



By Dilum De Silva

(Software Engineer at Circles.Life Singapore)

So, what we gonna know by the end of 3 weeks...

Week One	Week Two	Week Three
<ul style="list-style-type: none">• Intro to 'frontend vs backend'• Intro to Play as a backend• Combining Play with front end tech (Concepts)• What is MVC pattern• Play setup and base play project structure	<ul style="list-style-type: none">• What the heck is REST APIs?• Concepts of REST APIs• Developing REST APIs with play.	<ul style="list-style-type: none">• Combining Play with a frontend (angular) and exposing an API (coding).

FRONTEND
VS
BACKEND

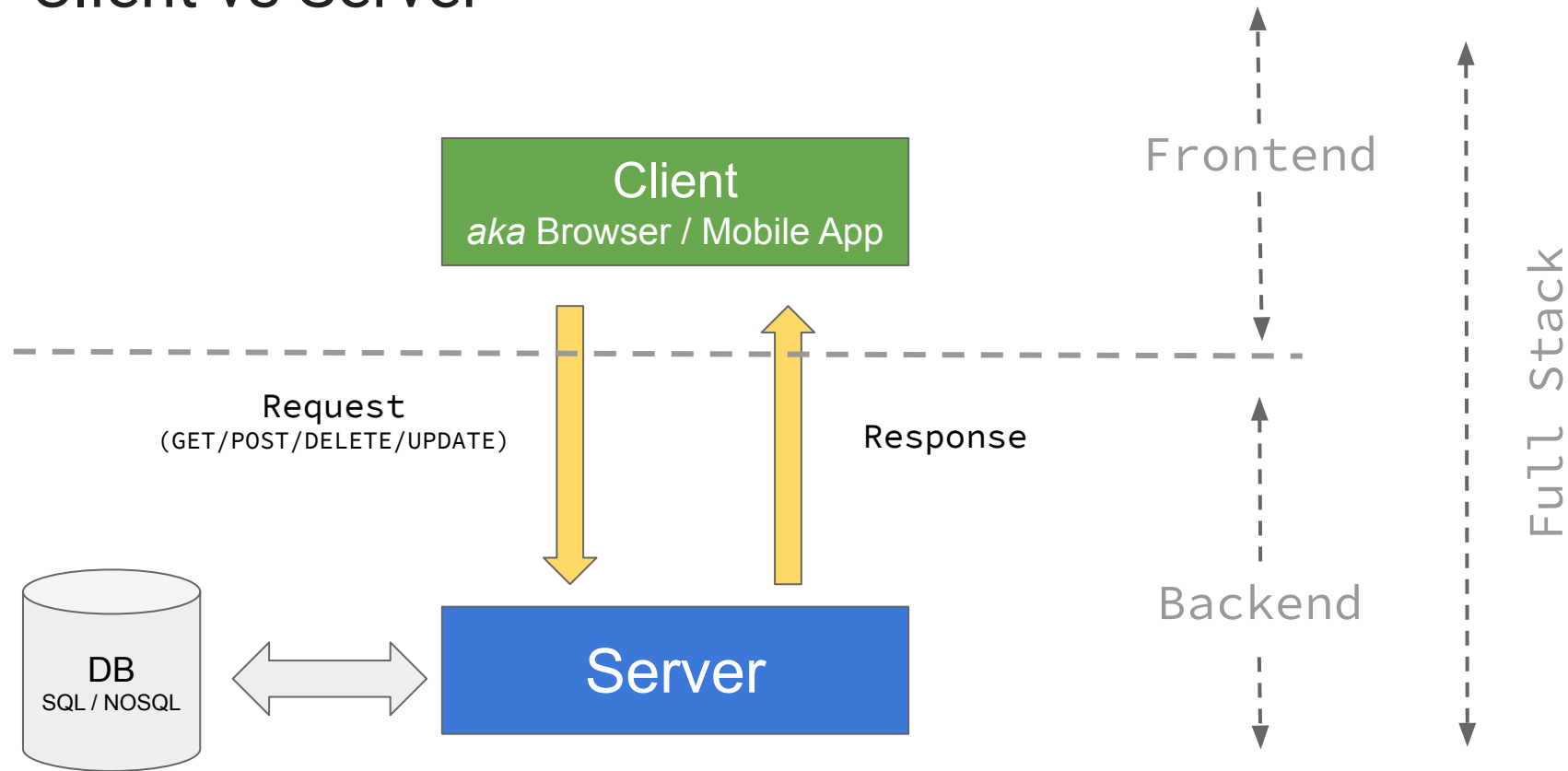
Frontend vs Backend



Frontend (client) vs Backend (server)



