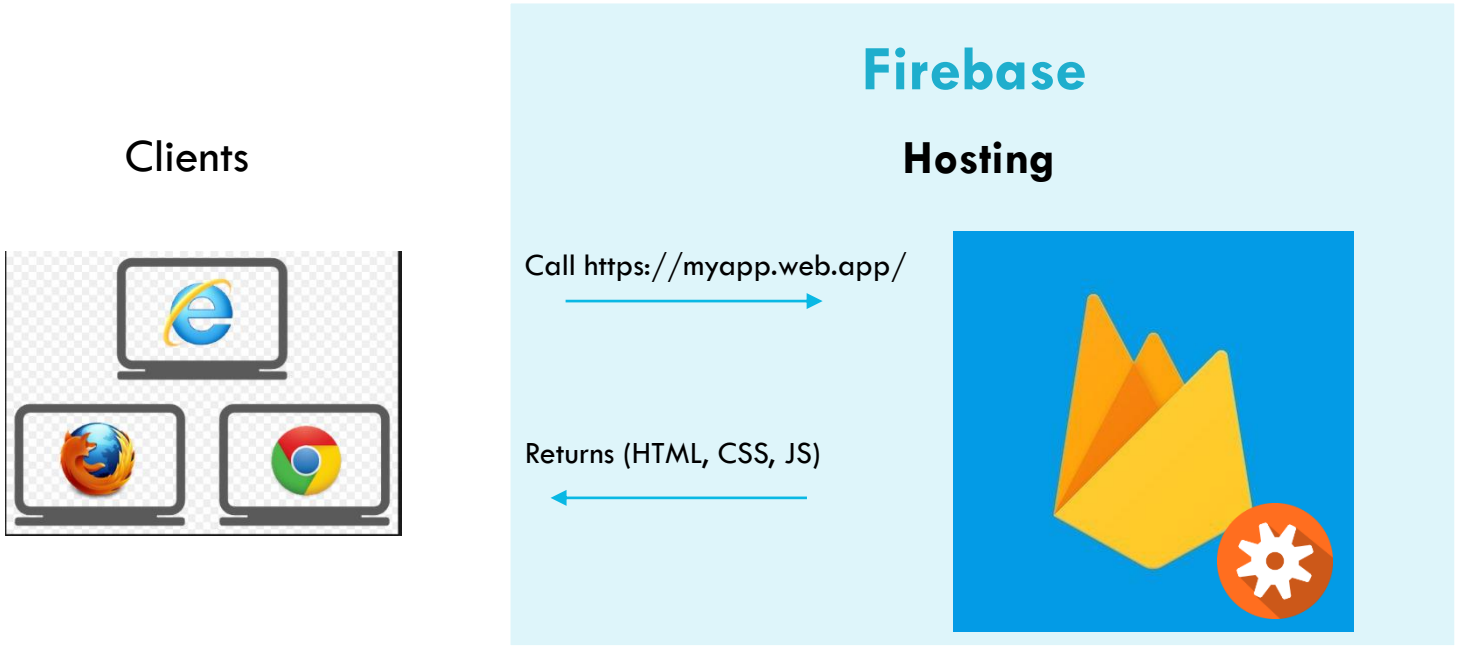


Overview

- Introduction to JavaScript
  - Adding client side JavaScript to our web pages
- JavaScript fundamentals
  - Variables
  - Types
  - Numbers
  - Strings
  - Booleans

Client side JS

- Client side JS
  - JS code that runs **locally** on the users machine/device (in the browser)



Client side JS cont.

- It allows you to manipulate/update the HTML content based on the users actions
  - If the user clicks a button “read more”, you can expand the content on the page. – News Blog
  - If the user continues scrolling further down, you can send a request to the server for more data. – Twitter feed
  - Sections of the page can be updated without reloading the entire page. – Dynamic Dashboards
- Brings an interactivity to what are essentially static HTML pages

JavaScript Syntax

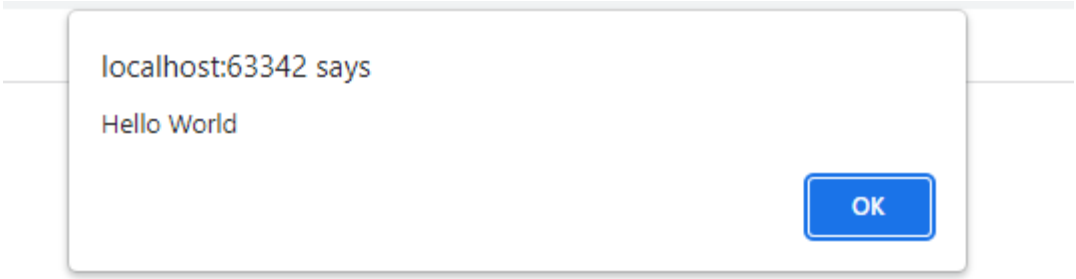
- 'C like' language
- Braces used to denote code blocks {} – like Java
- Semi-colons used at the end of lines
  - If you leave them out it will still compile, semi-colon insertion is automatically done during parsing so in most cases your code won't break if you leave them out.
- Comments as per C
  - Single line comments // Single line
  - Block comments /* Block comments
    can span multiple lines */
- Comments in HTML
  - <!-- This is my comment -->

JavaScript on HTML pages

- Similar to CSS we can include client-side JavaScript on HTML pages via the script tags

Inline JavaScript

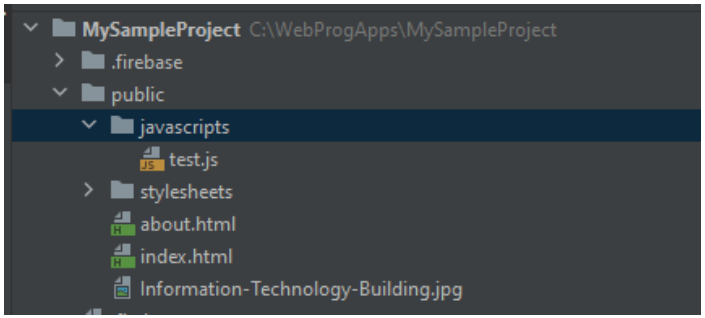
```
<script>
    window.alert("Hello World");
</script>
```



Inline vs External

- As with CSS where we can embed the CSS within the page or define a separate stylesheet, JavaScript also supports this.
- You can create **external JavaScript files** and using the **src** attribute on the script tag reference the URL

```
<script
src="javascripts/test.js"></script>
```



The Window Interface

- A Window Interface represents a window containing a HTML document.

- A window object is exposed to your JavaScript code and you can call various methods
  - window.alert("string") //displays a popup alert box
  - window.prompt("string") // displays a prompt where values can be entered by the user
  - window.confirm("string") // Confirmation box, allowing one to figure out what the user has pressed

Activity 1: Add some JS to our apps

- Create a new JS file. Pop it into the JavaScripts folder in your app (you will have to create this folder too)
- Write the following line in the JS file alert("Hello World");
- Save the file as hello.js
- Include the script on the page "hello.js" to the folder <mySampleApp>/public/javascripts

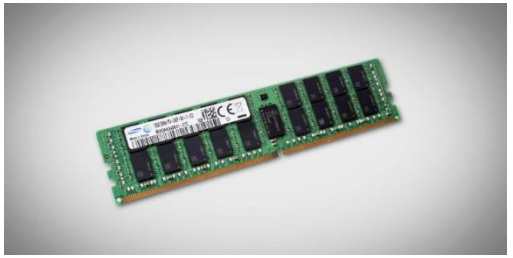
JAVASCRIPT

Variables - Numbers, Strings and Booleans

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Variables (Numbers)

- Variables are the names you give to computer memory locations which contain data in your computer programs
- Created using the **let** keyword (Let was introduced in ES6 2015)
  - **let** age = 24;
- Variables can also be created using the **var** key word but this essentially makes them globally scoped which can be problematic.
  - **var** age = 24;
- If you don't want the variable to change then use **const**
  - **const** age = 24;
- Unlike languages such as C, variables do not need to be declared before being assigned a value
  - **let** age = 24;
  - Variable "age" will be created if it isn't already
- Declare multiple variables
  - **let** age1, age2, age3;



Arithmetic in JavaScript (Numbers)

- Similar to C / C++ / Java etc.
- Basic **Arithmetic Operators**

Addition	+	ans=a+7
Subtraction	-	ans=a-b
Multiplication	*	ans=a*b
Division	/	ans=a/b
Modulus	%	ans=a%b

Exercise 2: Working with numbers

- Declare a variable called *salary* and assign it a value of 40000

- Assume that the €40k is your salary and you've just been awarded a bonus of €1000, so add this to the *salary* and using an alert display it on the page.

Variables (Strings)

- A **string** is a group of characters
- A **string literal** is a group of characters enclosed in quotes
  - “This is a string literal”
  - “This is too”
  - This isn’t
- A **string variable** is a variable that holds a string
- A **String** is a data type in JavaScript
- Space is a valid character in a string

String operators

- JavaScript supports string operators to join two strings together
- **let** name = “Enda ” + “Barrett”

- Can also concatenate string variables

```
let first, last, full;
first = “Enda ”;
last = “Barrett”;
full = first + last;
```

Variable types

- Variables in JavaScript are not associated with any particular type and any variable can be assigned (and re-assigned) values of all types.
- JavaScript supports dynamic or weak typing
  - This means that it will resolve the appropriate type at compile time, based on the input values
- **This does not mean that types don't exist, they do.**
  - Number.
  - BigInt
  - String.
  - Boolean.
  - Null.
  - Undefined.
  - Symbol.

```
let foo = 42; // foo is now a number
foo = "bar"; // foo is now a string
foo = true; // foo is now a boolean
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures

Exercise 3: Add string and number

- Create a variable *salary* and assign a value of 40000
- Create another variable *bonus* but assign it a value of "1000"
- Create a third variable called total which adds *salary* and *bonus* together
- Try adding 1000 euro bonus to the salary, what do you notice?

Adding a string to a number

- When you try to add a string and a number together the JavaScript JIT compiler will try to complete automatic type conversion. In this instance it converts the *number* 40000 to a *string* and concatenates it to the *string* bonus of “1000”.

Boolean Variables and Expressions

- Boolean variables are a special type of variable which can have two specific states **true** or **false**

- Declaring a Boolean variable
  - **let** isVar = true;
  - Note that **true** is a **value** not a string
  - *Do not enclose in quotes!*
  - Often pre-fixed with the naming convention "is"

Booleans

- Booleans are typically used in expressions, to evaluation one thing against another.

- If you wanted to check whether one number was larger than another number, you would place the two numbers within an expression and the result would be either **true** or **false.**

- Which leads to conditionals!

Variable Naming Rules

- Variable **names** known as **identifiers**
- **Identifiers** are case sensitive
  - myAge
  - MyAge
  - myAGe
  - All different variables above
- Must begin with a letter or underscore
- Cannot contain spaces
- Cannot use keywords

```
let age = 20;
```

JAVASCRIPT

Functions

Functions in JavaScript

- A JavaScript function is a block of code designed to perform a particular task.
- The function is executed when "something" invokes it (calls it)
- f(x) = x + 2

```
function multiply(param1, param2)
{
        return param1 * param2
}
```

Function won't execute unless it is invoked

Functions in JavaScript

- A function can be named or it can be anonymous

```
function(param1, param2)
{
        return param1 * param2
}
```

- The params are items of data that the function needs to perform its task
- Not passing a required parameter will result in an error
- Can have zero parameters but still requires empty parameters

ES6 Arrow functions

ES6

```
(param1, param2) =>
{
        return param1 * param2;
}
```

ES5

```
function(param1, param2)
{
        return param1 * param2;
}
```

- Arrow functions are more concise, developer can achieve the same functionality with fewer lines of code.
- Support concise function expressions

Assign function to a variable


```
let multiply =  (param1, param2) =>
{
   return param1 * param2;
}
multiply(3,3)
```

```
let multiply = function(param1, param2)
{
   return param1 * param2;
}
multiply(3,3)
```

Returning from a function

- Functions can return values to their calling environments

- Use the ***return*** statement to do this

```
function divide(numerator, denominator)
{
        return numerator / denominator
}
```

Returning from a function

- The returned value from a function can be assigned to a variable.

```
function incrementAge(myAge)
{
    myAge++;
    return myAge;
}

let incAge = incrementAge(26);
alert("Incremented age is " + incAge);
```

Invoking the function and passing arguments

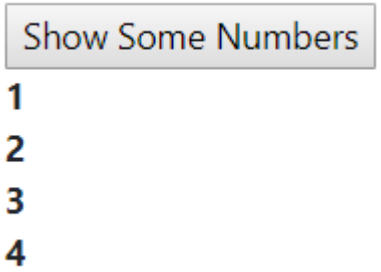
Functions

- Functions consist of
  - Unique name (cannot be keywords)
    - If they are named!
  - Parameters (again cannot be keywords)
  - **Don't** declare variables as parameters (**let** param1)
  - Code block to execute
  - Will only return once, but can use conditional statements to control the execution of the code and have multiple return statements

Invoking a function from HTML


```
<script>
function numbers() {
   let sHTML = "<b>";
   sHTML += 1 + "<br>" + 2 + "<br>" + 3 + "<br>" + 4 + "<br>";
   sHTML += "</b>";
   clientSideContent.innerHTML = sHTML;
}
</script>
<button onClick="numbers()">Show Some Numbers</button><br>
<div id="clientSideContent">Something here</div>
```

DOM Manipulation



Inline JS within our pages
-Note we can also link to a JS file too!
-Try and reproduce the same functionality but this time by using a separate JS file

Exercise: Functions

- Copy the JS code from the previous slide and place it into a separate JavaScript file.
- Modify the function “numbers” to accept a param
- If the argument passed in is 1 then the numbers printed out should be 1, 2, 3, 4.
- If the argument passed in is something else, then the numbers printed out should be 2, 4, 6, 8

JAVASCRIPT

Events

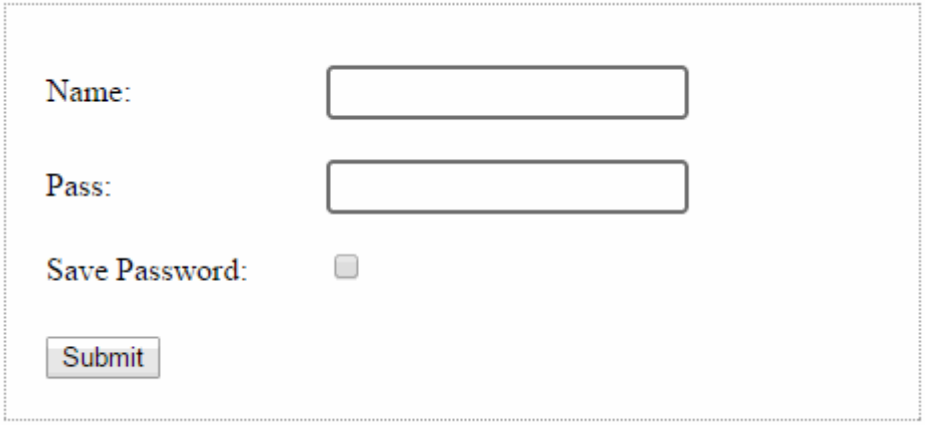
OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Events

- Actions that can be responded to by JavaScript
- Every element on a page has certain events which can trigger some JavaScript code
  - We can identify when a user clicks a button with the **onClick** event
  - Can then assign a function to run when the event is identified
  - Events defined as an attribute in the **HTML Tag**

Examples of Events

- A mouse click
- A web page or an image loading
- Moving the mouse over a hot spot on the web page



Other Mouse Events

- **onClick**
  - Triggered when the mouse clicks an element
- **onMouseDown**
  - Triggered when the mouse button is pressed
- **onMouseUp**
  - Triggered when the mouse button is released.

Selecting and De-Selecting Elements

- All three normally used with form input elements (text boxes, buttons etc.)
- onFocus
  - Triggered when an element gets focus
  - E.g. an element that is clicked is said to be in focus
- onBlur
  - Triggered when an element loses focus
- onChange
  - Triggered when the content of an element changes

As the mouse moves over HTML elements

- **onMouseOver**
  - Triggered for an element when the mouse cursor is moved over that element
  - e.g. moving the mouse over an image ('rollover')
- **onMouseOut**
  - Triggered for an element when the mouse cursor is moved away from that element
  - e.g. moving the mouse out of the image

Other Keyboard Events

- onKeyDown
  - Triggered when a keyboard key is pressed
- onKeyUp
  - Triggered when a keyboard key is released
- onKeyPress
  - Triggered when a keyboard key is pressed or held
- onSelect
  - Triggered when text is selected

Exercise: Multiply numbers from text input

- Place a text input on your web page
- Ask the user to pop in a number into the textbox using the placeholder attribute
- Place a button underneath it called “Multiply”
- When the user clicks on the button a function is invoked which takes the value from the text box, multiplies it by 3 and pops it back into the same text box
- Note that if you place an ID on the input, you can access the value the user enters via myelement.value

JAVASCRIPT OBJECTS

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

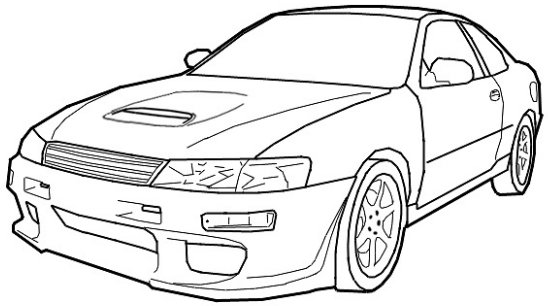
Objects

- We have seen how we can define primitives and store information using variables
- For example:
  - `let num = 123;`
  - `let str = "Enda B";`
- What if we want to represent something a little more abstract?

How would you represent these?

Sales person

Smartphone

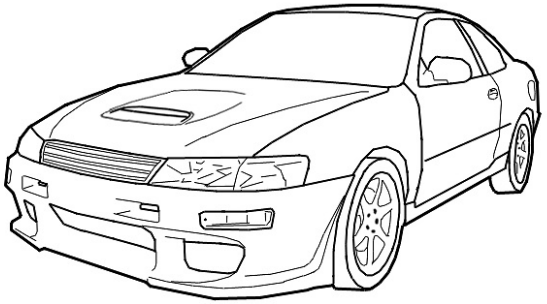


Car

Objects / User Defined Objects

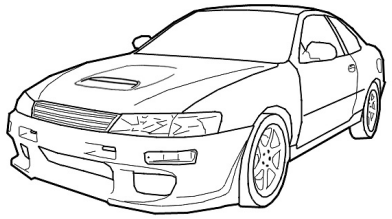
- Sales person
  - Name (String)
  - Phone number (Number)

- Car
  - Make (String)
  - Model (String)
  - Age (Number)



Object Example

- Objects are variables too, but contain many values
- **let** car = {make: "Ford", model: "Mondeo", age:2};
- The values are expressed in a *property:value* format



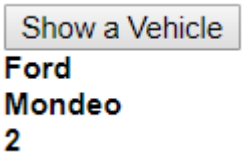
Accessing object properties & setting values

- The values can be accessed in two ways
  - objName.property or objName["property"]
  - **let** car = {make: "Ford", model: "Mondeo", age:2};

  - **let** make = car.make // make = Ford
  - **let** anoMake = car["make"] // anoMake = Ford

Simple example


