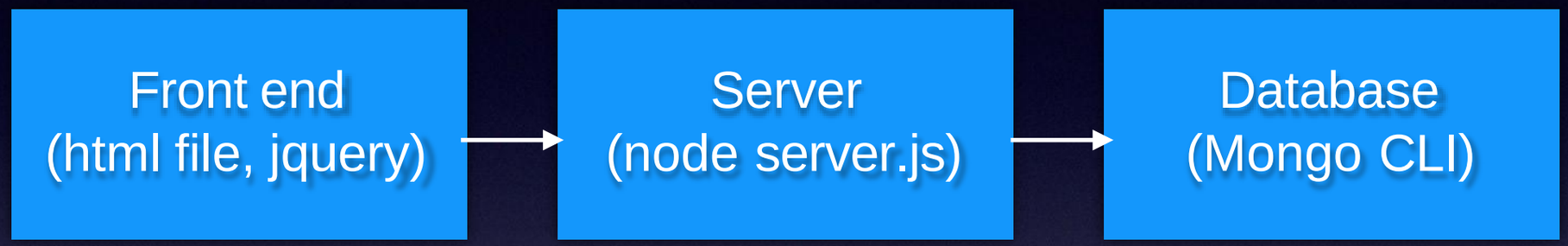
Lecture 5

Design patterns and software “rules”

This lecture covers:

- A little bit about previous labs
- Software design patterns: Model View Controller
- A little bit about this lab

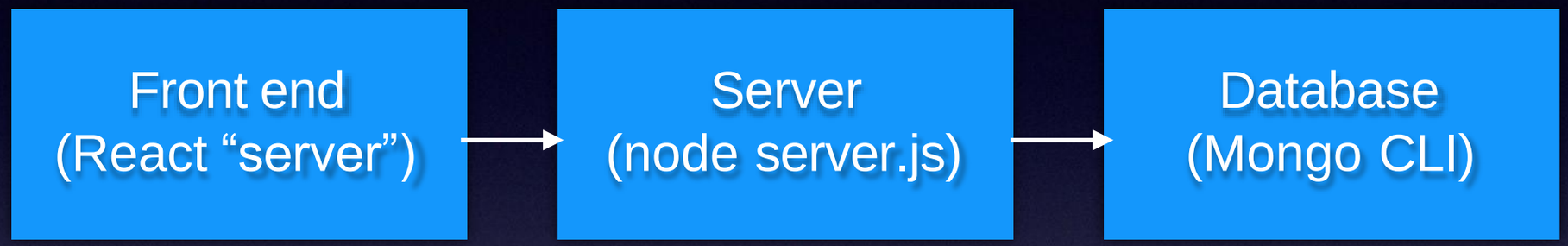
Lab 3 (Adding a Database)



Stored stuff to a database but was missing several key (Security) features:

- No user accounts, no actual user login (or routes with middleware)
- No encryption (no passwords, hash or salt)
- May not have used HTTP/REST consistency in the API

Lab 4: React (plus a database)



- The easy part of the lab was to introduce React as a front end framework.
- The slightly harder part was making your React front-end play with an existing back-end (API and database).

Lab 4: React with its own backend

Front end
(React “server”)

Server
(Next.js)

- We also used the native Next.js as a server and API.
- The nice part of this was that CORS was no longer necessary (but it makes the server less flexible).

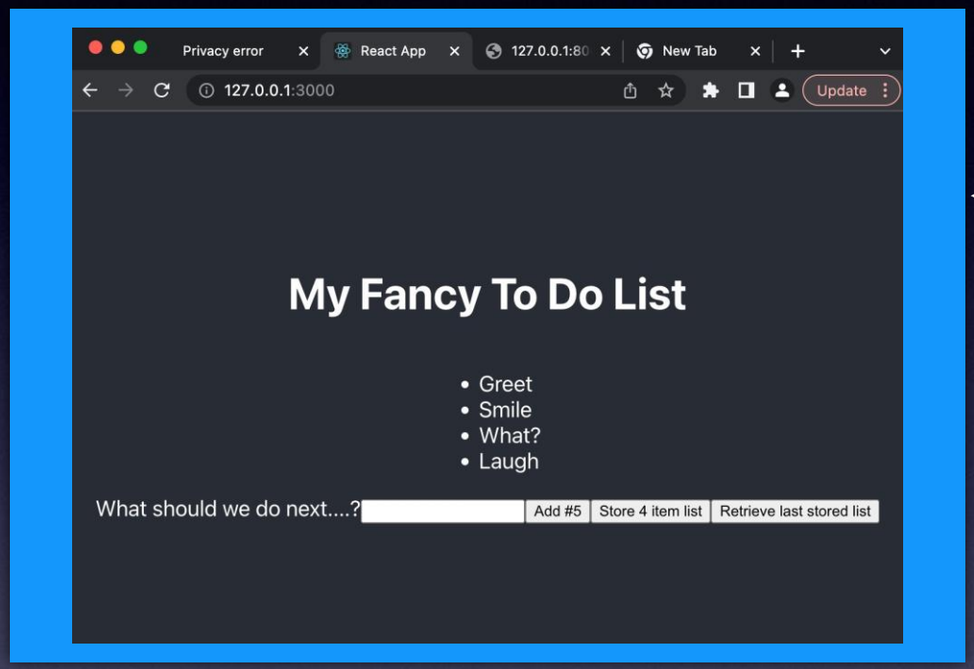
Lab 4: An error?