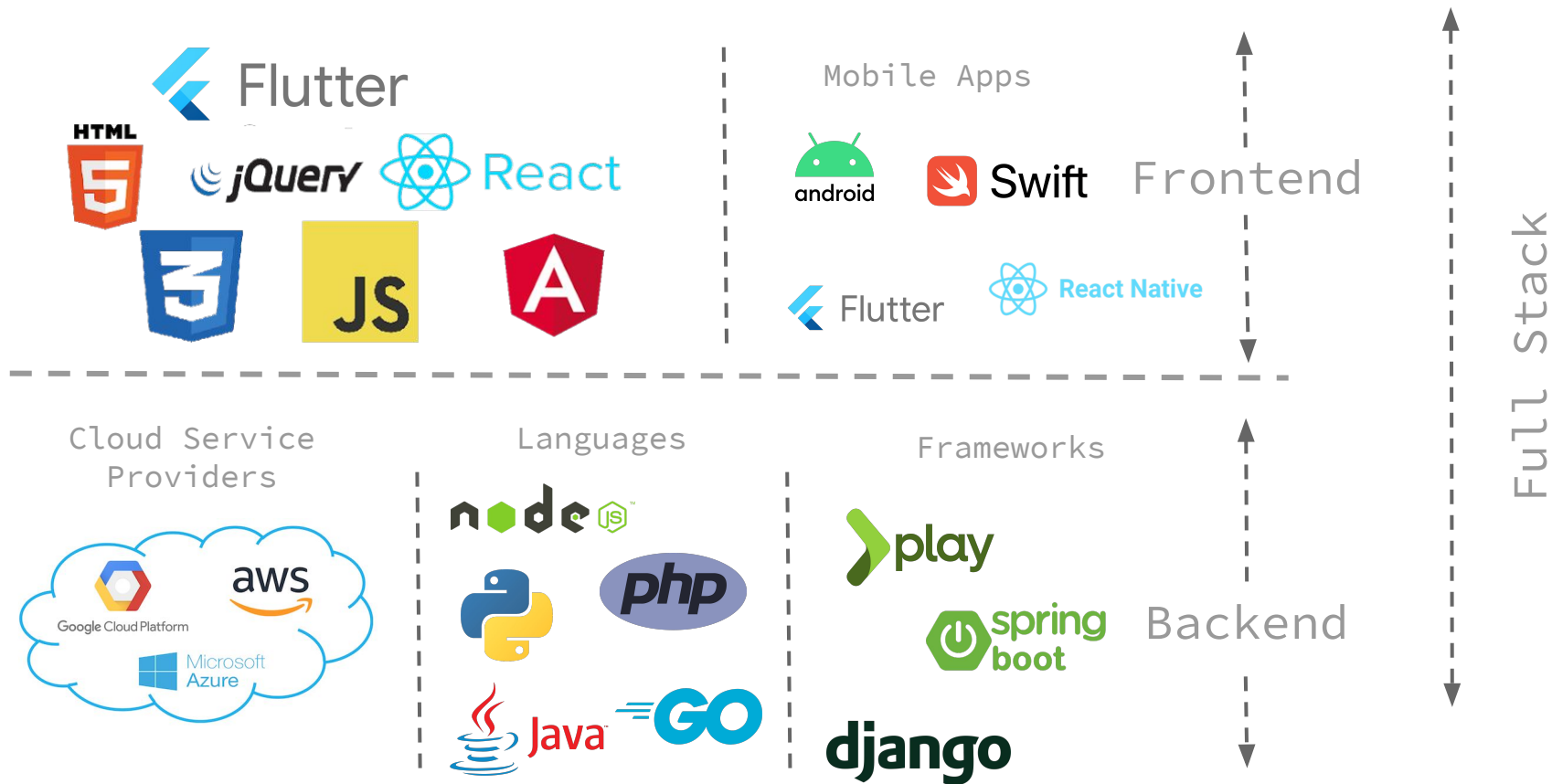
Client vs Server



Client vs Server in Real World



Client vs Server in Tech Perspective



FRONTEND DEVELOPMENT

#CLIENT

Frontend EVOLUTION
1995-2019
TWITTER.COM/MANZ

Legend:

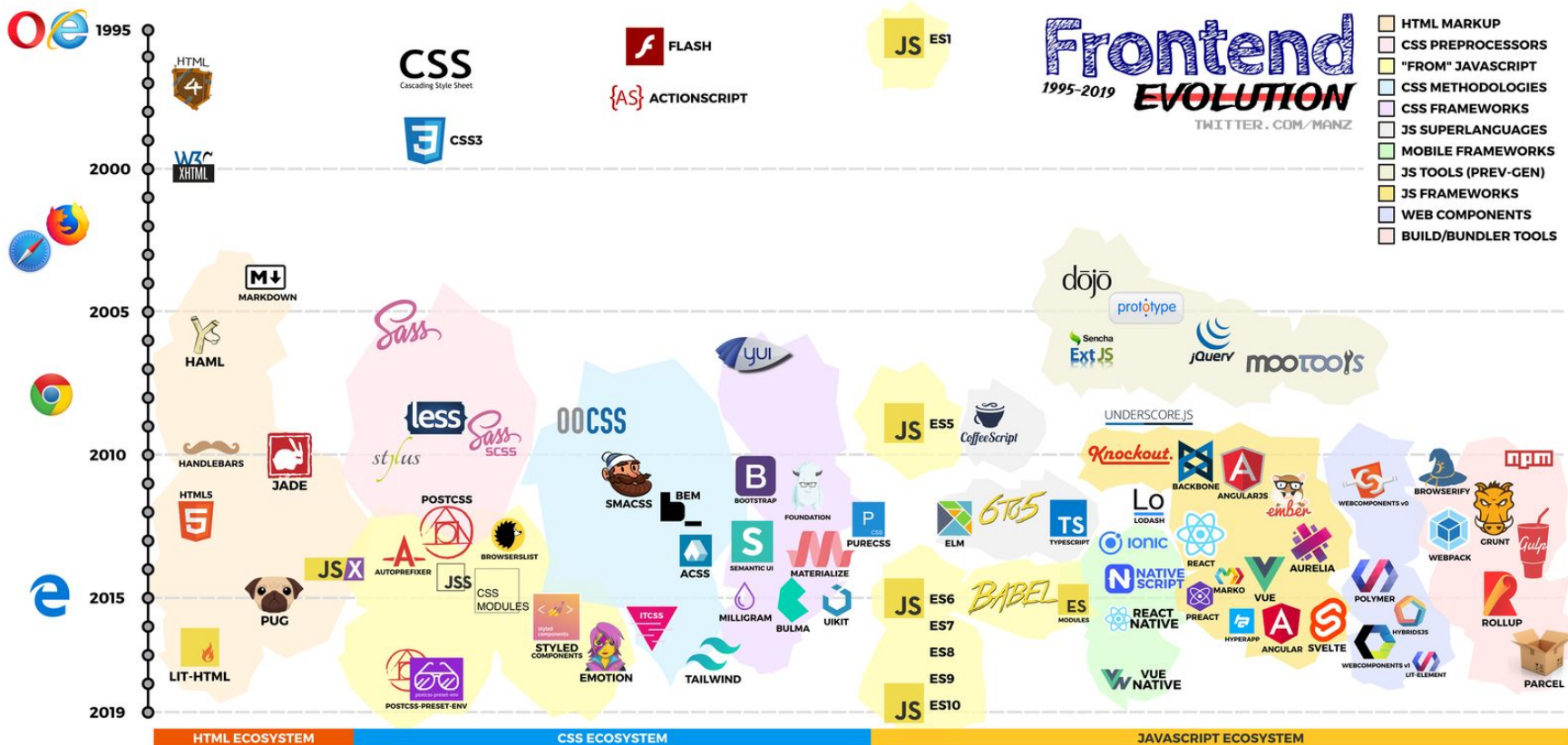
- HTML MARKUP
- CSS PREPROCESSORS
- "FROM" JAVASCRIPT
- CSS METHODOLOGIES
- CSS FRAMEWORKS
- JS SUPERLANGUAGES
- MOBILE FRAMEWORKS
- JS TOOLS (PREV-GEN)
- JS FRAMEWORKS
- WEB COMPONENTS
- BUILD/BUNDLER TOOLS