```
<script>
function displayVehicles() {
   let vehicle = {make:"Ford", model:"Mondeo", age:2};
   let sHTML = "<b>";
   sHTML += vehicle.make + "<br>" + vehicle.model + "<br>" +
vehicle.age;
   sHTML += "</b>";
   clientSideContent.innerHTML = sHTML;
}
</script>
<button onClick="displayVehicles();">Show a Vehicle</button><br>
<div id="clientSideContent">Something here</div>
```



Exercise: Put into table

- Modify the previous example (slide 8) to display the contents of the single vehicle object in a table.

- Use Bootstrap to style it something like below (https://getbootstrap.com/docs/5.0/content/tables/)

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

JAVASCRIPT ARRAYS

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Arrays

- Arrays can store multiple values within a single variable.
- It is a special variable that is suited for storing lists of items.
- **let** array = [“Enda”, “Barrett”];

- **let** array = [];
- You define it using square brackets

Arrays cont.

- Each item within the array is known as an element
- **let** array = [“Enda”, “Barrett”];

- Array elements can be accessed by referencing the correct index position

- **let** fullName = array[0] + “” + array[1]

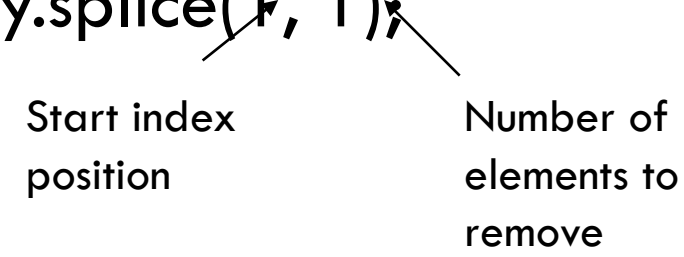
Adding array items

- When initializing an array you can add values but you may wish to add more throughout the program's execution
- This can be achieved by specifying an index position and assigning a value to that position.
- **let** array = ["Enda", "Barrett"];
- array[2] = "Peter";
- array[3] = "Devon";

Removing array items with splice

- **let** array = [“Enda”, “Barrett”];
- array[2] = “Peter”;
- array[3] = “Devon”;

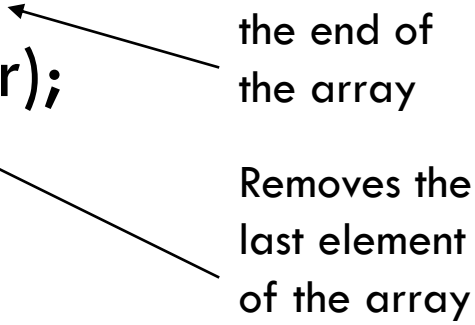
- array.splice(1, 1);



Arrays (push, pop) with Objects

- An array can also store objects
- **let** array = [];
- **let** car = {make: “Ford”, model: “Focus”, year: 2, mileage:10000}

- array.push(car);
- array.pop()



Exercise: Arrays

- Create an array of 5 objects such as vehicles etc.
- Each object has a make, model, year and mileage property.
- Populate it with mock data
- Using a loop iterate over the array of objects and place them in tabular form on your web page

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

INTRODUCTION TO NODEJS

AND SERVER-SIDE CODING

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Overview

- Introduction to NodeJS
  - Background on NodeJS
- Where do we run it?
- Server side coding
- What does the code look like?

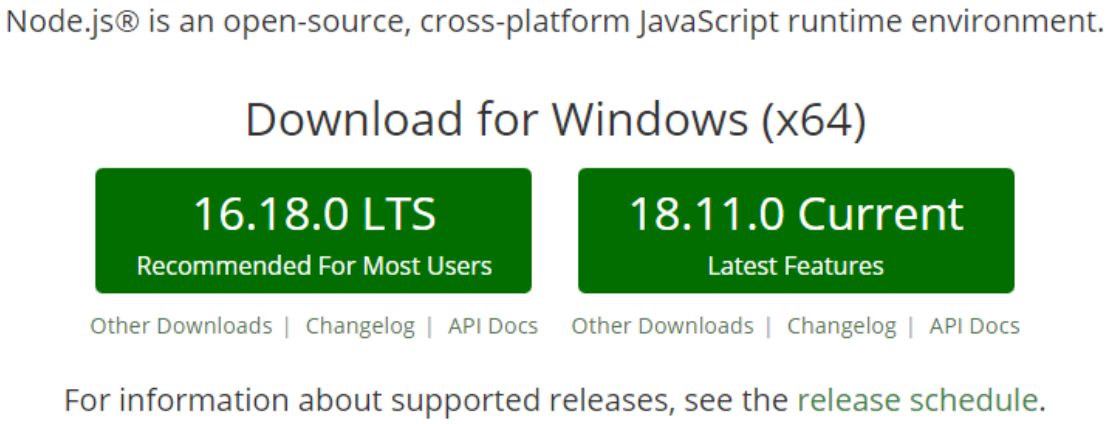
Background

- **V8** is an open-source **JavaScript engine** developed by Google. It's written in C++ and is used by the Google Chrome Browser
- Short video on V8
  - http://www.youtube.com/watch?v=hWhMKalEicY
- **Node.js** (Node) runs on the V8 engine
  - It's not a programming language – it's a runtime environment
- Write it in JavaScript
  - Most web dev's are used to writing in JavaScript

- Good News - It's just more JavaScript!

Background cont.

- It was created by **Ryan Dahl.**
  - Presented at JSConf 2009 – standing ovation
- You have downloaded it already!
- It is **Open Source.** It runs on both Linux, Mac and Windows operating systems.

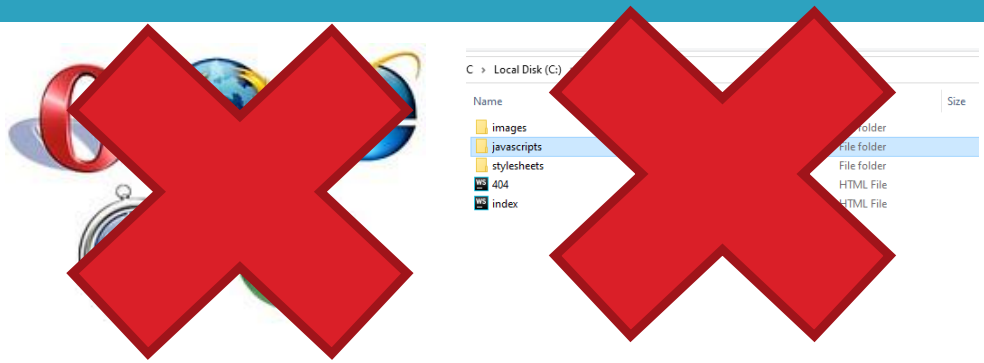


Introduction

- In simple terms Node.js is '**server-side JavaScript**'
- In not-so-simple words, Node.js is a high **performance network applications** application, well optimised for high concurrent environments
- In 'Node.js', '.js' doesn't mean that it's solely written in JavaScript. It is 40% JS and 60% C++.
- You can write modules for node in C++

Where do we run it?

- <script></script>
  - It's not for the browser



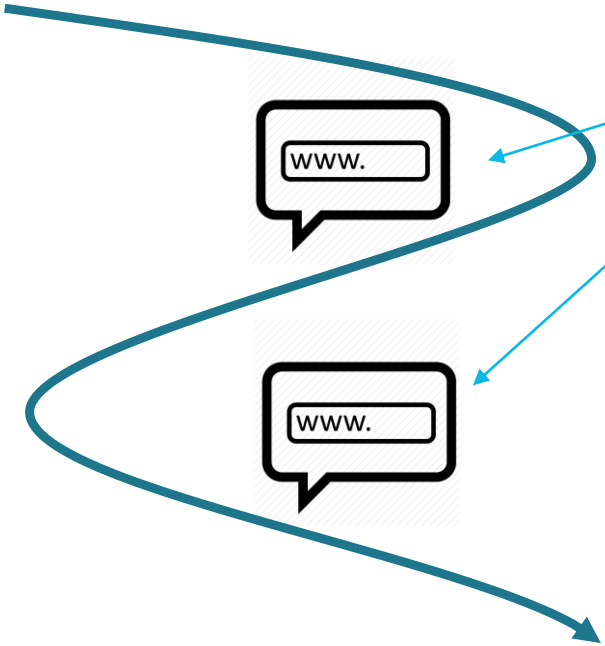
- Doesn't run on the user's laptop/phone or desktop computer!

- Where does it run?
  - On the server!

What's the server?

Bounces through the internet

Request messages from the client



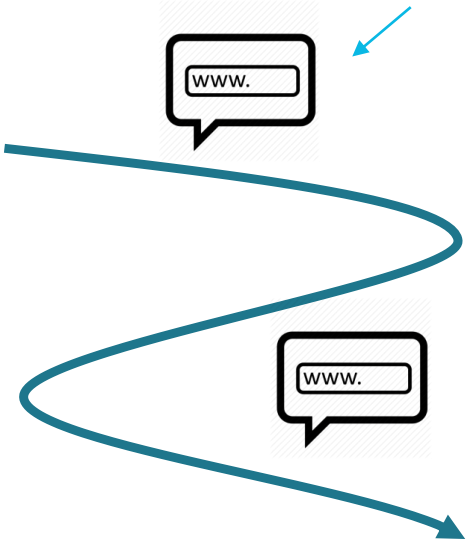
Firebase

Don't we already have this?

- Yes but at the minute all Firebase is doing is returning our static HTML pages
  - Our current HTML, CSS, and JavaScript is static, once we deploy it, it doesn't change
- Things we can't do right now
  - Save some user data to a database?
  - Register a user and store their username and password
  - Check if a user is logged in and if so allow them access to a restricted section of the site

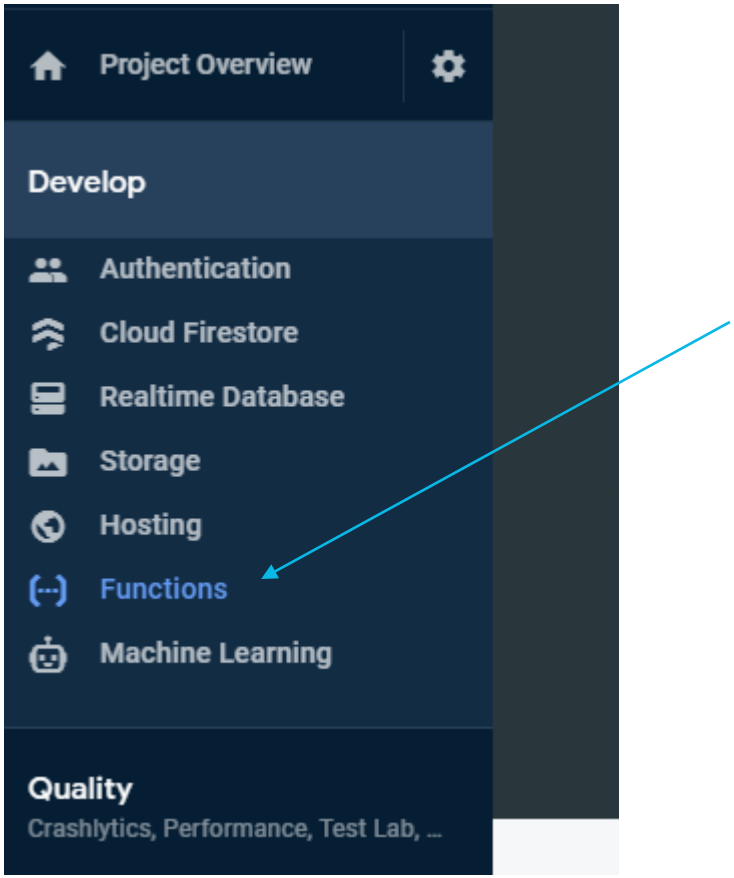
Server side coding

- In order to add the type of functionality listed on the previous slide we need to write "server side code"
  - Sometimes called backend code or a business logic layer
- It's just code that runs when the server receives a request from a client



What does the code look like?

Where does it run on Firebase



Summary

- Introduction to NodeJS
  - Background on NodeJS
- Where do we run it?
- Server side coding
- What does the code look like

REST APIS AND DEPLOYING TO FIREBASE FUNCTIONS

Overview

- Introduction to REST APIs
- Configuring Firebase functions on our apps
- Examining our first NodeJS code
- Modules in JavaScript
- Deploying our NodeJS functions to Firebase

REST API's

https://www.youtube.com/watch?v=7YcW25PHnAA&t=82s

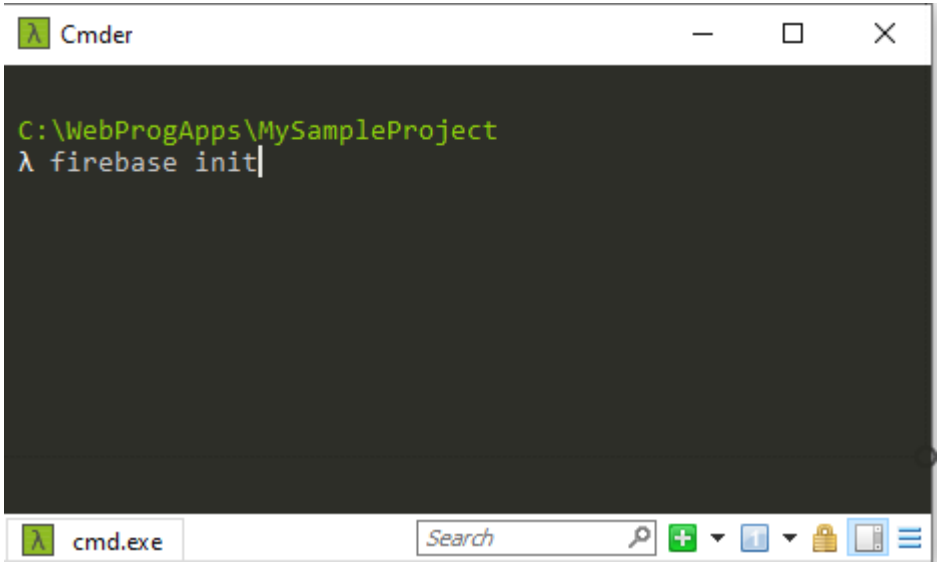
Application Programming Interface (API)

- An API expresses a software component in terms of its operations, inputs, outputs and underlying types
- RESTful APIs
  - **RE**presentational **S**tate **T**ransfer
- Use HTTP methods (verbs), GET, POST …

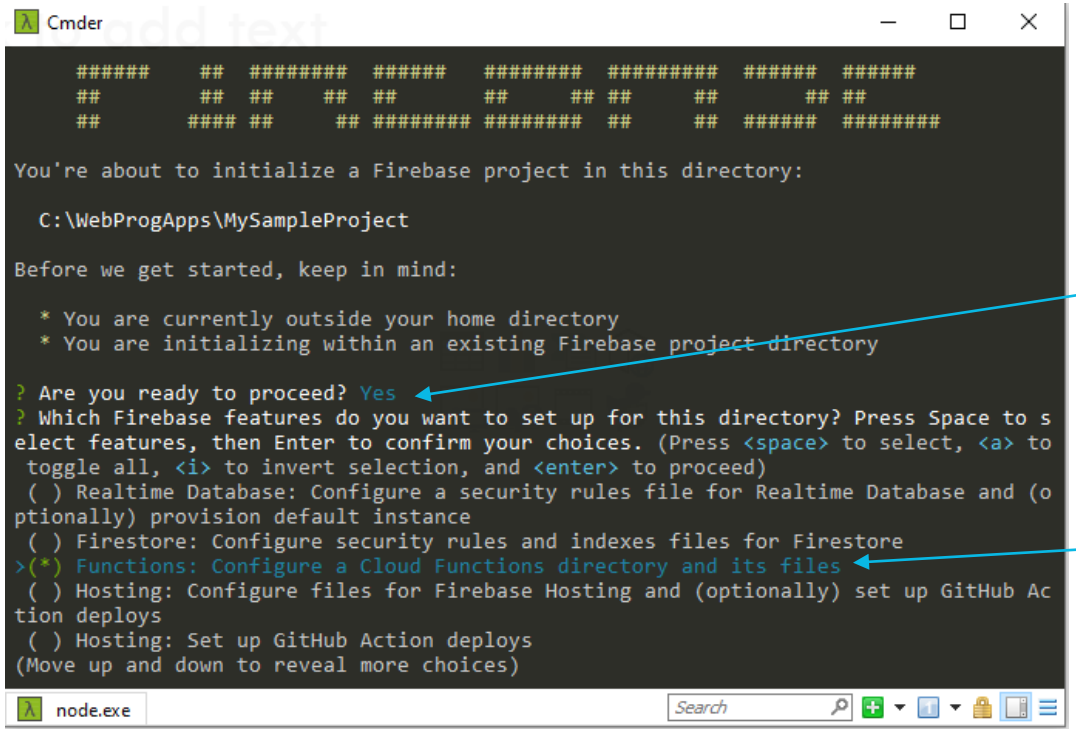
- Interface is exposed and can be invoked without caring about the underlying implementation
- Accessed via a URL

Adding Firebase Functions

- We are using Firebase functions to host our REST APIS
- Thus as when we setup hosting we must now initialise functions within our projects
- Using the command line, navigate to the directory containing your project
- firebase init



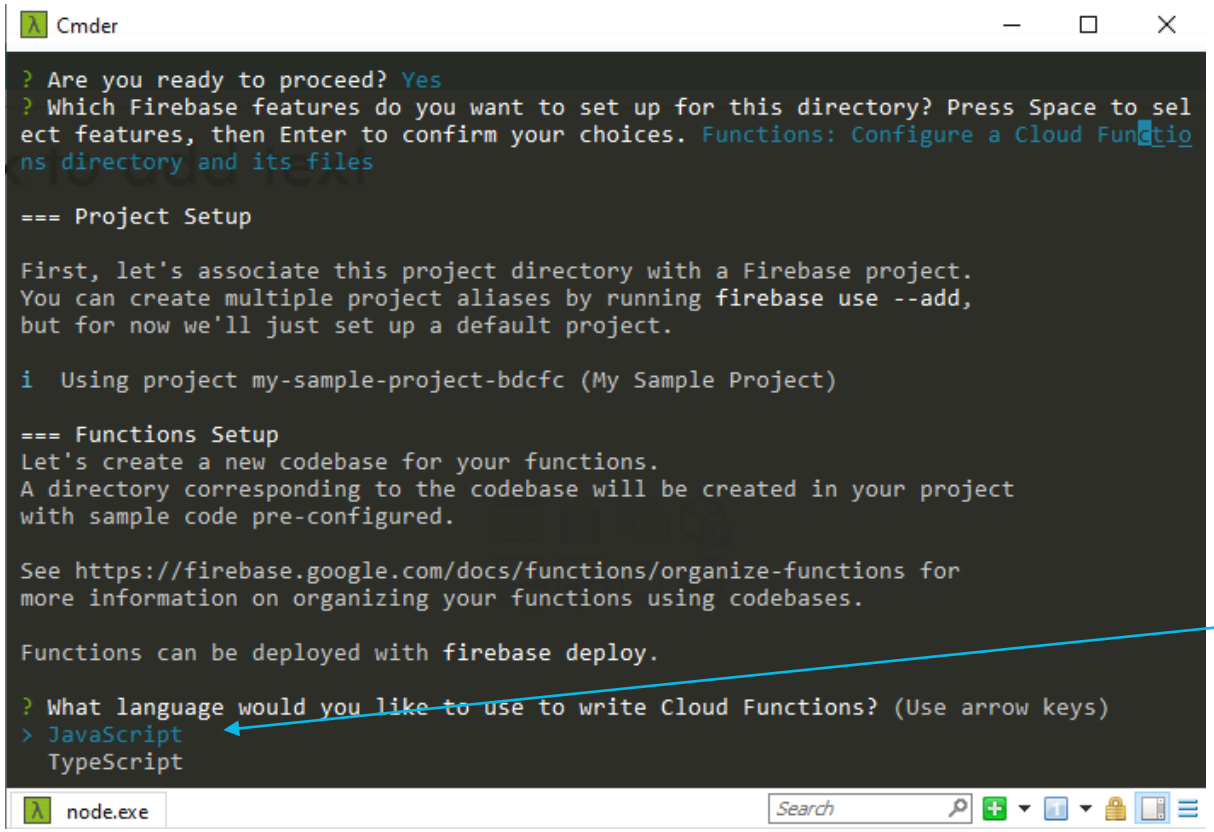
Adding Firebase Functions cont.



Select Yes

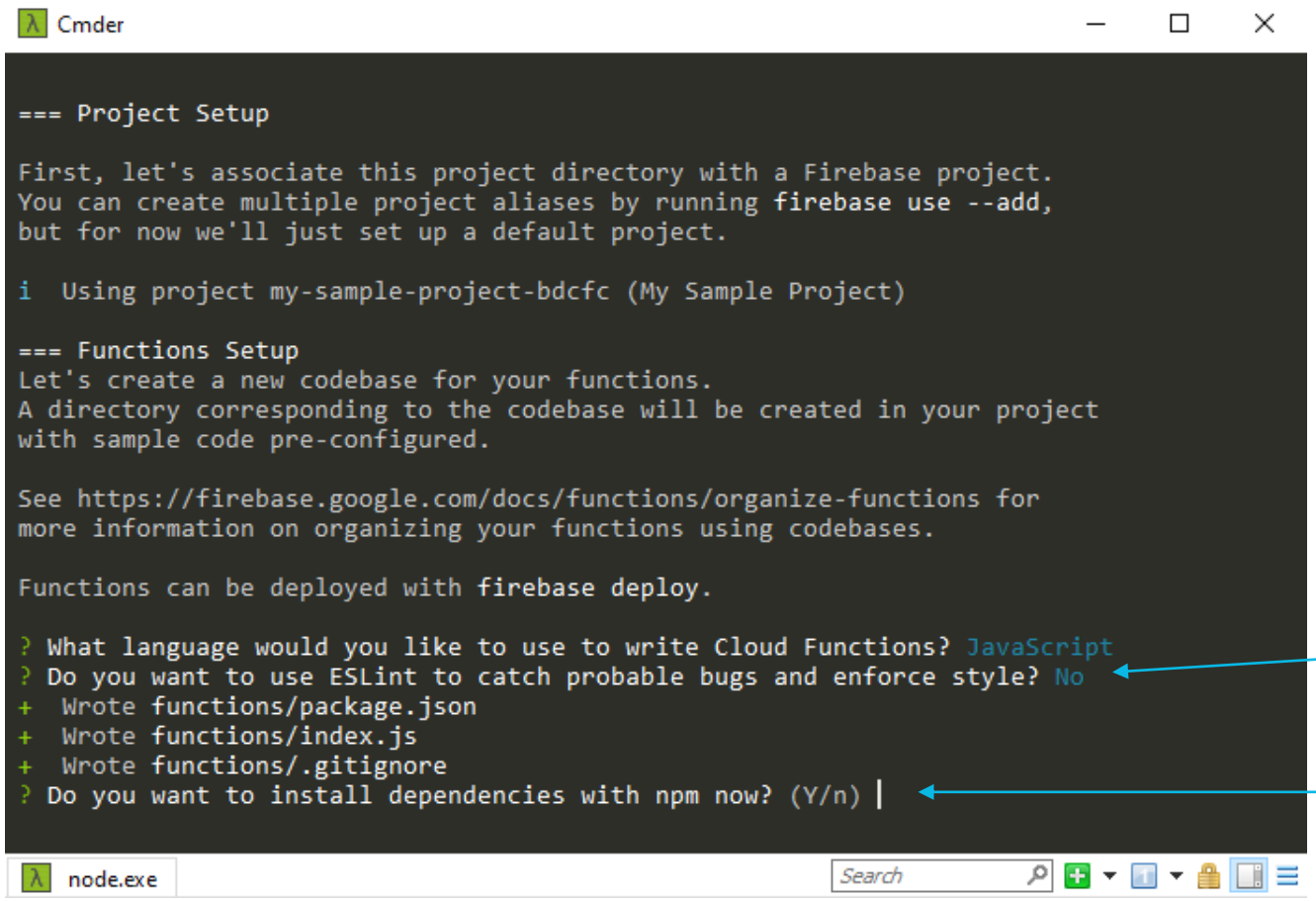
Select functions and hit enter

Adding Firebase Functions cont.



Select JavaScript

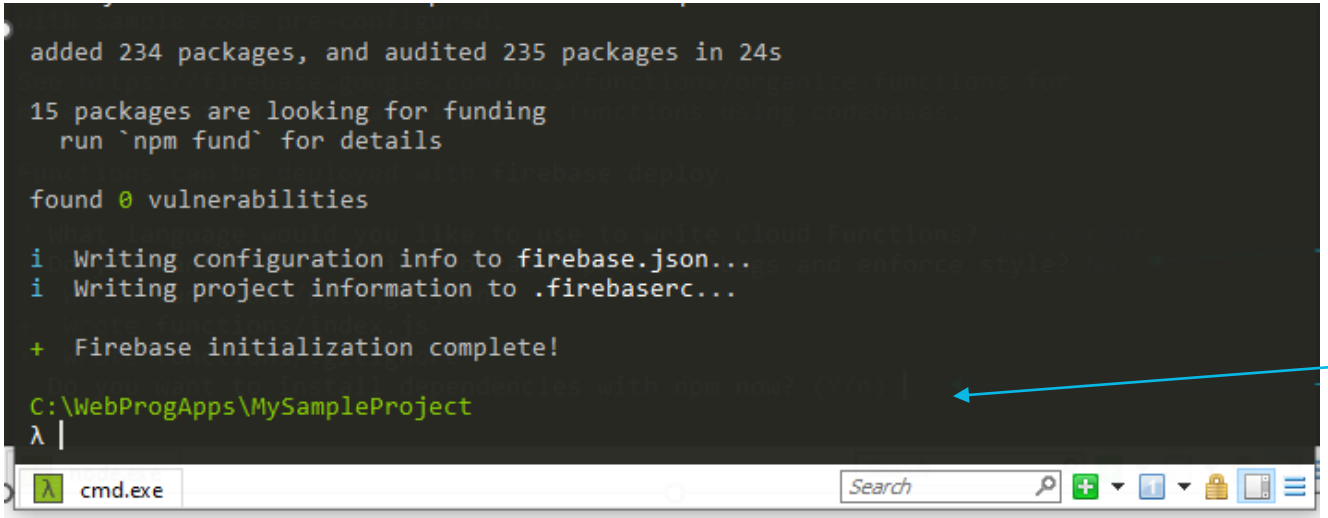
Adding Firebase Functions cont.



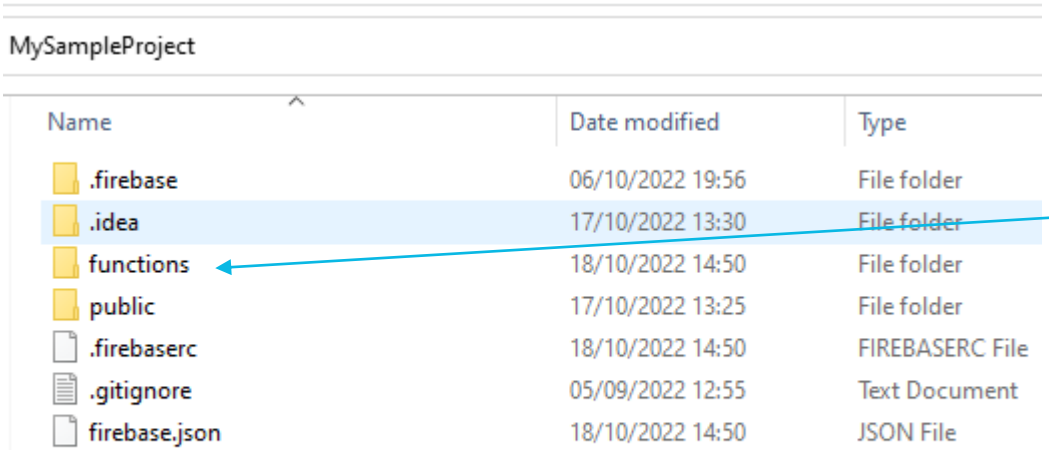
Select no to ESLint

Select Yes to installing the dependencies

Adding Firebase Functions cont.

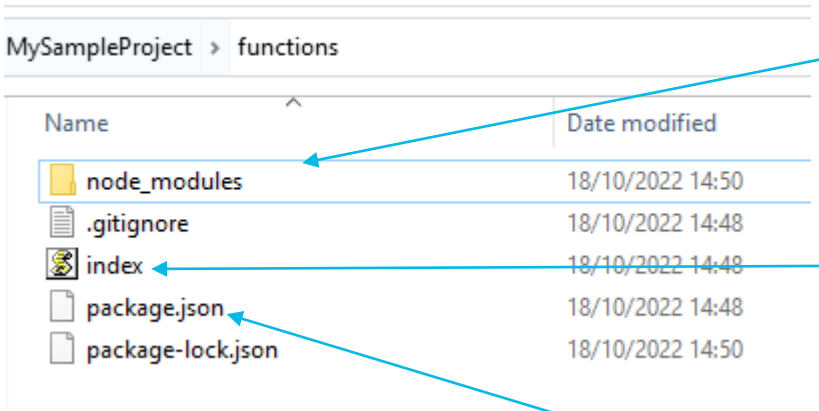


Finished installing the dependencies



Now be a new functions directory

Examining the functions folder



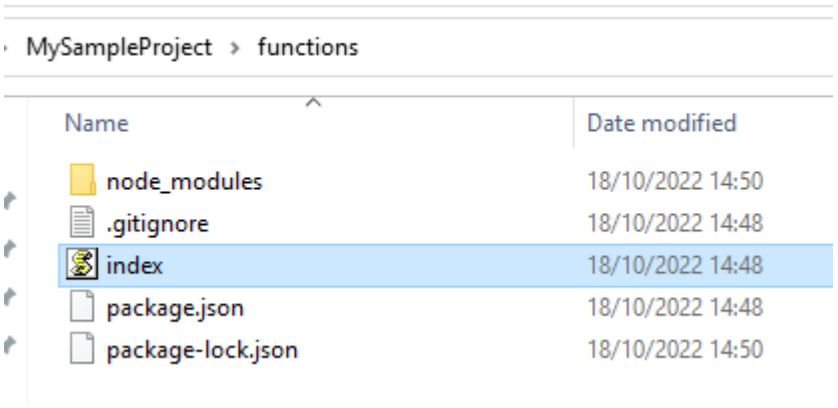
Contains dependencies

Where you will write your NodeJS code and functions

Allows you to define dependencies and other config settings

Examine the code