PS C:\Users\pj8227\Documents\react_stuff> npx create-next-app@latest
npm error code ENOENTnpm error syscall lstat
npm error path C:\Users\pj8227\AppData\Roaming\npmnpm error errno -4058
npm error enoent ENOENT: **no such file or directory**, lstat
'C:\Users\pj8227\AppData\Roaming\npm'
npm error enoent

This is related to npm not being able to find a file.

Create the directory it wants!
(Also, No to TypeScript?)

Aside about React…



JavaScript

React “Server”
(Your JSX “transpiled” into JavaScript for the browser)

- Yes, technically React is “served” by a “front end server” that might be separate to the (headless) “back end server” (i.e. node).

Lab solutions might not work out of the box…

- **My** (raw) solutions have **my** variables/URIs in them
- To run them, you will need to run **npm install** to get the node_modules folder
- You will also need to know the functions to start servers and database
- And you will need to find any environment variables and optional variables and create suitable alternatives to them

In short, you will need to customise the code. So, copy code snippets and patterns or be a diligent editor.

Some debugging tips

- Are you editing the right file?
- Have you hit “save”?
- Do you need to start or restart anything?
- Do you need to install anything?
- Is there a firewall in the way?
- Do you need to wait a minute?

git status example

```
RG45609:web pam$ git status
On branch master
Your branch is ahead of 'origin/master' by 3 commits.
  (use "git push" to publish your local commits)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   Magic Hate Ball/hate_ball.php
        new file:   Magic Hate Ball/hate_ball_frontend.html
        new file:   Magic Hate Ball/hate_ball_get_post.php
        new file:   Magic Hate Ball/hate_ball_get_post_no_format.php

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        navigation2.php
```

- It tells me I need a “git push”
- It tells me I git added some new files (that are not committed yet) - I need a “git commit”
- It tells me that there is a file that is untracked

Git Login faffing

Install/use gh:

sudo apt-get install gh

gh auth login

Finding the bug

- Do you have everything you need running?
- Is the bug in the front end or the back end or the database?
- Build bug or run-time bug?
- package.json, syntax error, path error, missing environment variable, stubborn firewall?

Debugging Things to Try

- Is there an error in the browser’s Developer Console?
- Can you craft a URL using cURL/Postman and does the backend work with that?
- Is the database registering any activity at all?

Software Design Patterns

- What is a design pattern?
- Why should I use one?
- How do I use one?

What is a software design pattern?

- It’s a set of rules or a template that guides you on how to structure code
- It is a means of deciding what goes where
- It is **not** tied to any specific language
- Some frameworks follow a design pattern by default (e.g. Angular uses MVC)

Why should I use a software design pattern?

- To aid decision making (if a pattern dictates where some functionality should go, you don't have to make a decision).
- To make code less complicated and more easily understood by **everyone**.
- To enable code reuse

Developer Virtues

- Laziness - you should go to great lengths to save your future self some effort.
- Impatience - why should you do this again and again when you can get your computer to do it for you?
- Hubris - because if other people don’t think your code is fabulous, they will simply rewrite it better.

DRY

Don’t **R**epeat **Y**ourself

- Understand abstraction
- Reuse code where possible
- Write code so it’s reusable

For Example….