Timeline (1995-2019):

- 1995:** HTML 4, CSS (Cascading Style Sheet), JS ES1
- 2000:** W3C XHTML, CSS3
- 2005:** MARKDOWN, HAML, Sass, LESS, SCSS, OOCSS, YUI
- 2010:** HANDLEBARS, JADE, STYLUS, POSTCSS, SMACSS, BEM, BOOTSTRAP, FOUNDATION, PURECSS, ELN, 6T05, TS TYPESCRIPT, COFFEESCRIPT, JS ES5
- 2015:** PUG, JSX, AUTOPREFIXER, JSS, CSS MODULES, BROWSERSLIST, ACSS, SEMANTIC UI, MATERIALIZE, MILLIGRAM, BULMA, UIKIT, JS ES6, ES7, ES8, ES9, ES10
- 2019:** LIT-HTML, POSTCSS-PRESET-ENV, EMOTION, TAILWIND, DOJO, PROTOTYPE, SENCHA EXT JS, JAVERY, MOOTOOLYS, UNDERSCORE JS, KNOCKOUT, BACKBONE, ANGULARJS, REACT, NATIVE SCRIPT, REACT NATIVE, VUE NATIVE, MARKO, VUE, AURELIA, POLYMER, HYBRIDJS, WEBCOMPONENTS V0, WEBPACK, GRUNT, GULP, ROLLUP, PARCEL, WEBCOMPONENTS V1, LIT-ELEMENT

Ecosystems:

- HTML ECOSYSTEM:** HTML 4, W3C XHTML, MARKDOWN, HAML, HANDLEBARS, JADE, HTML5, PUG, LIT-HTML
- CSS ECOSYSTEM:** CSS, CSS3, Sass, LESS, SCSS, OOCSS, STYLUS, POSTCSS, SMACSS, BEM, BOOTSTRAP, FOUNDATION, PURECSS, ELN, 6T05, TS TYPESCRIPT, COFFEESCRIPT, JS ES5, JS ES6, ES7, ES8, ES9, ES10
- JAVASCRIPT ECOSYSTEM:** JS ES1, JS ES5, JS ES6, ES7, ES8, ES9, ES10, DOJO, PROTOTYPE, SENCHA EXT JS, JAVERY, MOOTOOLYS, UNDERSCORE JS, KNOCKOUT, BACKBONE, ANGULARJS, REACT, NATIVE SCRIPT, REACT NATIVE, VUE NATIVE, MARKO, VUE, AURELIA, POLYMER, HYBRIDJS, WEBCOMPONENTS V0, WEBPACK, GRUNT, GULP, ROLLUP, PARCEL, WEBCOMPONENTS V1, LIT-ELEMENT



If you want to become a frontend developer...

Technologies/ Languages

- HTML
- CSS
- JavaScript
- CSS Pre-processors (Sass, Stylus...)
- JavaScript Libraries (e.g. lodash) and Frameworks (Angular, React, Vue)
- Build Tools (npm, Webpack, ...)

You'll work on ...

- JS-driven User Interfaces
- Re-usable UI Components with JS logic and CSS Styling
- Forms & Input Validation
- Backend Communication Channels
- UX Strategies (PWAs, Live Updates)

Less Relevant Technologies/ Languages

- Server-side Languages (e.g. Node, PHP)
- Databases/ Query Languages (e.g. SQL)
- Server Configuration

You'll NOT work on ...