```
const functions = require('firebase-functions');

// // Create and Deploy Your First Cloud Functions
// // https://firebase.google.com/docs/functions/write-firebase-functions
//
// exports.helloWorld = functions.https.onRequest((request, response) => {
//   functions.logger.info("Hello logs!", {structuredData: true});
//   response.send("Hello from Firebase!");
// });
```



What is a library

- Libraries in programming are just blocks of code that you import so your program can use them?

```
require('firebase-functions');
```

- Who writes them?
  - Other Devs, yourself, could be anyone really…
- Why use them?
  - You can't use Firebase without this library.
    - It's like trying to control your TV without a remote
    - It's like trying to drive a car without a steering wheel, pedals etc.
- It's their platform so you must use their interface!

JS modules

- JavaScript allows the developer to package up code and functionality into modules

- Think of them as set of packaged functions that you wish to include in your application

JS modules

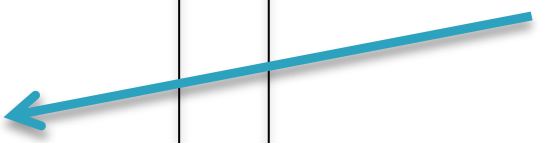
Both files in the same folder

hello.js

```
module.exports.hello = hello;

let …

function hello()
{
    return “hello”;
}
```

Driver.js

```
let mod = require('./hello');

let value = mod.hello();
```



Target

Source

Modules

- Examining our first REST API (cloud function)

```
const functions = require('firebase-functions');

// // Create and Deploy Your First Cloud Functions
// // https://firebase.google.com/docs/functions/write-firebase-functions
//
```

Exporting a module “helloWorld”

```
exports.helloWorld = functions.https.onRequest((request, response) => {
  functions.logger.info("Hello logs!", {structuredData: true});
  response.send("Hello from Firebase!");
});
```

Deploying our first cloud function

- The command is the same *firebase deploy*!
- You will see the function URL outputted to the console
- Paste it into the browser address bar to see if it works…Don't Google it! ☺

Summary

- Introduction to REST APIs
- Configuring Firebase functions on our apps
- Examining our first NodeJS code
- Modules in JavaScript
- Deploying our NodeJS functions to Firebase

FIREBASE FUNCTIONS, CALLBACKS, CREATING AND TESTING OUR FIRST FUNCTIONS

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

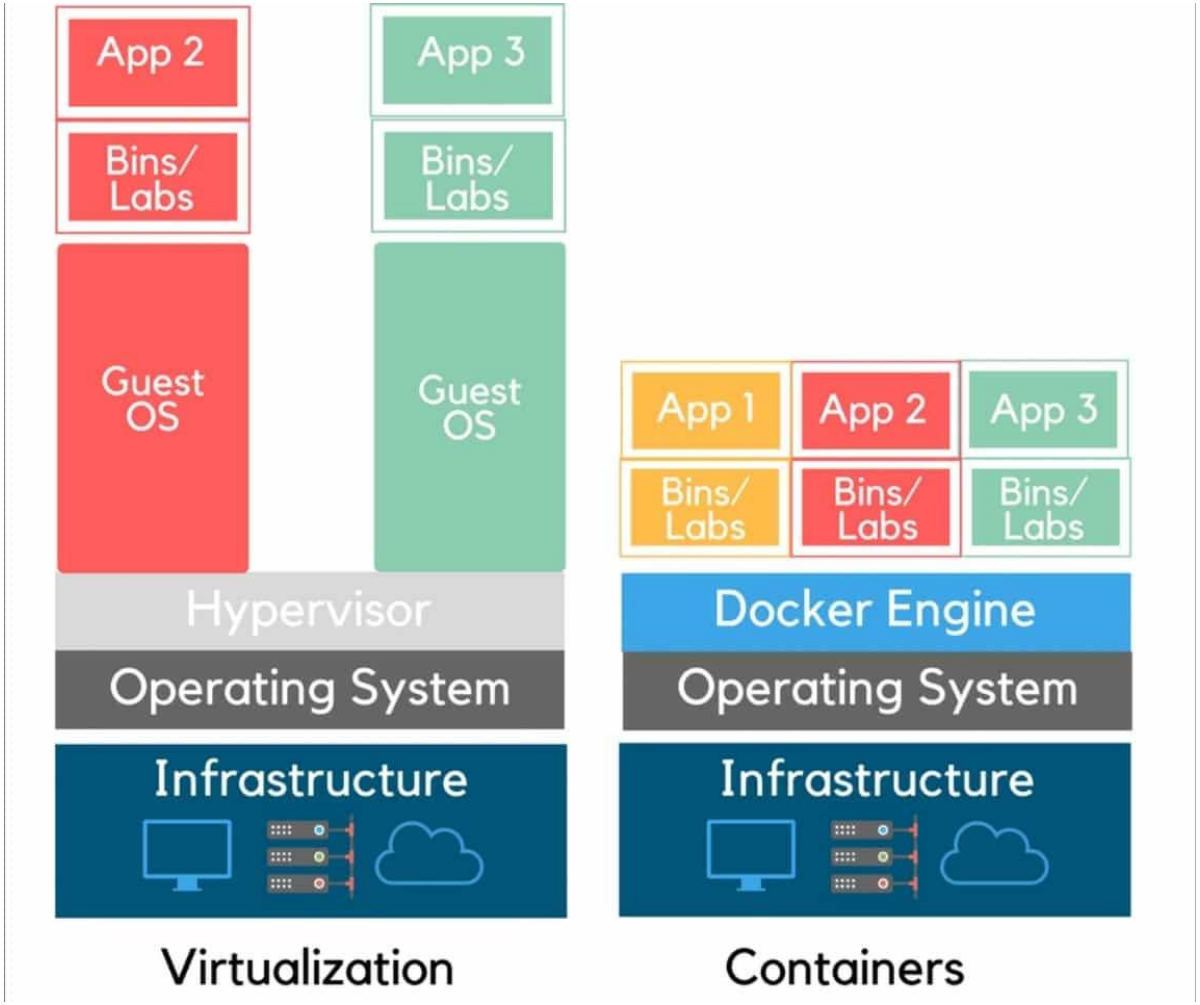
Summary

- Firebase functions
- Callback functions
- HTTP Verbs GET and POST
- Creating a dumb function which receives data and returns it back
- Testing with POSTMAN
- JSON
- Summary

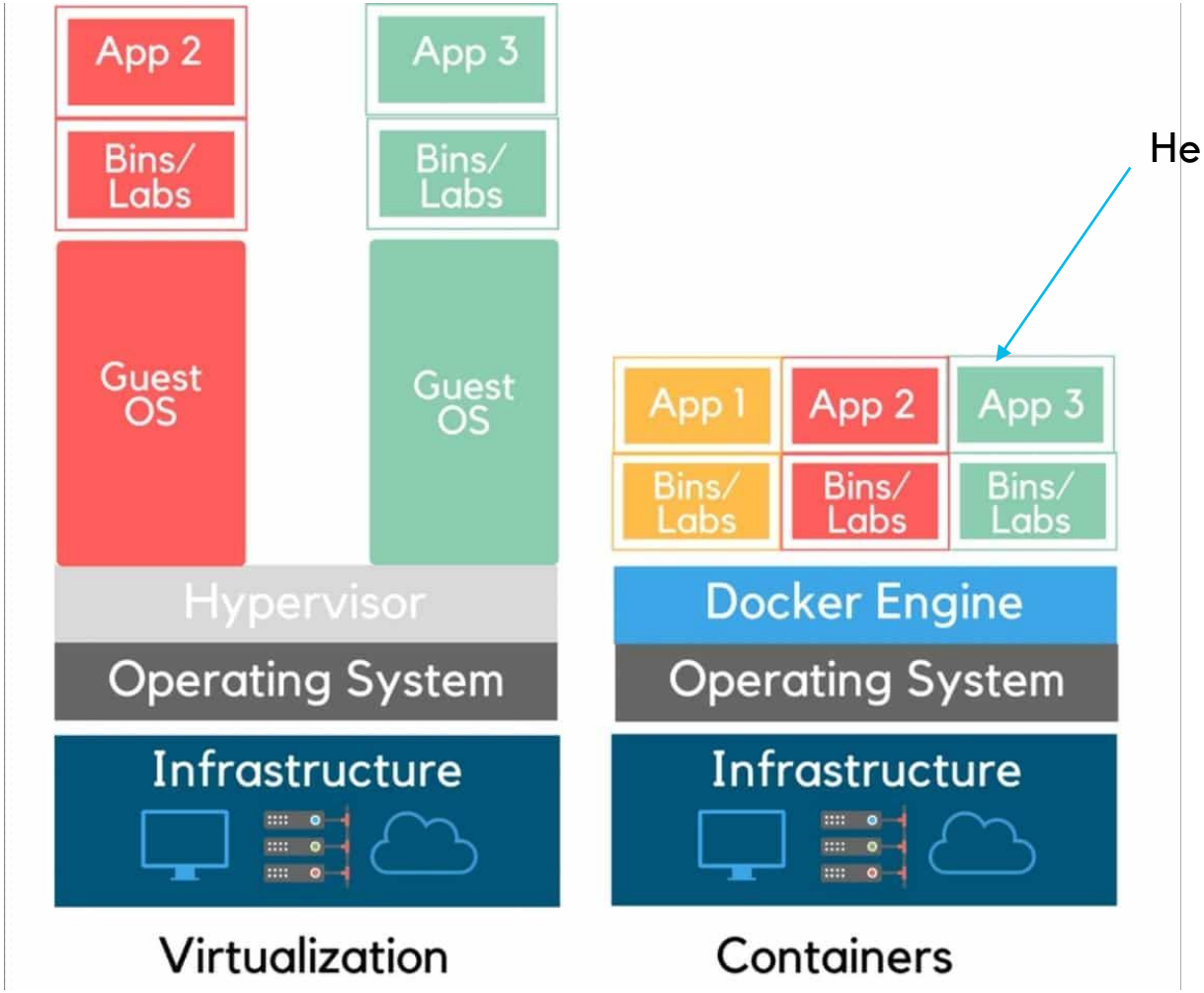
Firebase Functions

- It is a compute service that lets you run code without provisioning or managing servers.
- Similar to AWS Lambda, Azure Functions… It runs your code only when needed and scales automatically, from a few requests per day to thousands per second.
- You pay only for the compute time you consume - there is no charge when your code is not running.

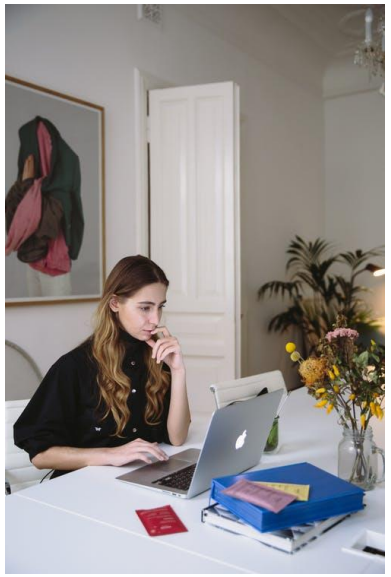
Containers vs VMs



Deploy a function what happens



Callback functions



Too busy to take a call, please leave your name and number (**function**) and I'll call you back when I'm **finished** my work (**invoke your function**).

Function callbacks

- Examining our first REST API (cloud function)