- Server-side Business Logic (e.g. User Authentication, Order Handling)
- Automatic E-Mail Notifications
- Database Access

BACKEND DEVELOPMENT

#SERVER

If you want to become a backend developer...

Technologies/ Languages

- Server-side Languages like Node, PHP
- Frameworks like Express, Laravel
- Databases & Query Languages
- Partly: Server Configuration
- Basic HTML, CSS, JavaScript

You'll work on ...

- Server-side Business Logic (e.g. User Authentication, Order Handling)
- Automatic Notifications
- Data Validation
- Data Storage/ Database Access
- Scheduled Processes

Less Relevant Technologies/ Languages

- Advanced JavaScript & CSS
- JavaScript Libraries & Frameworks
- Build Tools (npm, Webpack)

You'll NOT work on ...

- Client-side Validation
- Complex User Interfaces
- Advanced UX Strategies (PWAs, ...)

It's a matter of your choice and passion...

Frontend

Extremely Fast
Development
Technologies & the
Ecosystem



Growing Demand in
Developers

"Just the User Interface"

Backend

Security stays important,
ever more Data and User
Interactions require
"better" Algorithms and
Processes



Growing Demand in
Developers

"No direct Connection to
the User/ Customer"

Fullstack

No/ Little Dependencies on
other Developers, Full
Understanding of the
Complete Tech Stack



Perfect for Small
Companies & Freelancers

"Jack of all Trades"

FRONT-END VS BACK-END

Ruby

`console.log("Hi")`

CSS

VAR

HELLO WORLD

PHP

LET

`<HTML>`

`<code>`

`For (i = 0)`

`const`



LET'S DIVE
INTO

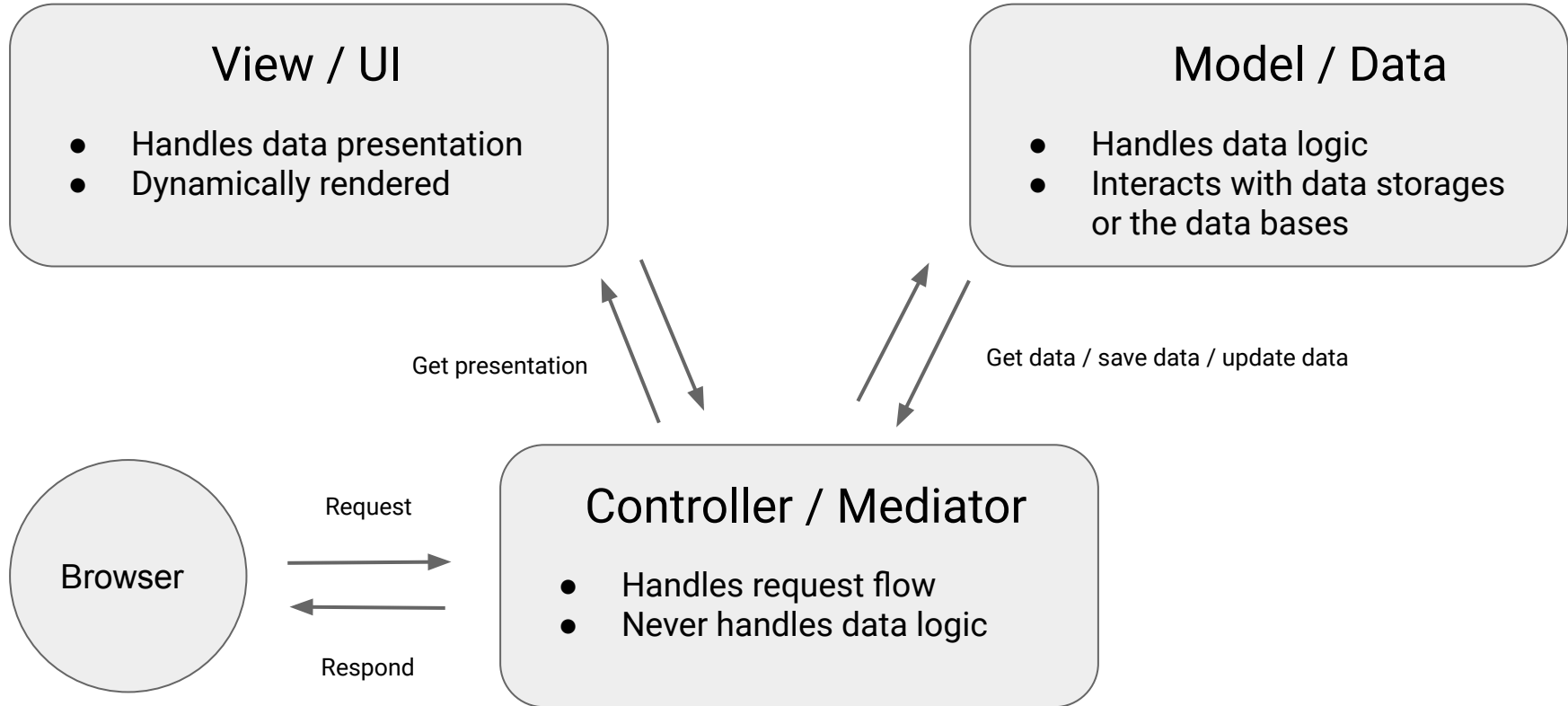


What is  play ?

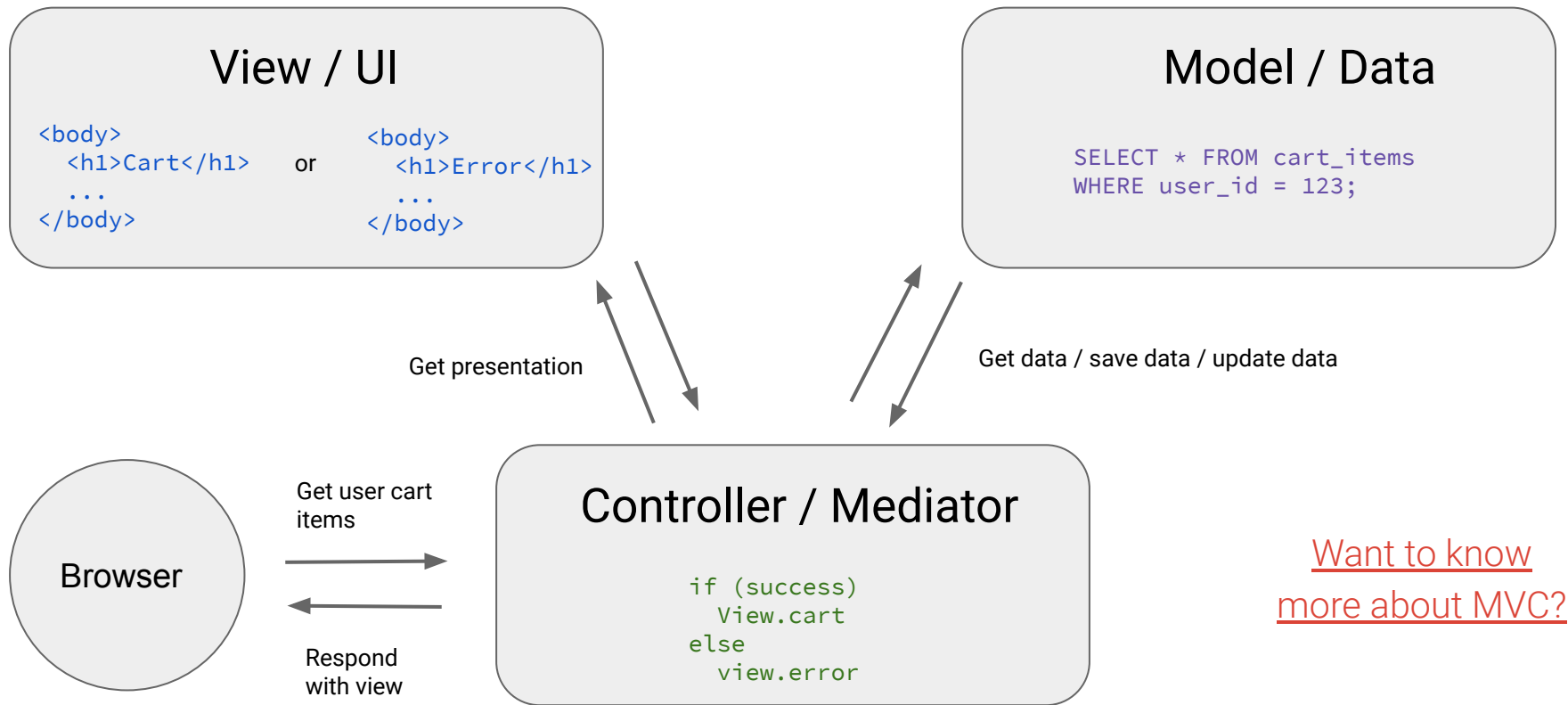
Play Framework is an **open-source web application framework** which follows the **model-view-controller (MVC)** architectural pattern. It is written in Scala and usable from other programming languages that are compiled to JVM Bytecode, e.g. Java

[Want to know more?](#)

What is MVC pattern ?



What is MVC pattern ?



MVC
Explained
in
8 Minutes

Why we should consider play ?

- Developer friendly
- Scalability and language compatibility
- Eco-System support (Java)
- Performance (compiled code runs on jvm)
- Modern web and mobile support
- Production ready and proven

Why we should consider  play ?



eero

theguardian

Walmart 



verizon[✓]