```
const functions = require('firebase-functions');

// // Create and Deploy Your First Cloud Functions
// // https://firebase.google.com/docs/functions/write-firebase-functions
// Callback function -- please run this code for me Firebase when a request is made

exports.helloWorld = functions.https.onRequest((request, response) => {
   functions.logger.info("Hello logs!", {structuredData: true});
   response.send("Hello from Firebase!");
});
```

myCoolApp/functions/index.js

Exercise: Deploying a function

- Deploy a function onto Firebase and return a string when it is invoked which says “Welcome to my cool new backend function”

- Add a second function below the first one and this time return a message saying “Not logged In!”

Passing Data - URL Query String

- We can encode parameters in the URL, we generally refer to this as the query string

- E.g. If you run a google search query, look at the URL after you search…

- http://www.mywebsite.com?data=hello

Simple function to mirror data

- Assume you want to create a function which parses out data submitted per request and mirrors it back to the requester

```
const functions = require('firebase-functions');

// Accept comment and return the same comment to the user

exports.echofunction = functions.https.onRequest((request, response) => {
    response.send(request.query.data);
});
```

myCoolApp/functions/index.js

Exercise: Parsing params from QS

- Write a function that assumes the data passed in the via the Query String is a number, take the value that is submitted per request, double it and then return it via the response.

- If the user submits a value that isn't a number then return a response that says "Please Enter a Number"

HTTP Verbs (GET and POST)

- GET and POST are HTTP request methods to transfer data from client to server.
- So far we have been making GET requests, GET is designed to request data from a specified resource
- POST is designed to submit data to the specified resource
- Both can be used to send requests and receive responses

Request Structure

- All HTTP requests have a three main parts
  - Request line
    - HTTP Method (GET, POST, etc.)
    - URL – address of the resource that is being requested
    - HTTP version
  - Headers
    - Additional information passed between the browser and the server, i.e. cookies, browser version, OS version, auth tokens, content-type
  - Message body
    - Client and server use the message body to transmit data back and forth between each other. POST request method will usually have data in the body. GET requests leave the message data empty

GET Method

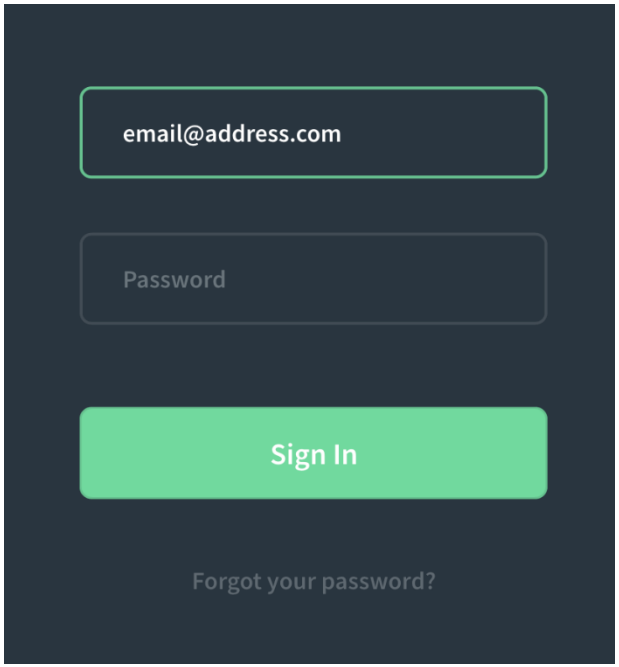
- GET requests can be cached
- GET requests remain in the browser history (you can go back!)
- GET can't be used to send binary data, like images or word documents to the server
- GET requests can be bookmarked
- GET requests have length restrictions
- GET requests should only be used to retrieve data
- Using GET data can be sent to the server by adding name=value pairs at end of the URL, i.e. Querystring
  - `mysite.app.web…/page?id=101&name=John`

POST Method

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length
- The POST method can be used to send ASCII as well as binary data

POST Request Form Data

- Form data is often sent to the server via a POST request



Alternative option is to send username and password to the server via the QueryString – uid=email@address.com
pwd=password

But is this a good idea?

GET vs POST

- Use GET if you are requesting a resource
  - You may need to send some data to get the correct response back, but in general the idea is to GET a resource

- Use POST if you want to send data to the server
- Other methods
  - PUT //Update/Replace
  - DELETE //Delete
  - PATCH //Partial update/modify

POST data to server

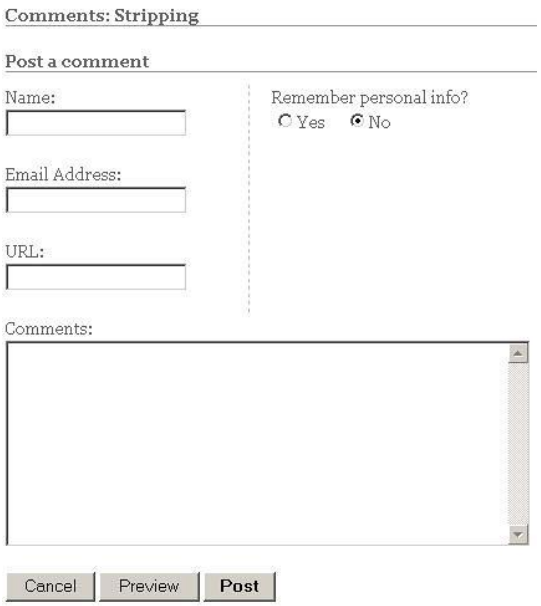
- Assume you are building a form which posts comments from your blog page to the server
- First step is to write a function to accept the comment

```
const functions = require('firebase-functions');