LinkedIn


NORWEGIAN
CRUISE LINE

SAMSUNG

 UniCredit Group

weightwatchers

zalando

Let's install

You need to have,

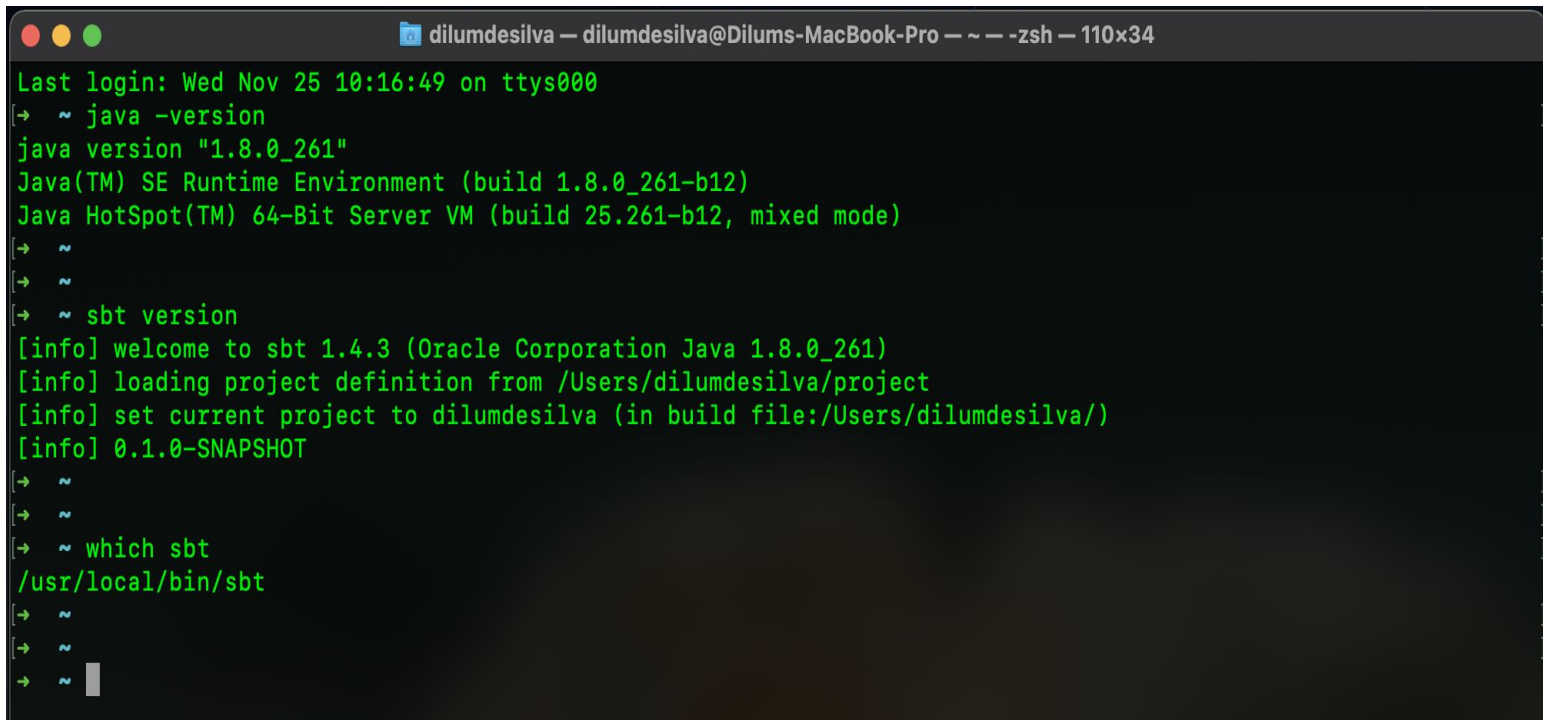
- Java 1.8 build (need to switch between multiple java version?)
- sbt latest - <https://www.scala-sbt.org/index.html>
- IDE (Prefer IntelliJ) - <https://www.jetbrains.com/idea>
- IDE plugins - scala and play

If you're on macOS

You need to have **homebrew** installed and if you recently updated to macOS **Big Sur** with **homebrew** you need to reinstall terminal tools.

<https://apple.stackexchange.com/questions/401899/homebrew-your-ctl-does-not-support-macos-11-0>

Let's verify our installations...



```
dilumdesilva — dilumdesilva@Dilums-MacBook-Pro — ~ — zsh — 110x34

Last login: Wed Nov 25 10:16:49 on ttys000
[→ ~ java -version
java version "1.8.0_261"
Java(TM) SE Runtime Environment (build 1.8.0_261-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.261-b12, mixed mode)

[→ ~
[→ ~
[→ ~ sbt version
[info] welcome to sbt 1.4.3 (Oracle Corporation Java 1.8.0_261)
[info] loading project definition from /Users/dilumdesilva/project
[info] set current project to dilumdesilva (in build file:/Users/dilumdesilva/)
[info] 0.1.0-SNAPSHOT

[→ ~
[→ ~
[→ ~ which sbt
/usr/local/bin/sbt

[→ ~
[→ ~
[→ ~
```



```
> sbt new playframework/play-java-seed.g8
```

app	→ Application sources
assets	→ Compiled asset sources
stylesheets	→ Typically LESS CSS sources
javascripts	→ Typically CoffeeScript sources
controllers	→ Application controllers
models	→ Application business layer
views	→ Templates
build.sbt	→ Application build script
conf	→ Configurations files and other non-compile resources (on classpath)
application.conf	→ Main configuration file
routes	→ Routes definition
dist	→ Arbitrary files to be included in your production distribution
public	→ Public assets
stylesheets	→ CSS files
javascripts	→ Javascript files
images	→ Image files
project	→ sbt configuration files
build.properties	→ Marker for sbt project
plugins.sbt	→ sbt plugins including the declaration for Play itself
lib	→ Unmanaged libraries dependencies
logs	→ Logs folder
application.log	→ Default log file
target	→ Generated stuff
resolution-cache	→ Info about dependencies
scala-2.13	
api	→ Generated API docs
classes	→ Compiled class files
routes	→ Sources generated from routes
twirl	→ Sources generated from templates
universal	→ Application packaging
web	→ Compiled web assets
test	→ source folder for unit or functional tests

COMBINE
PLAY WITH ANGULAR

Configuration 01

Build backend and frontend isolated in different projects and use REST interface to communicate.

Configuration 02

Build both backend and frontend in the same project, Use scala views to expose frontend entry point and communicate with backend using the REST interface.

Configuration 03

Build both frontend and backend in the same project:

Use play static routes to serve frontend and
communicate with backend using the REST interface.

This is the approach that we are planning to use.

Other resources

- Installing and maintaining multiple java versions using **jenv**.
 - <https://github.com/jenv/jenv>
- Installing and maintaining multiple node versions using **nvm**.
 - <https://github.com/nvm-sh/nvm>
- Play framework docs
 - <https://www.playframework.com/documentation/2.8.x/Home>

SO, WHAT'S
COMING NEXT

Homework



What is a
Rest API?



REST(API)

So, it's the week two

Week One	Week Two	Week Three
<ul style="list-style-type: none">• Intro to 'frontend vs backend'• Intro to Play as a backend• Combining Play with front end tech (Concepts)• What is MVC pattern• Play setup and base play project structure	<ul style="list-style-type: none">• What the heck is REST APIs?• Concepts of REST APIs• Developing REST APIs with play.	<ul style="list-style-type: none">• Combining Play with a frontend (angular) and exposing an API (coding).

Prerequisites



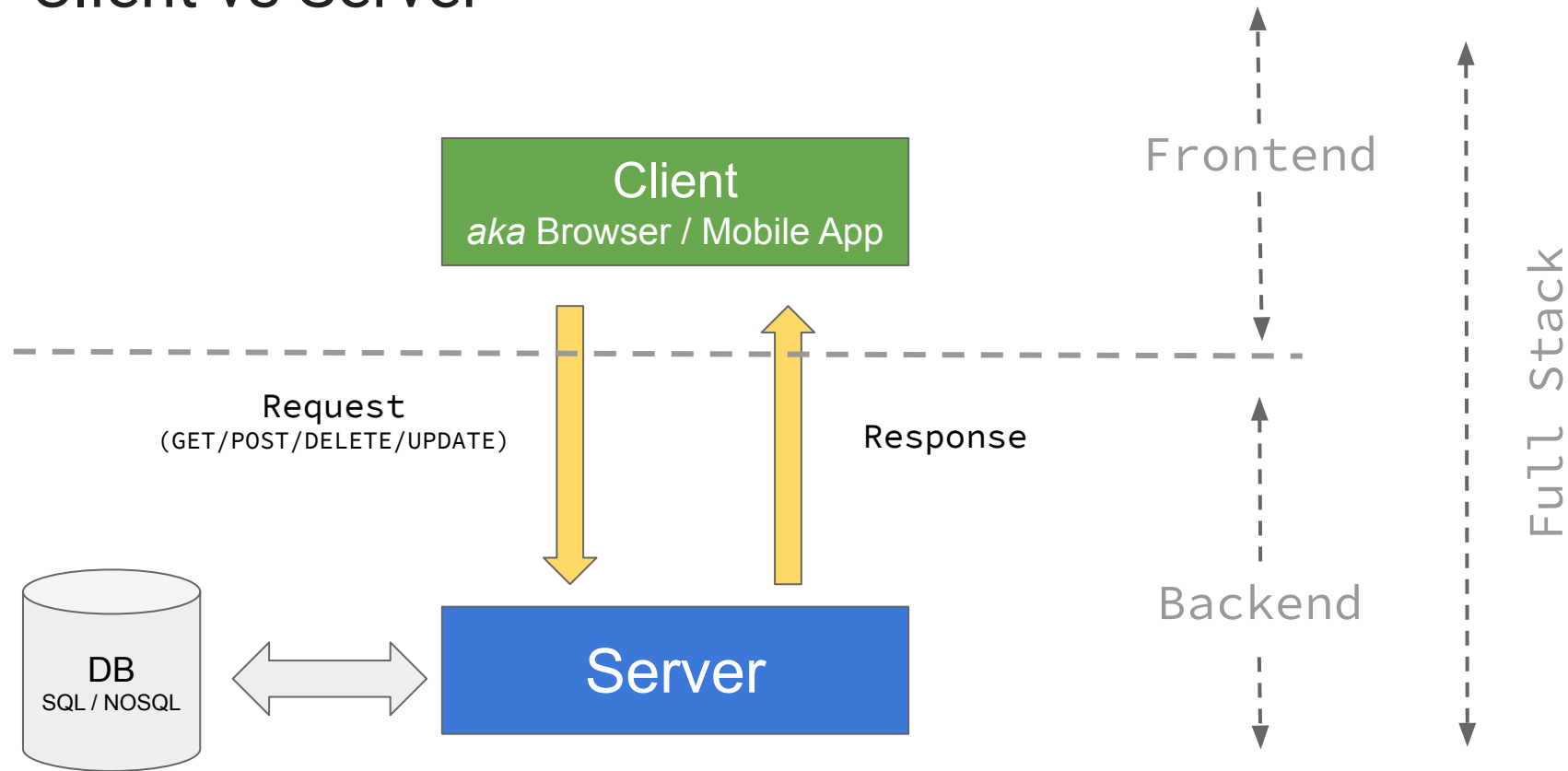
<https://www.postman.com>



<https://curl.se>

An **API** is an application programming interface. It is a set of rules that allow programs to talk to each other. The developer creates the API on the server and allows the client to talk to it.

Client vs Server





REST

REpresentational **S**tate **T**ransfer

REST determines how the API looks like. It stands for “**Representational State Transfer**”. It is a set of **rules** that developers follow when they create their API. One of these rules states that you should be able to get a piece of data (called a resource) when you link to a specific URL.

Anatomy of a request

- Endpoint
- Method
- Headers
- Data/body

<https://github.com/public-apis/public-apis>

Endpoint

The **root-endpoint** is the starting point of the API you're requesting from. The **root-endpoint** of Github's API is

<https://api.github.com>

The path determines the **resource** you're requesting for.