// Accept comment and return the same comment to the user

exports.postcomment = functions.https.onRequest((request, response) => {
    response.send(request.body);
});
```

myCoolApp/functions/index.js

How to test a POST request?

- We can create a form on a web page, then write JavaScript to send the data via POST to the server.
- However it would be nice if there was a way to test it first without having to go back to the frontend



Postman client

- When writing backend APIs such as the one we have just completed, it's often necessary to test it quickly.
- You don't want to have to write a client side request to test each API. Sometimes you may even want to pass in values which would take even longer to code up.
- Postman can help!
- https://www.postman.com/downloads/

POSTMAN

- It's brilliant for letting us test our APIs without having to write client side code to make the requests.
- It will work for all request methods, i.e. GET, POST, PUT etc.

- You can code the backend independent of the frontend!
- How else could we test to see if postcomments is working!

Sending data, what format?

- Now that we have an API available to receive data, and we have a client (postman) willing to send the data, we need to decide on a data format…

- Enter JavaScript Object Notation or JSON

JSON

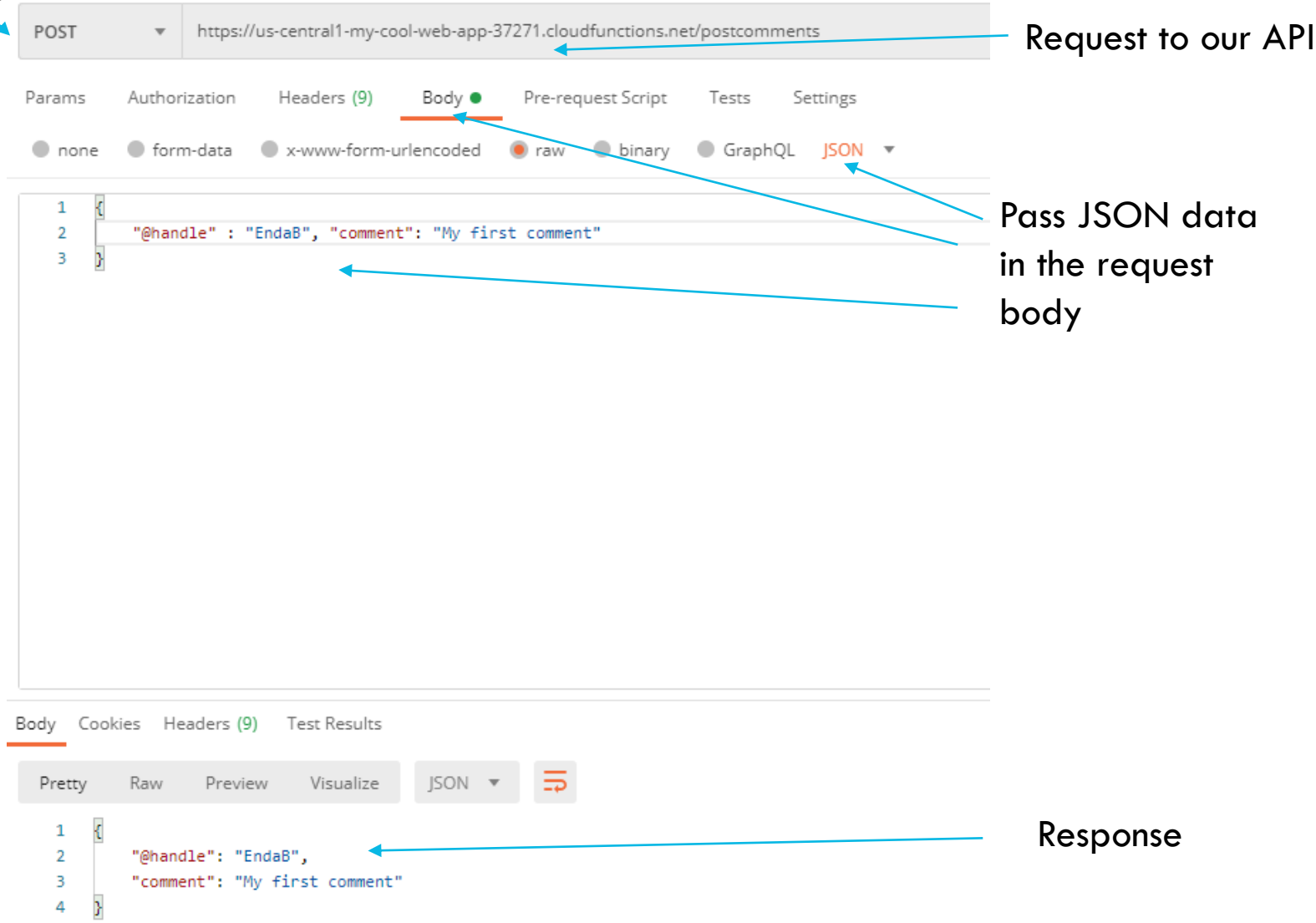
- JavaScript Object Notation (JSON)
- It is an open, human and machine-readable standard that facilitates data interchange
- Along with XML it is the main data interchange format on the web
- Data types
  - Numbers, Strings, Booleans, Arrays, Objects
  - ISODate() returns a date object
- Firestore uses JSON documents to store records of information

JSON cont.

- Arrays
  - [“a”, “b”, “c”, “d”, “e”, “f”]
  - [“apple”, 3, null, true]
- Objects
  - {“Enda” : 45, “John” : 33, “Sam” : “Smith”}
- Array of Objects
  - [{}]
- Use double quotes, no comma last value

POST JSON to Server

Set POST



Exercise 3

- Create a function which accepts comment information (JSON formatted) in the body of the request i.e. {“Comment”: “This is my comment”}
- Make a request via Postman to send the data via a POST request
- Respond with a message saying “I received your comment, thank you”.

Summary

- Firebase functions
- Callback functions
- HTTP Verbs GET and POST
- Creating a dumb function which receives data and returns it back
- Testing with POSTMAN
- JSON
- Summary

PART 1: INTRODUCTION TO FIRESTORE AND CREATING OUR FIRST DATABASE

Lecture Overview

- Firestore Database
  - Overview of Document Driven Databases
  - Creating our first database
- Connecting the database to our Firebase functions
  - Writing our comment data to the database
  - Reading our comment data from the database
- All will be tested using POSTMAN

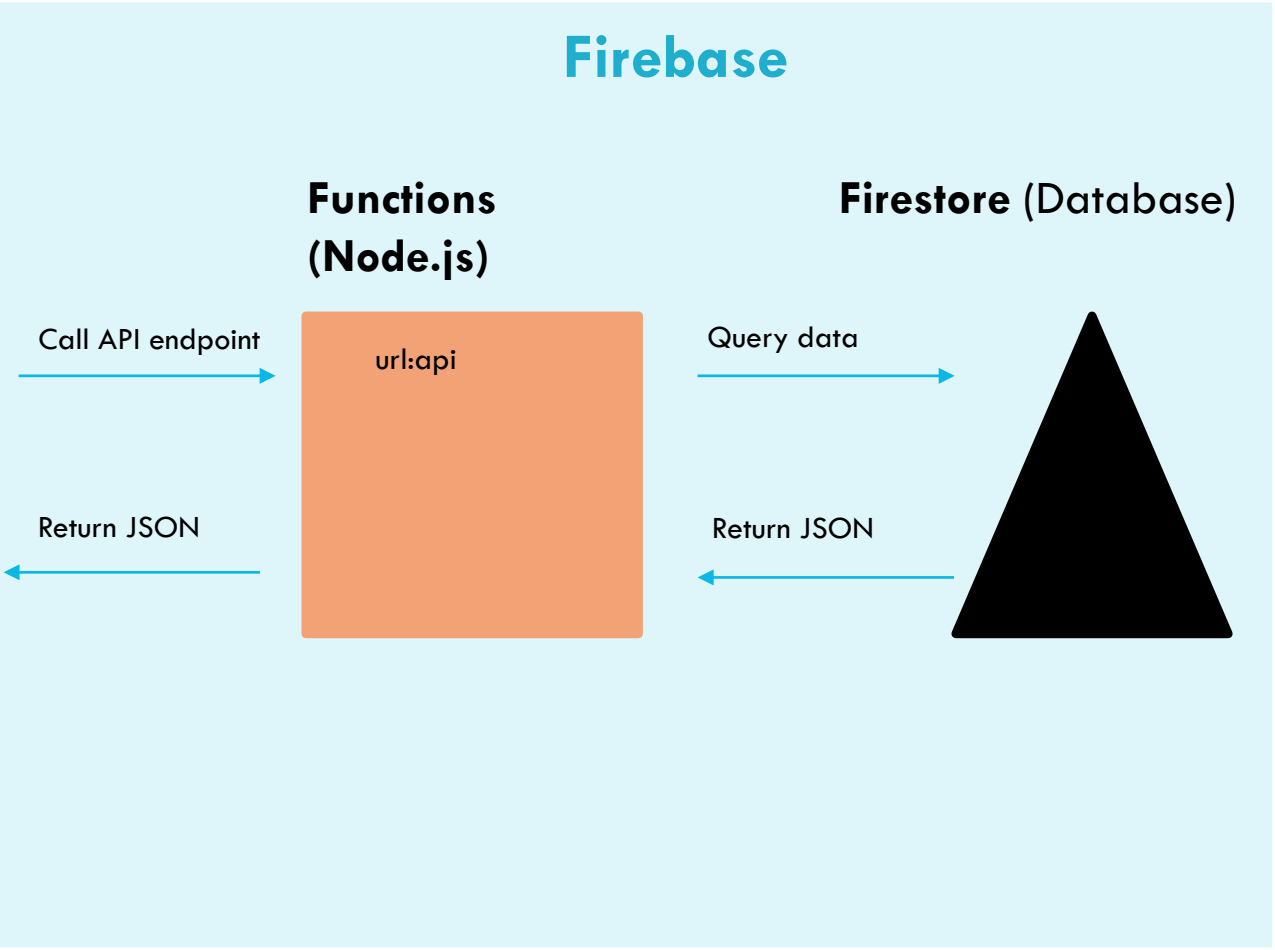
Purpose of the lecture

- The goal is to introduce you to Firestore, from the point of view of using it as a backend for your applications. The majority of the discussion will be practically focussed, with little theory concerning more advanced database concepts such as sharding, normalisation, concurrency, BSON, locking writes/reads etc.

- It will be a basic introduction on how to get a database connected to your applications.

Architecture

Clients



What is Firestore?

- Firestore is a Document Driven Database.
- Documents follow a **property:value** format
  - JSON
- Scalable, highly performant and document oriented.
- The databases tend to scale more easily horizontally.

Database concepts

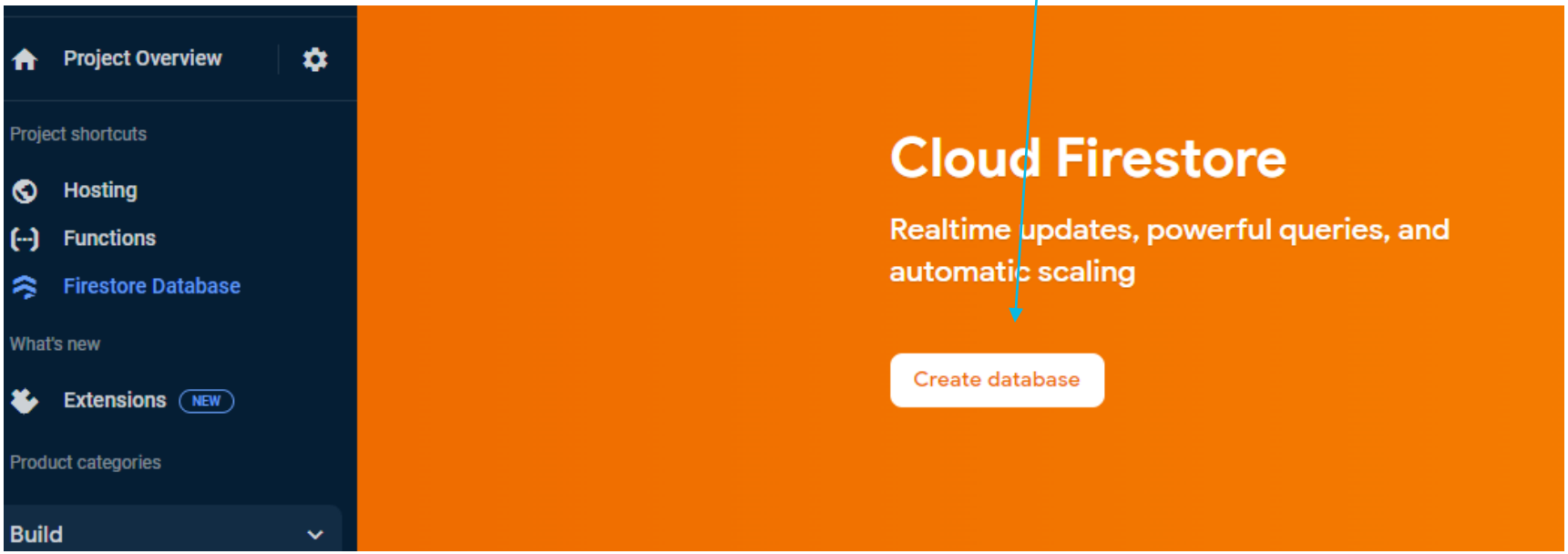
- Records in Firestore are known as "**Documents**"
  - These *documents* are just JSON data
- Documents are grouped into **"Collections"** which are equivalent to tables in relational databases
- Queries are still queries, however there is NoSQL!

SQL to Firestore Terminology

Database	⟷	Database
Table	⟷	Collection
Record/Tuple/Row	⟷	Document
Column	⟷	Field

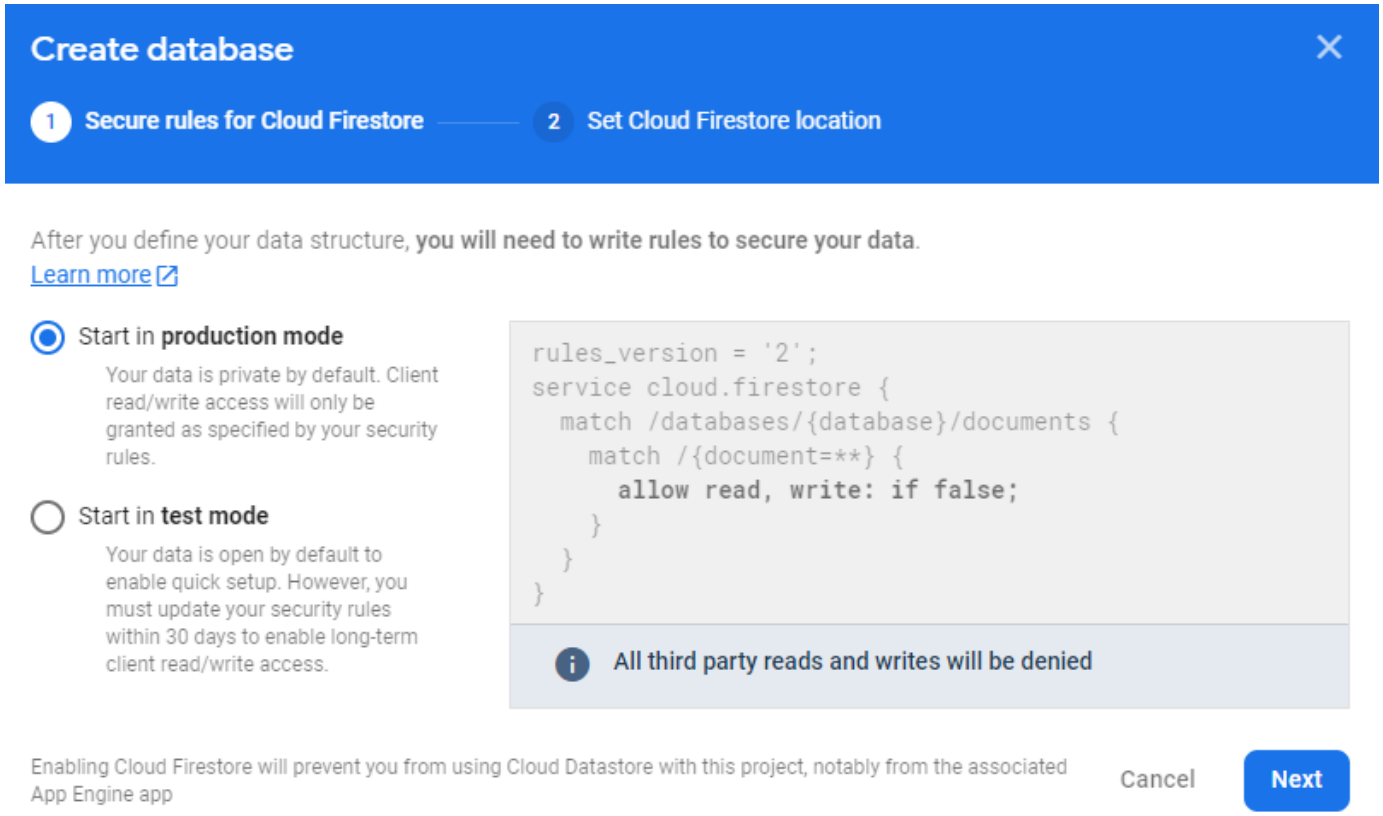
Creating our first database

Login to the Firebase dashboard, click on Firestore and then “Create database”



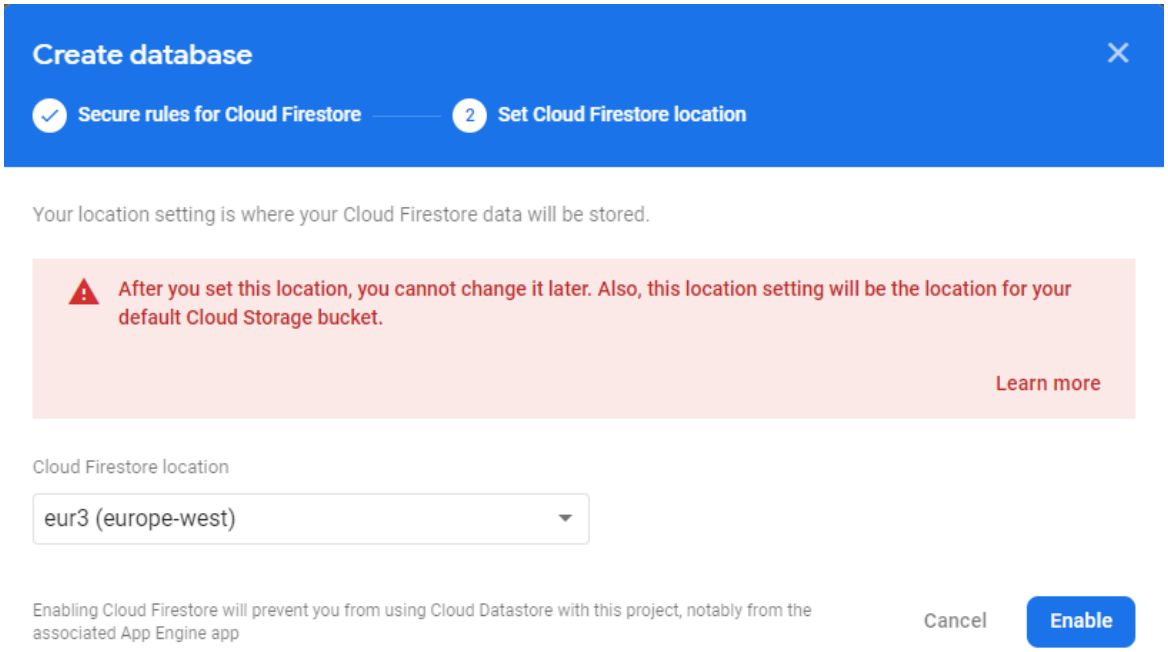
Open in production mode

- Start in production mode



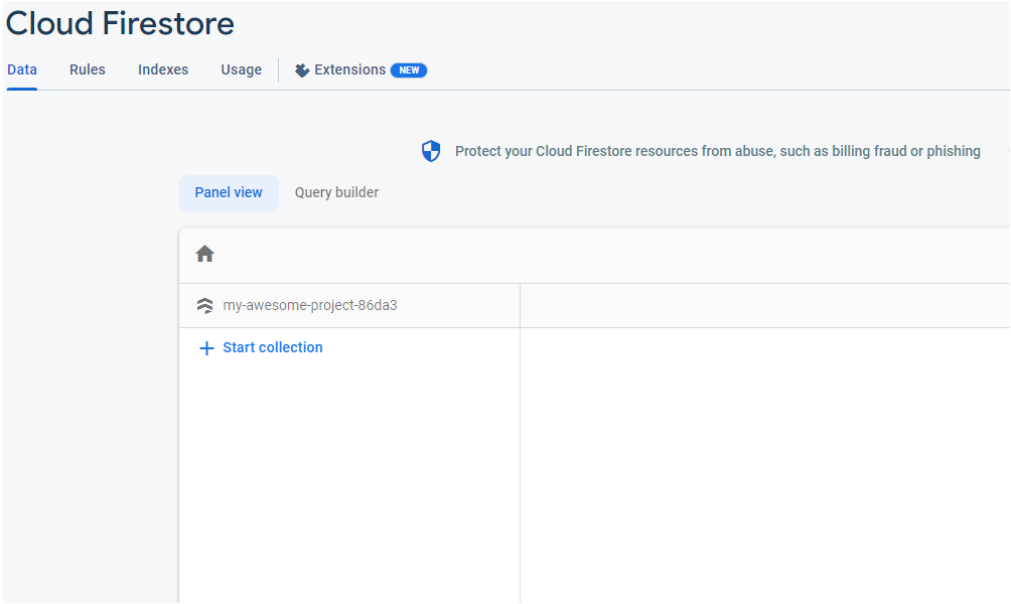
Choosing a region

- The latency should be fairly low so the default region will be fine, but if you want to place it in Europe please select it in the dropdown and then click enable



Database is now created

- You can create a collection and add documents manually via this web interface. But the next step is to connect to it with our functions and read/write data.



Summary Overview

- ~~Firestore Database~~
  - ~~Overview of Document Driven Databases~~
  - ~~Creating our first database~~

- Connecting the database to our Firebase functions
  - Writing our comment data to the database
  - Reading our comment data from the database

Writing data to the database

- To motivate data writing we will reuse the postcomments function
- This is known as “Creating” a document
- I’ll create a new document every time the postcomments function is called and save it in the database
- https://firebase.google.com/docs/firestore

Firebase admin

- Firebase provides an admin library to allow your server code (functions) to run in an authenticated mode

- This means your code can connect to the database, create docs, delete docs, update etc. all securely

```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp();
```

Promise – More async hell

- In ES6 a new concept was added to JavaScript to handle **Callback hell**
- These are called promises
- What's the difference between callbacks and promises?
  - Callback is passed as an argument
  - Promise is something that is achieved or completed in the future.
    - Promise is an object, **then()** method (if promise is fulfilled) and **catch** (if promise is rejected)

Code examples


```
asyncFunc(result => {
    console.log(result);
});
```

Callback

```
const promise = asyncFunc(()=>{
    return new Promise...
});
promise.then(result => {
    console.log(result);
});
```

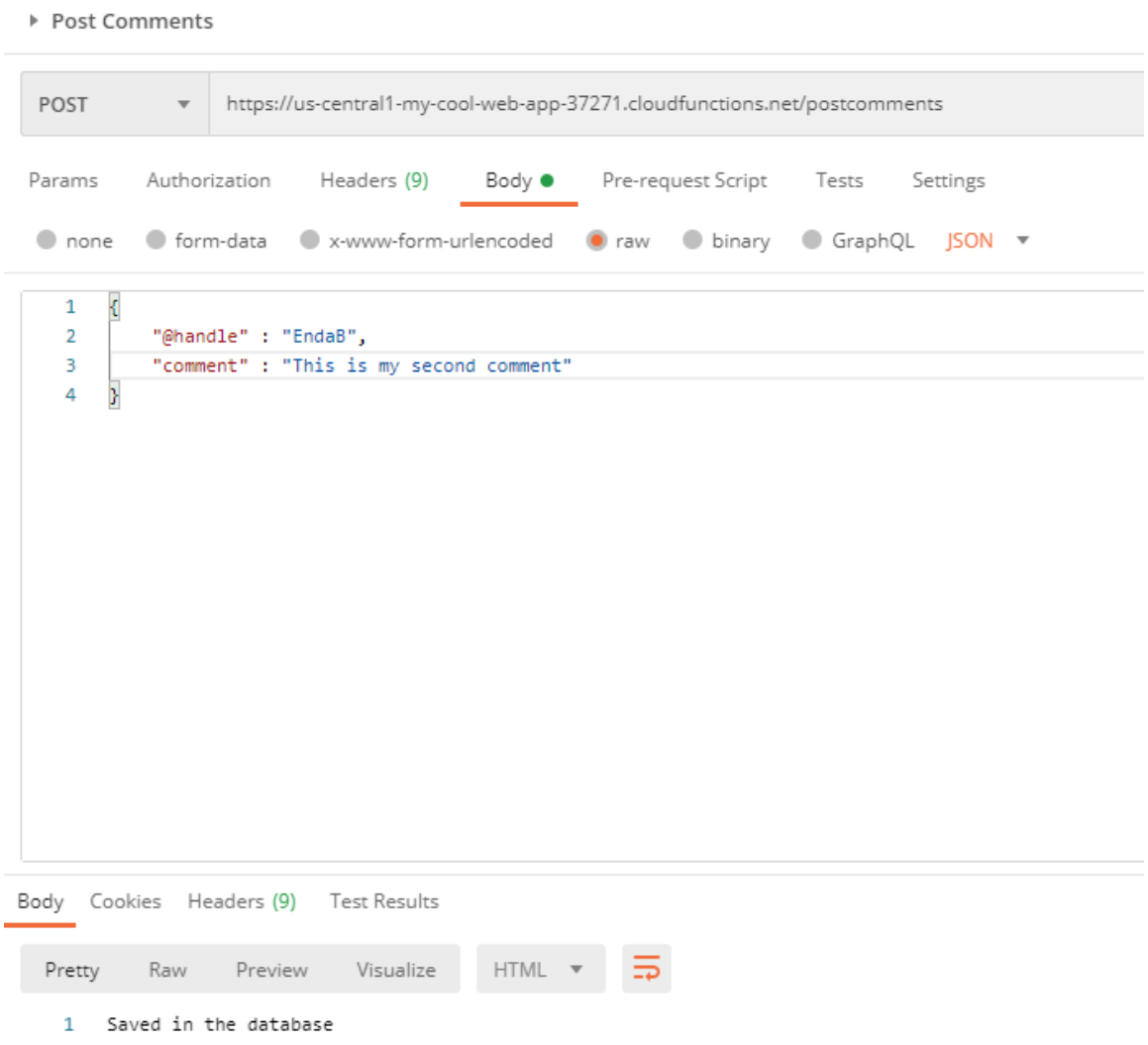
Promise

Adding a document


```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp();

exports.postcomments = functions.https.onRequest((request, response) => {
   // 1. Receive comment data in here from user POST request
   // 2. Connect to our Firestore database
   return admin.firestore().collection('comments').add(request.body).then(()=>{
                        response.send("Saved in the database");
   });
});
```

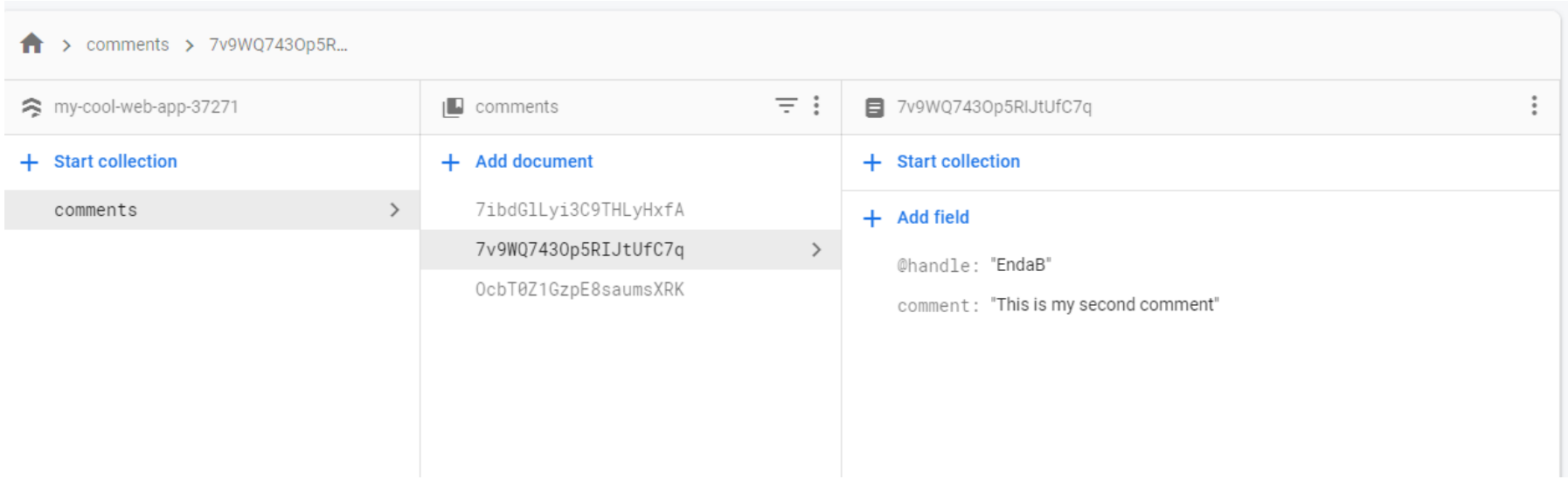
myCoolApp/Functions/index.js

Using POSTMAN POST to the fn



Check the database to see if it saved

- If you check on Firebase you should now see your comment

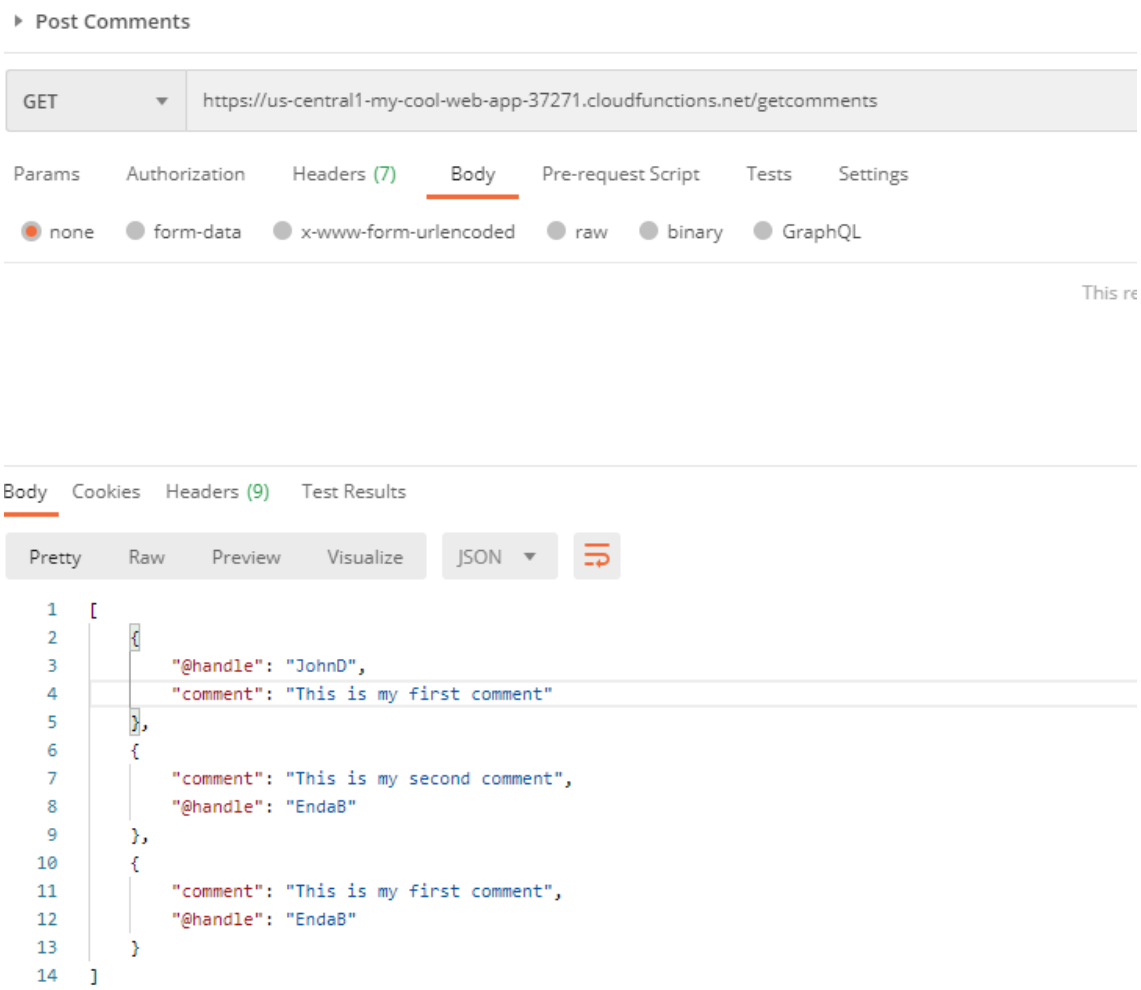


Reading our documents


```
exports.getcomments = functions.https.onRequest((request, response) =>
{

    // 1. Connect to our Firestore database
    let myData = []
    admin.firestore().collection('comments').get().then((snapshot) => {

                if (snapshot.empty) {
                        console.log('No matching documents.');
                        response.send('No data in database');
                        return;
                }

                snapshot.forEach(doc => {
                        myData.push(doc.data());
        });

        // 2. Send data back to client
        response.send(myData);
    })
});
```

myCoolApp/Functions/index.js

Test the function with POSTMAN



```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp();

exports.postcomments = functions.https.onRequest((request, response) => {

   // 1. Receive comment data in here from user POST request
   // 2. Connect to our Firestore database
   admin.firestore().collection('comments').add(request.body);
   response.send("Saved in the database");
});

exports.getcomments = functions.https.onRequest((request, response) => {

   // 1. Connect to our Firestore database
   let myData = []
   admin.firestore().collection('comments').get().then((snapshot) => {

                    if (snapshot.empty) {
                              console.log('No matching documents.');
                              response.send('No data in database');
                              return;
                    }

                    snapshot.forEach(doc => {
                              myData.push(doc.data());
          });

          // 2. Send data back to client
          response.send(myData);
   })
});
```

myCoolApp/Functions/index.js

OrderBy

- So far when reading comments from the database we have not given any consideration to their order
- Perhaps it would be useful to order them by postdate or perhaps by the number of likes etc.
- To do this we need to modify our Firebase functions postcomments and getcomments to order the comments

Creating comments - postcomments

- The Firestore database supports a timestamp field, which we can use to store the date and time each comment was posted.

- Once this is recorded on each document we can return the comments to the user in order of their post date/time.

Posting comments

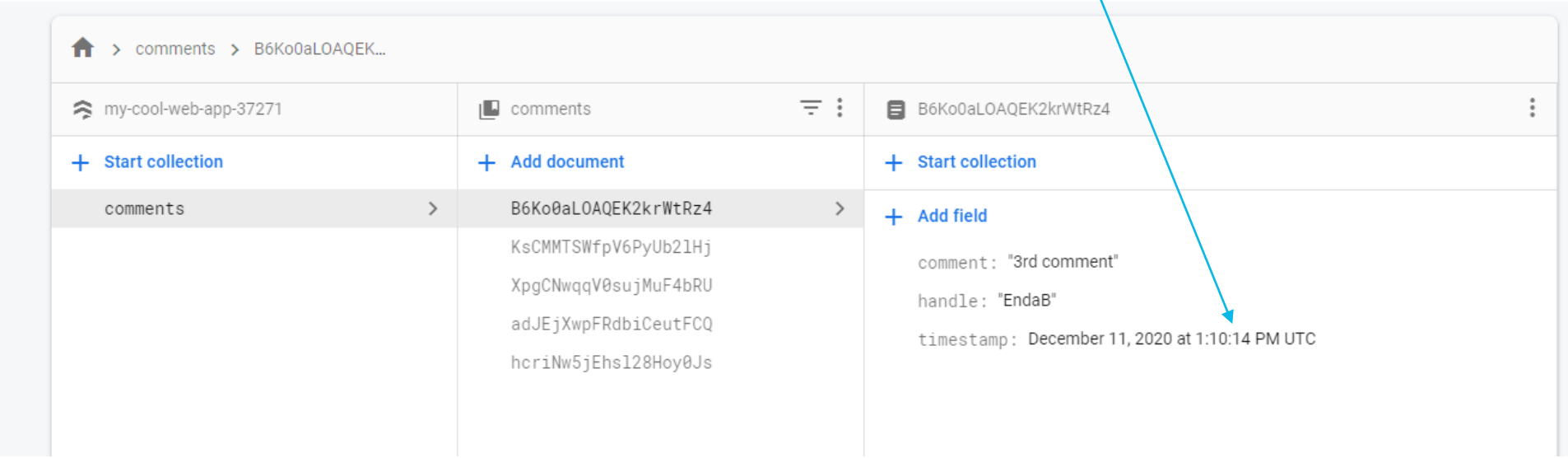

```
exports.postcomment = functions.https.onRequest((request, response) => {
        console.log("Request body", request.body);
                // Create a timestamp to add to the comment document
        const currentTime = admin.firestore.Timestamp.now();
        request.body.timestamp = currentTime;

        admin.firestore().collection('comments').add(request.body).then(()=>{
                response.send("Saved in the database");
        });
});
```

Don't forget to hit *firebase deploy* once you have made your changes

myCoolApp/functions/index.js

Check database

- When you post a comment you should now see a timestamp beside each comment



Ordering documents by timestamp

- We now modify the get comments firebase function to order the comments by timestamp

```
exports.getcomments = functions.https.onRequest((request, response) => {

    // 1. Connect to our Firestore database
    let myData = []
    admin.firestore().collection('comments').orderBy('timestamp').get().then((snapshot) => {

                        if (snapshot.empty) {
                                    console.log('No matching documents.');
                                    response.send('No data in database');
                                    return;
                        }

                        snapshot.forEach(doc => {
                                    myData.push(doc.data());
            });