Ex: `http://localhost:9000/employee`

resource

root-endpoint

Testing - Endpoints

JavaScript users can use methods like the [Fetch API](#) and [jQuery's Ajax method](#).

Ruby users can use [Ruby's Net::HTTP class](#),

Python users can use [Python Requests](#); and so on.

Method

Do you know what are **CRUD** operations?

Hint:

C - Cre?, R - Rea?, U - Upd?, D - Del?

Methods

GET

- Fetch / read resource(s)
- Path Params + Query Params
- No body

POST

- Create new resource(s)
- Path Params + request body

PUT

- Update existing resource - full update
- Path Params + request body

DELETE

- Delete an existing resource - (for critical resources do soft delete)
- No query params
- No body

Methods

PATCH

- Modify existing resource - partial update
- To make it idempotent and safe from race conditions -
 - either pass a last accessed timestamp, or
 - present values of the fields being updated as filters in the body for conditional partial update.
- Path Params + request body

Method

Method Name	Request Meaning
'GET'	This request is used to get a resource from a server. If you perform a 'GET' request, the server looks for the data you requested and sends it back to you. In other words, a 'GET' request performs a 'READ' operation. This is the default request method.
'POST'	This request is used to create a new resource on a server. If you perform a 'POST' request, the server creates a new entry in the database and tells you whether the creation is successful. In other words, a 'POST' request performs an 'CREATE' operation.
'PUT' and 'PATCH'	These two requests are used to update a resource on a server. If you perform a 'PUT' or 'PATCH' request, the server updates an entry in the database and tells you whether the update is successful. In other words, a 'PUT' or 'PATCH' request performs an 'UPDATE' operation.
'DELETE'	This request is used to delete a resource from a server. If you perform a 'DELETE' request, the server deletes an entry in the database and tells you whether the deletion is successful. In other words, a 'DELETE' request performs a 'DELETE' operation.

Headers

Headers are used to provide information to both the client and server. It can be used for many purposes, such as authentication and providing information about the body content.

You can find a list of valid headers on MDN's [HTTP Headers Reference](#).

HTTP Headers are property-value pairs that are separated by a colon. The example below shows a header that tells the server to expect JSON content.

Data/body

The data (sometimes called “body” or “message”) contains information you want to be sent to the server.

This option is only used with **POST**, **PUT**, **PATCH** or **DELETE** requests.

To send data through cURL, you can use the `-d` or `--data` option:

HTTP Status Codes & Error Messages

- **200+** means the request has succeeded.
- **300+** means the request is redirected to another URL
- **400+** means an error that originates from the client has occurred
- **500+** means an error that originates from the server has occurred

FOR MORE ABOUT REST APIS

<https://github.com/dilum1995/IIT-PlayFramework-Session>

PLAY
+
ANGULAR

So, where are we now...

Week One	Week Two	Week Three
<ul style="list-style-type: none">• Intro to 'frontend vs backend'• Intro to Play as a backend• Combining Play with front end tech (Concepts)• What is MVC pattern• Play setup and base play project structure	<ul style="list-style-type: none">• What the heck is REST APIs?• Concepts of REST APIs• Developing REST APIs with play.	<ul style="list-style-type: none">• Combining Play with a frontend (angular) and exposing an API (coding).

Configuration 01

Build backend and frontend isolated in different projects and use REST interface to communicate.

Config 01 Pros/Cons

Build backend and frontend isolated in different projects and use REST interface to communicate

Pros	Cons
Frontend and backend concerns are separated out to different projects.	Isolated deployment steps are required for both backend and frontend.
Frontend can be deployed in a CDN.	Need to enable <u>CORS</u> to directly access services exposed from other origins or same origin different ports
	All the API's are exposed to the external parties with cross origin accessibility
	Development experience is bad (one command run project is not available).

Configuration 02

Build both backend and frontend in the same project, Use scala views to expose frontend entry point and communicate with backend using the REST interface.

Config 02 Pros/Cons

Build both backend and frontend in the same project, Use scala views to expose frontend entry point and communicate with backend using the REST interface

Pros	Cons
Play activator associated commands (run, build, publish, etc..) to triggers frontend tasks associated (no need to worry how things get wired up).	Have to manage frontend code in multiple places (index html within scala views and rest of the code in frontend sub directory).
No more <u>CORS</u> hassle, secure out of the box.	No clear separation between frontend and backend code.
Easy CI/CD integration.	Frontend cannot be deployed in a <u>CDN</u> directly (It is technically feasible using a <u>reverse proxy</u>).

Configuration 03

Build both frontend and backend in the same project:
Use play static routes to serve frontend and
communicate with backend using the REST interface.

This is the approach that we are planning to use.

Config 03 Pros/Cons

Build both frontend and backend in the same project: Use play static routes to serve frontend and communicate with backend using the REST interface.

Pros	Cons
Play activator associated commands (run, build, publish, etc..) to triggers frontend tasks associated (no need to worry how things get wired up).	Need to write proxy API's to access micro services to ensure security (which is not a bad thing anyway).
Painless development experience (with play run hooks). Watch task with live reload and what not.	Frontend cannot be deployed in a CDN directly (It is technically feasible using a reverse proxy).
No more CORS hassle, secure out of the box.	
Frontend code is served via the same server instance which is serving the backend.	
Easy CI/CD integration.	

LET'S START COMBINING

<https://github.com/dilum1995/IIT-PlayFramework-Session>