            // 2. Send data back to client
            response.send(myData);
    })
});
```

myCoolApp/functions/index.js

Lecture Overview

- Firestore Database
  - Overview of Document Driven Databases
  - Creating our first database
- Connecting the database to our Firebase functions
  - Writing our comment data to the database
  - Reading our comment data from the database

SOURCE CONTROL – CT 216 SOFTWARE ENGINEERING I

Dr. Enda Barrett

Ollscoil na Gaillimhe
University of Galway

Source control - overview

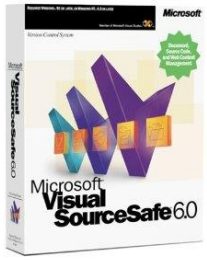
- Version (Source) control
  - Version control software
  - Why we need it?
- Git
  - What is Git
  - Branching
  - Pull requests

Version control

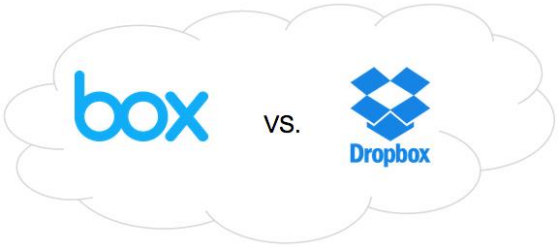
- What is version control?
  - **Version control** is a system that records changes to a file or set of files over time so that you can recall specific **versions** later
  - Also known as **revision control** or **source control**
- Keeps track of changes, by whom and when
- Fundamental tool for developing software projects

Version control software?

- Subversion, GIT, VSS, CVS, Mercurial



- Revision control is actually present in a variety of software products

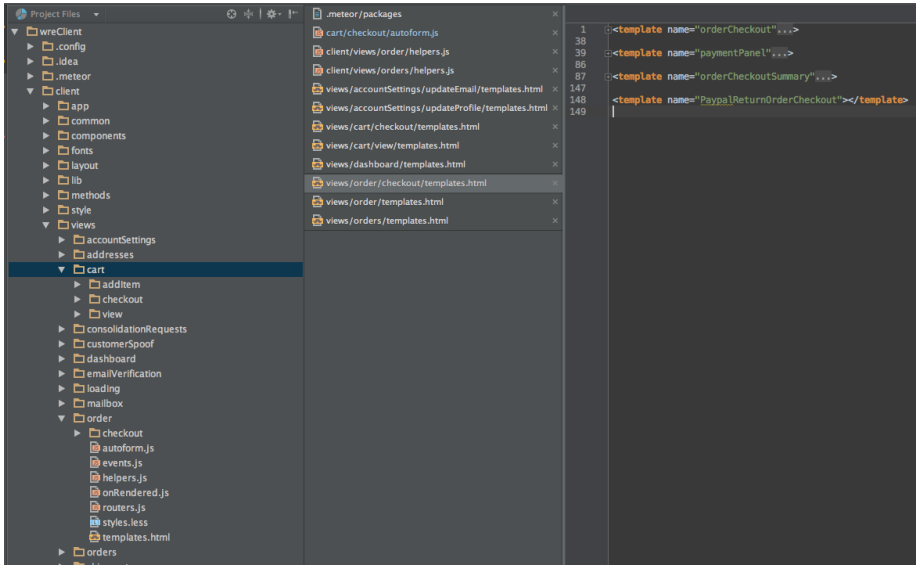


Why do we need it?

- Backup software source
  - Roll back to previous versions
- Keeping a record of who did what and when
  - Know who to praise and who to fire!
- Collaborating with other people (teams)
- Troubleshooting
  - Analyse the change history to figure out what caused the problem
- Statistics
  - Find out who is the most productive!

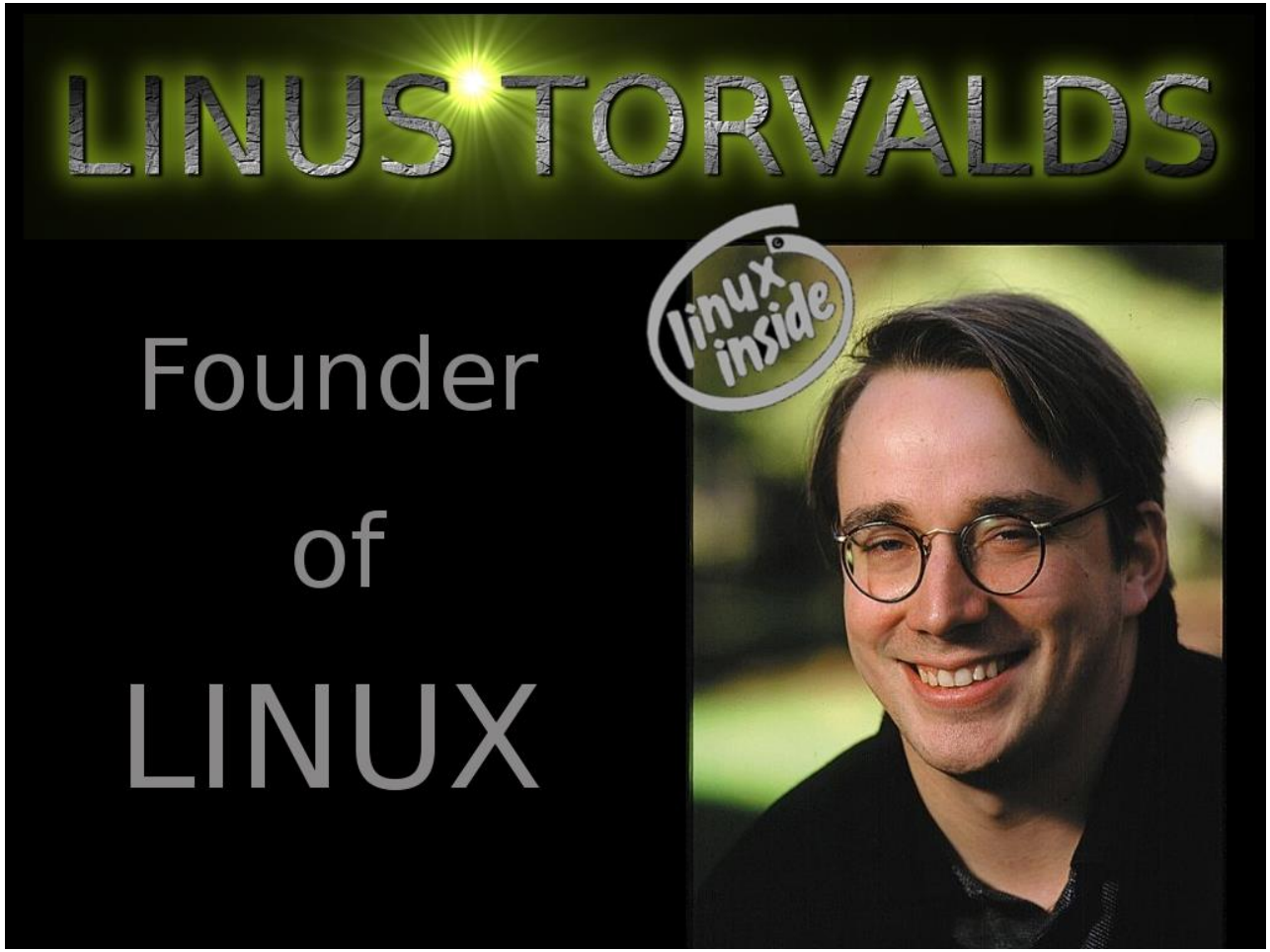
What should you commit?

- Web project (HTML, CSS, JavaScript, Images, Documentation, Functions, Configuration files)



- Not the Node Modules folder

GIT Creator



Why was it created?

- For a long time Torvalds wasn't using any version control for the Linux Kernel (1991-2002).
- Changes were passed around as patches and archived files.
- In 2002 they began using BitKeeper for managing the source for the Linux Kernel
- In 2005 the relationship broke down and BitKeeper revoked their licence.
- Initial release 7th April 2005

https://git-scm.com/book/az/v1/Ba%C5%9Flan%C4%9F%C4%B1c-A-Short-History-of-Git

They needed something similar to BitKeeper

- The developers had the following objectives in mind:
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like the Linux kernel efficiently (speed and data size)

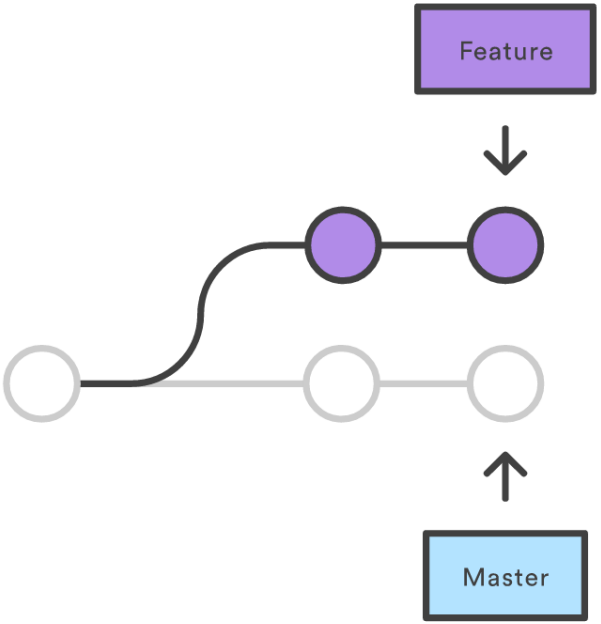
Why is everyone moving to GIT?

- Not all but quite a few!
- One of the most touted reasons is that of DVCS
  - Works great when you have no access to the internet! No VPN access to the SVN server…
  - No single point of failure
- Even offline you can access the history, branches, versions etc…

Feature branch workflow

- It encourages branching for every feature

- No matter how big or small the feature, a branch can easily be created and is encouraged.



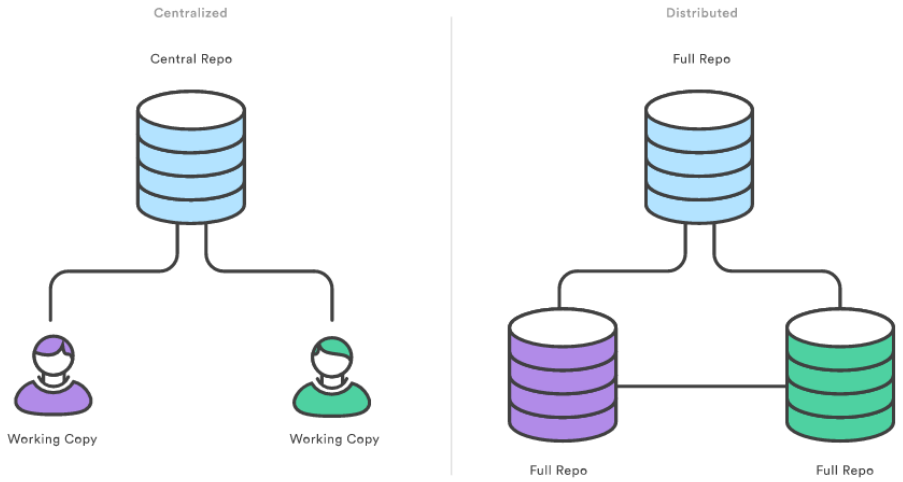
https://www.atlassian.com/git/

Distributed development

Each developer gets their repo complete with history of commits

This makes GIT extremely fast
You don't need a network connection to

- Commit changes
- Inspect previous versions
- Perform diffs between commits

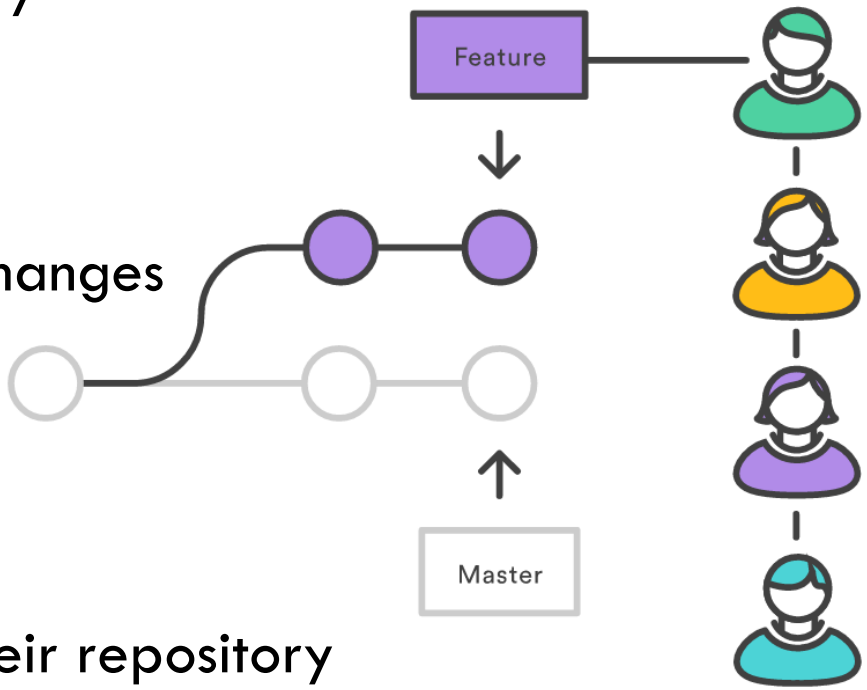


If someone breaks the production branch/trunk in SVN, it blocks everyone else from committing, with GIT you can continue

https://www.atlassian.com/git/

Pull requests

- A pull request is where you ask another developer to merge your feature into their repository
- Proj. leads can keep track of changes
- Proj. leads can merge it with their repository



https://www.atlassian.com/git/

Source control - overview

- Version (Source) control
  - Version control software
  - Why we need it?
- Git
  - What is Git
  - Branching
  - Pull requests

USING GIT AND GITHUB IN OUR APPS – CT 216 SOFTWARE ENGINEERING I

Dr. Enda Barrett

Ollscoil na Gaillimhe
University of Galway

Source control - overview

- Git
  - Installing Git
- Adding Git to your projects
  - Committing code
- GitHub
  - Pushing source to remote GitHub repo
  - Cloning a repository
  - Pull requests

Getting GIT – Windows && Mac

- Install it on your machine
- Downloads available at for Windows and Mac
  - https://git-scm.com/download/windows
  - https://git-scm.com/download/mac
- Go with the default install options
- Set details so that every commit is logged correctly…
  - git config --global user.name "Enda Barrett"
  - git config --global user.email "Enda.Barrett@nuigalway.ie"

GIT – Adding Git to your projects

- Once installed navigate to your app directory (myCoolNewApp) on the command line
- Type *git init* from the root of that directory on the command line
  - Creates a repository in this directory
- Execute command *git add* -A
  - Adds all files in the directory to the local repository
- Execute command *git commit*
  - Commits everything to version control

Commit

- When committing you will be asked to put a comment at the top to indicate what changes you made. It's important to be descriptive here so your colleagues can understand the changes made

- Type *i* to begin inserting text

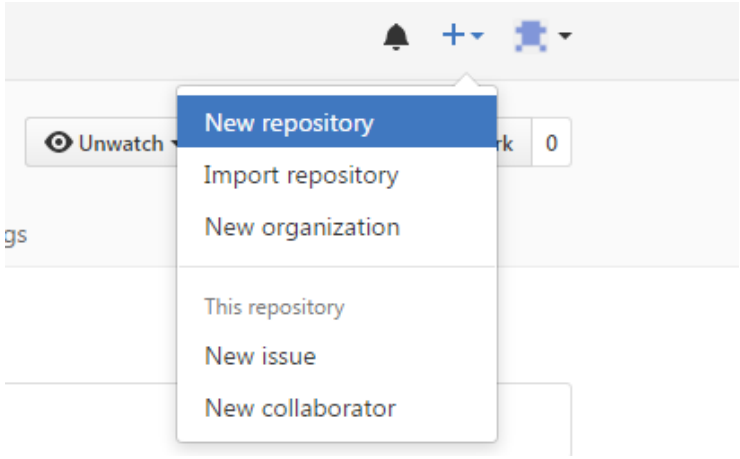
- When finished typing press *Esc* and then *:wq* and hit *Enter*

GitHub

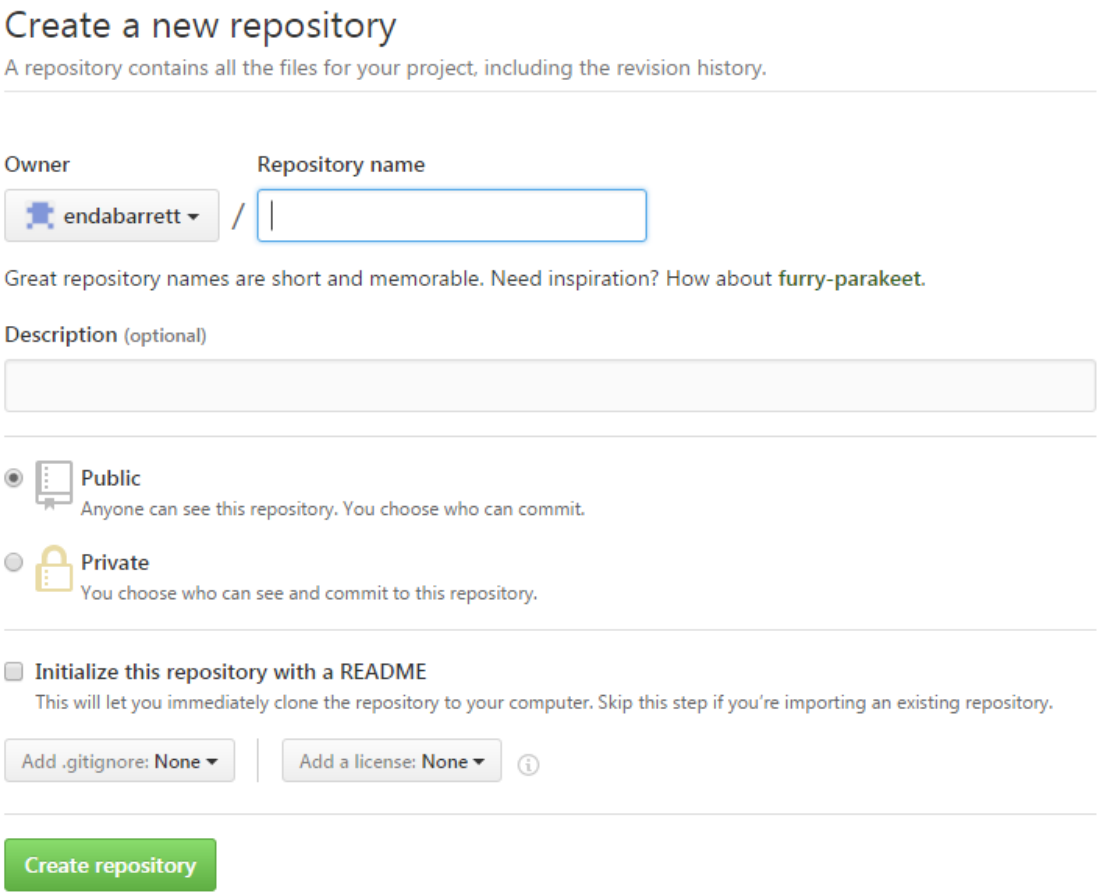
- What is GitHub?
  - GitHub is a web based hosted service for Git repositories. Git allows you to host remote Git repositories and has a wealth of community based services that makes it ideal for open source projects.
- It's really three things
  - A publishing tool
  - Version control system
  - Collaboration platform

Pushing your repository to Github

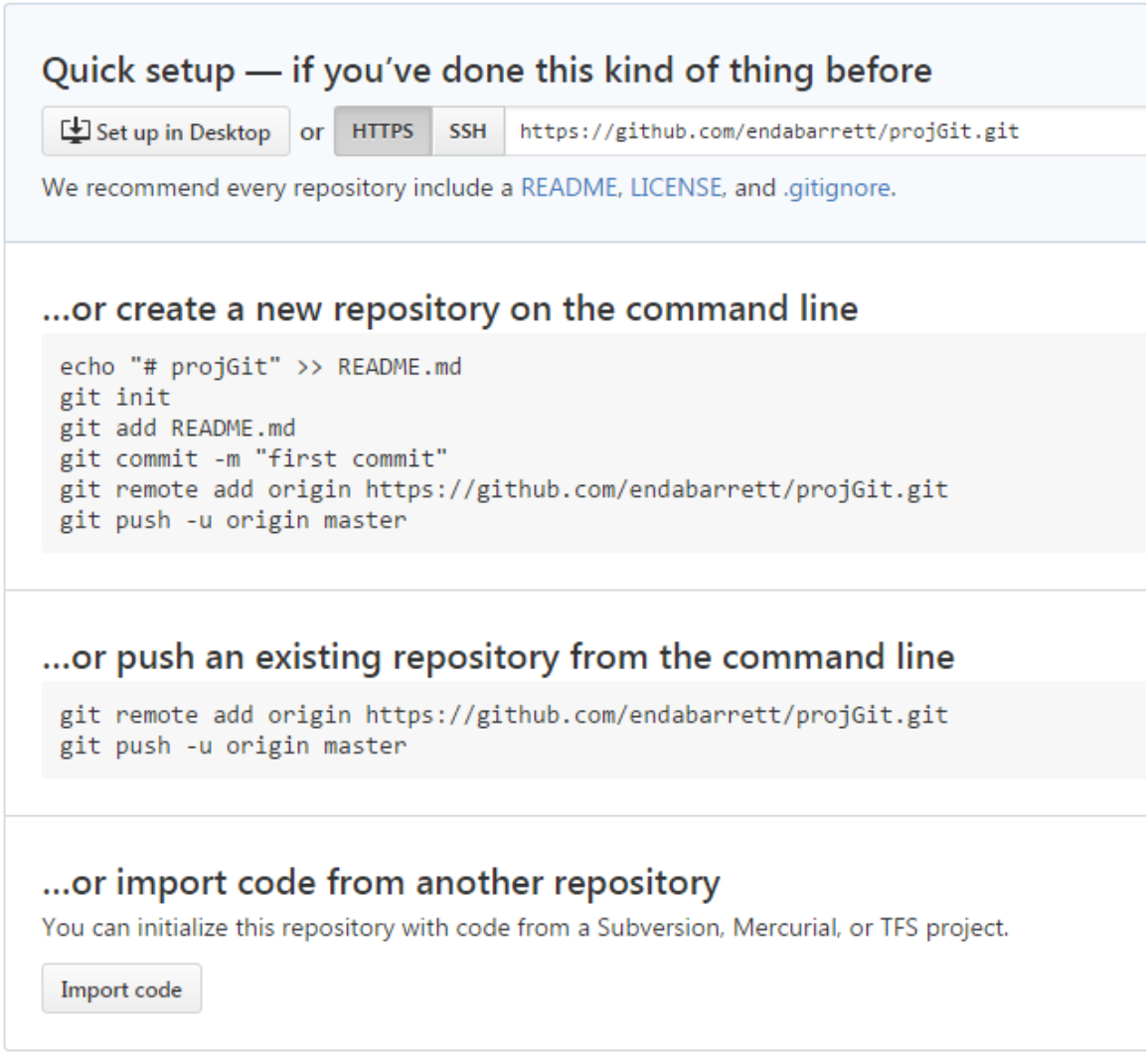
- Create an account on Github
  - It's free for public projects
- Create a new repository



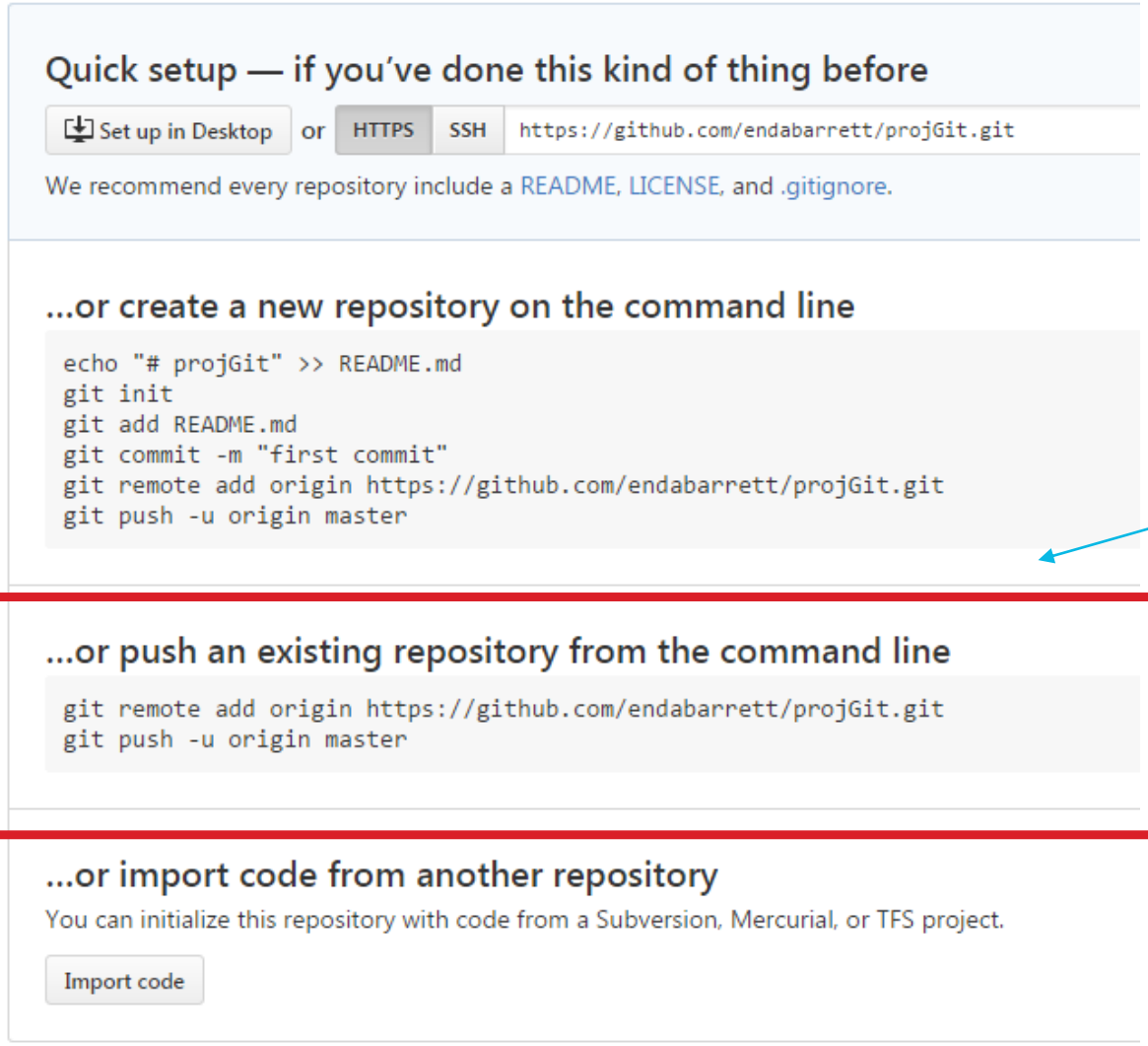
Name it and give it a description



Quick setup options



Pushing to a remote repository



Execute these commands to push to Github

Personal Access Token


```
C:\myCoolNewApp (main)
λ git push -u origin main
Username for 'https://github.com': endabarrett
Password for 'https://endabarrett@github.com':
remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.
remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.
fatal: Authentication failed for 'https://github.com/endabarrett/myCoolNewApp.git/'
```

More info

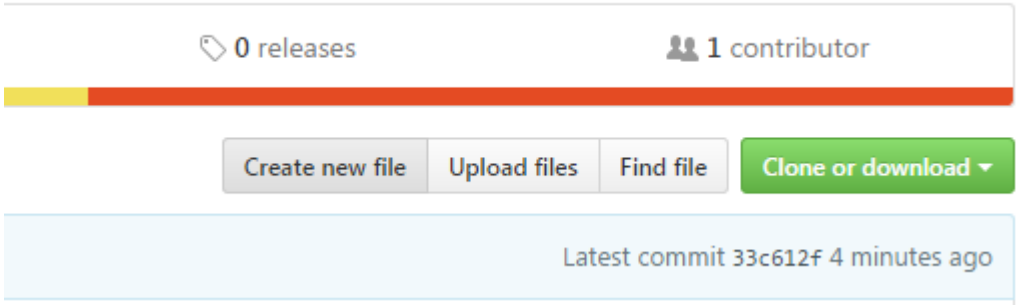
https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/

Steps involved in creating one, please follow these to create the token

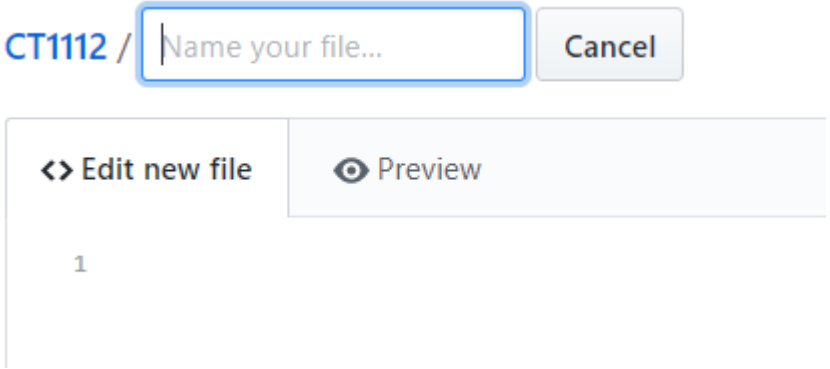
https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

Also a files on the web client Github

- Create new file



- Opens up an editor



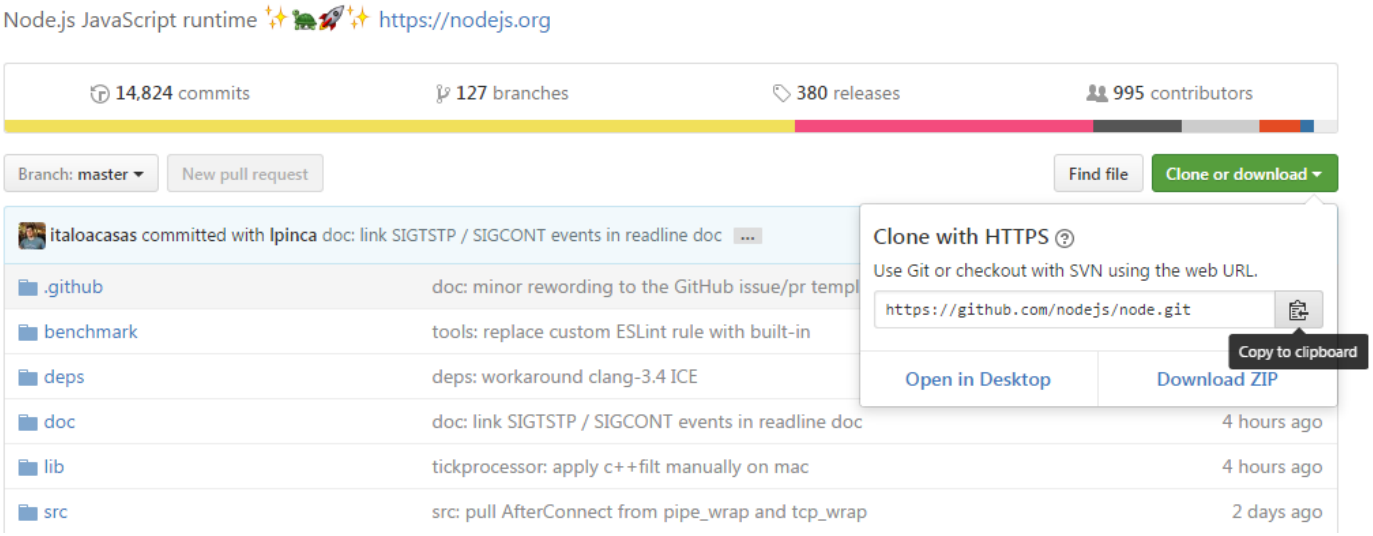
Note: Be careful though, if you are working locally make sure to pull any changes that were made via the web client

Update your repository

- git pull
  - Will pull the latest version from the repository you cloned.
- git fetch and merge
  - Pull is actually a combination of the two and you can run them separately

GIT – Cloning a repository

- Many projects don't require you to create your own repository, instead you clone it from a remote location such as GitHub
- git clone <repo>

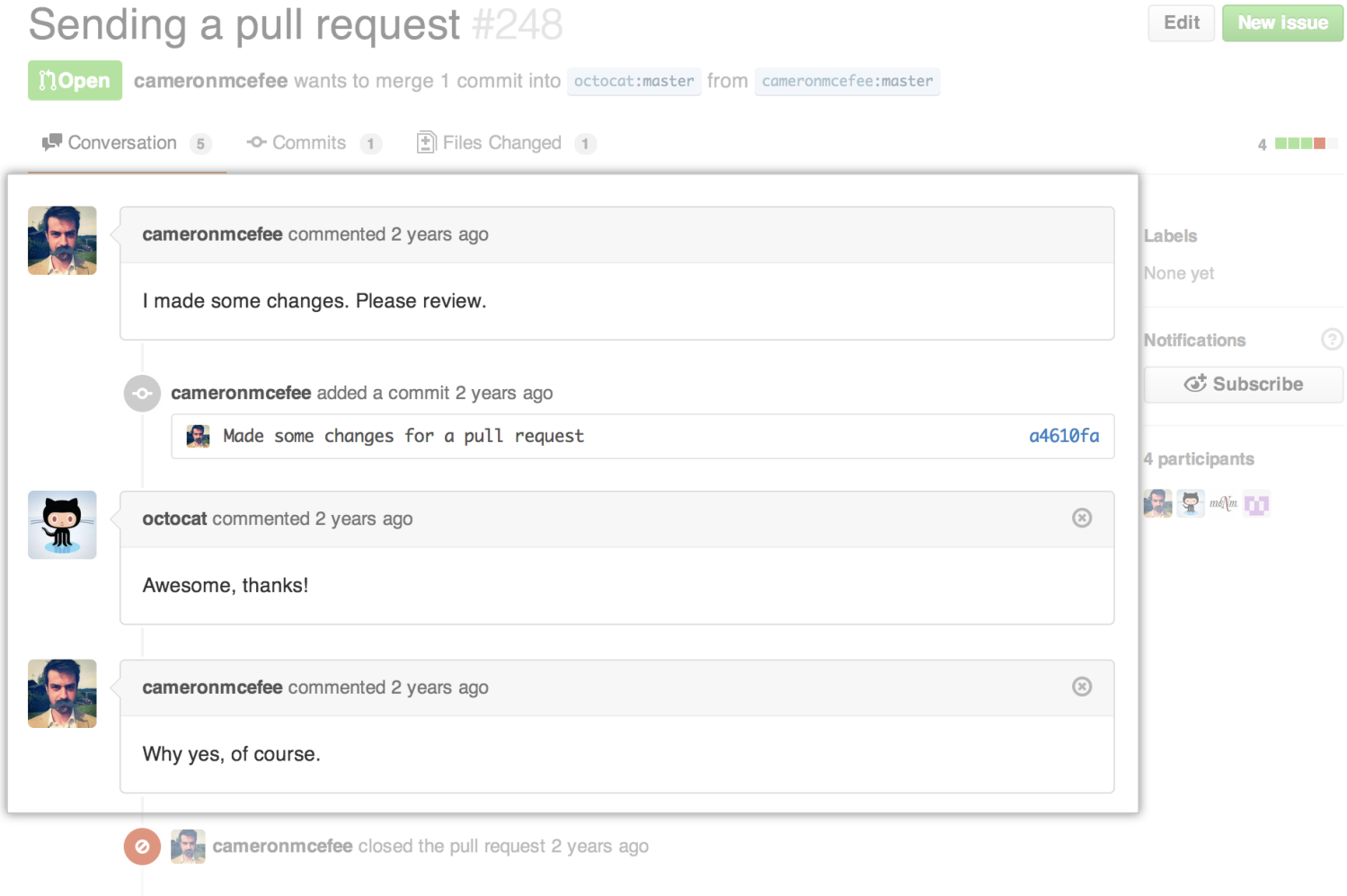


Pull requests

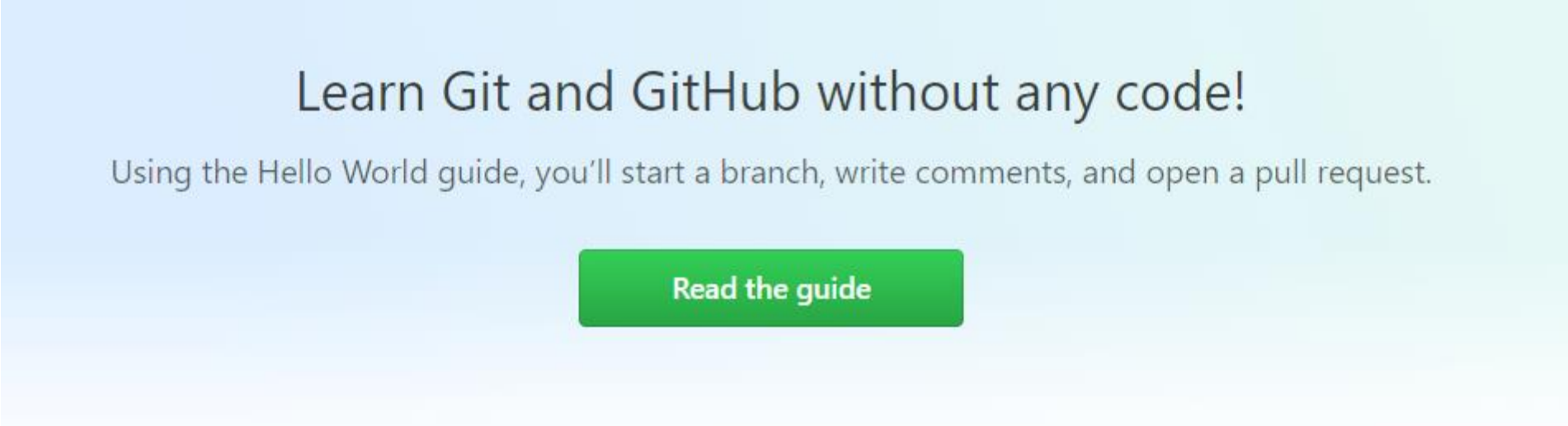
- If you make a change i.e. add a feature you can create a pull request.
- Everyone can review the code and decide whether or not it should be included in the *master* branch
- It's a forum for discussing the changes

- Git commands https://git-scm.com/docs

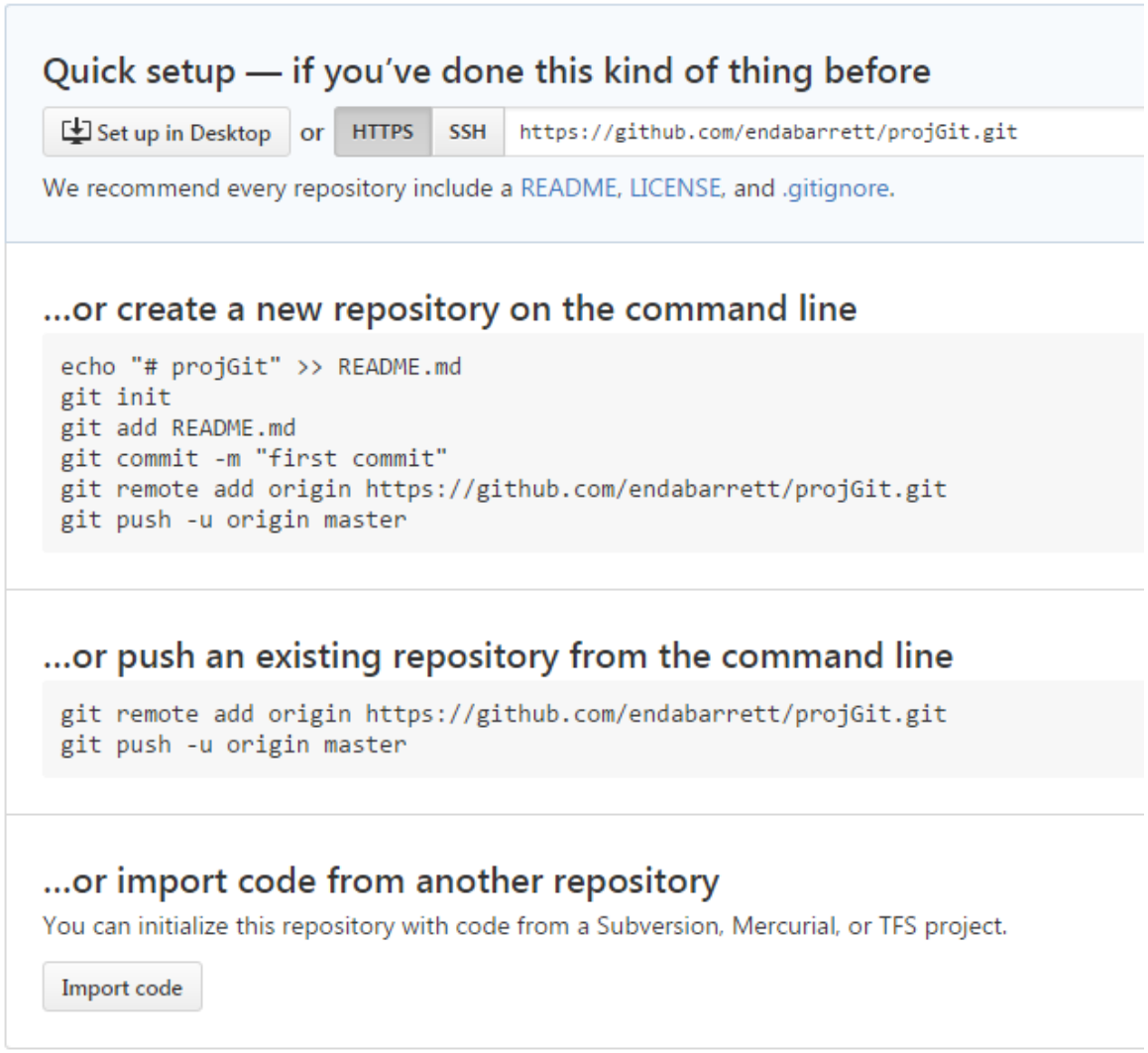
Sample pull request



Excellent Guide



Practical – Creating a repo



Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/endabarrett/projGit.git

We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# projGit" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/endabarrett/projGit.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/endabarrett/projGit.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

Commit some changes

- Make some changes to your app's codebase
  - git add –A
  - git commit
  - git push –u origin <master>

Creating branches (locally and remotely)

- To list all branches (local, remote)
  - git branch
  - git branch –r
- Adding a branch locally
  - git branch <branch_name>
- Push it to the remote repo
  - git push -u origin <branch_name>

Switching to a branch and commiting

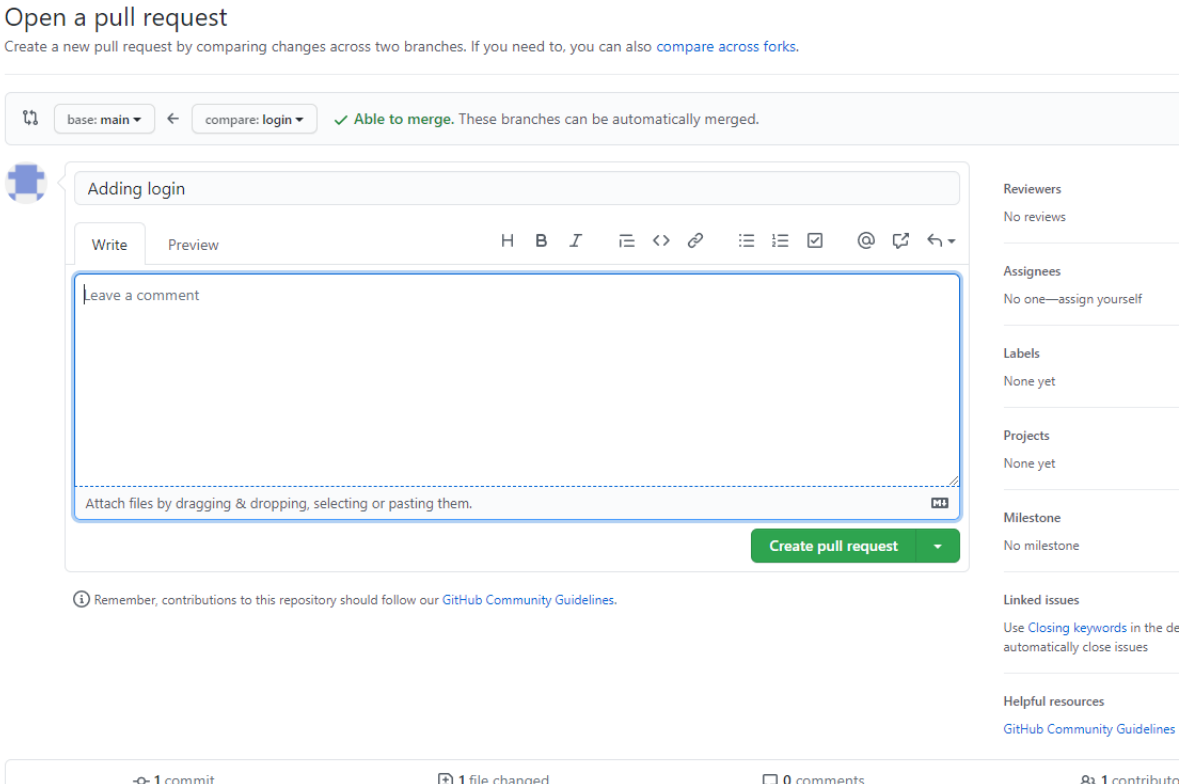
- Now that you've created a branch, we need to check it out so that we can start committing to it
  - git checkout <branch_name>
  - git add –A
  - git commit
  - git push –u origin <branch_name>

If the branch is already remote

- If one of your colleagues has created a branch and you want to work with it
  - git fetch
  - git checkout <branch_name>
  - git add –A
  - git commit
  - git push –u origin <branch_name>

Pull requests

- You've completed your feature and integrate it with the main master branch
- Best to do it using the GitHub web client



Deleting the branch

- Once you have finished and pulled your code into the main master branch, you may want to delete the branch
  - git branch -d <branch_name>
  - git push origin --delete <branch_name>