A website for strong men could look like this:

thebeast.jpg

thebeast.js

thebeast.html

thebeast.css?

thebeast

mountain.jpg

mountain.js

mountain.html

mountain

stoneking.jpg

stoneking.js

stoneking.html

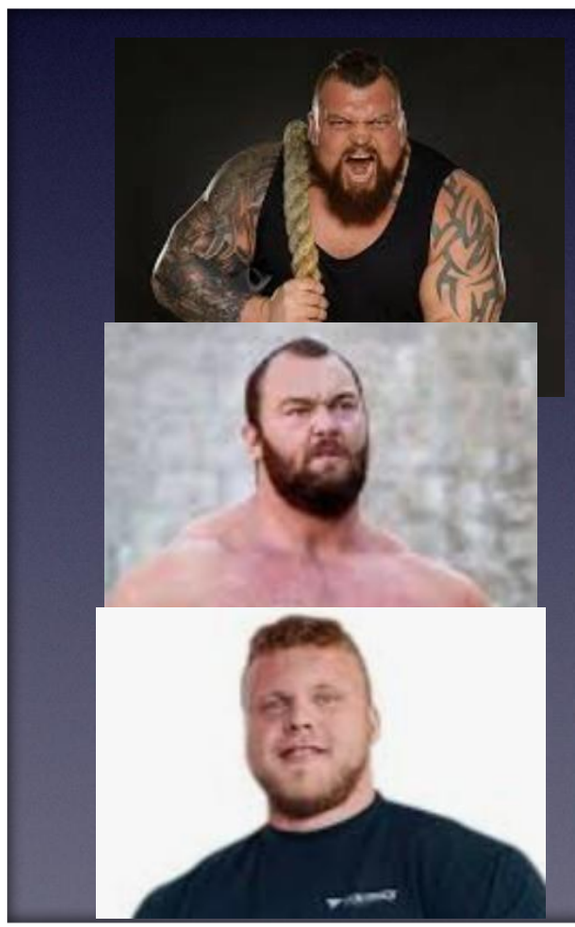
stoneking

And code like this:

```
<!DOCTYPE html>
<html>
<!-- Thebeast.html –>
<script src="thebeast.js"></script>
<body>


<h1>The Beast</h1>


<h2>Name: Eddie Hall</h2>

<img src="thebeast.jpg" alt="The Beast"
width="104" height="142">


<div style="border: 1px solid black”>Eddie
eats things</div>

<button type="button"
onclick="likeFunction()"> Click to like </
button>


</body>

</html>
```

```
<!DOCTYPE html>
<html>
<!-- Themountain.html –>
<script src="mountain.js"></script>
<body>


<h1>The Mountain</h1>


<h2>Name: Hafthor Bjornsson</h2>

<img src="mountain.jpg" alt="The mountain”
width="104" height="142">


<div style="border: 1px solid black”>Eddie
eats things</div>

<button type="button"
onclick="likeFunction()"> Click to like </
button>


</body>

</html>
```

But it’s far easier to extend if it looks like this:



images

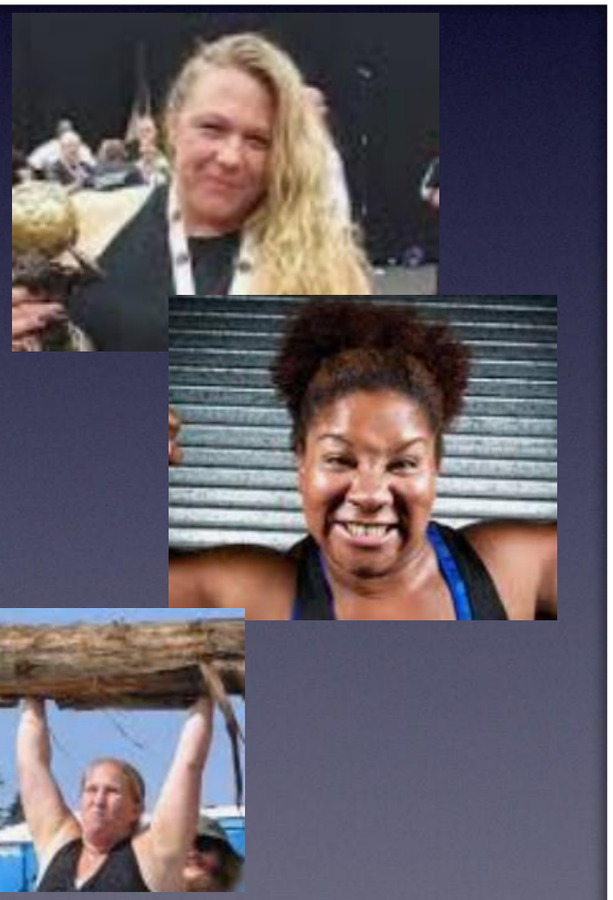
profile.html
profile.js
profile.css

src

Name	AKA
Eddie Hall	The Beast
Hafthor Bjornsso	The Mountain
Tom Stoltman	The Albatross

database

And then the code can be reused:



images

profile.html
profile.js
profile.css

src

Name	AKA
Donna Moore	null
Andrea Thompso	null
Kristin Rhodes	null

database

If you write your code to be reusable

- You can reuse it!
- If you find one bug in it, you might have fixed many bugs
- You save on typos

If you find you're doing an awful lot of "find and replace", you probably haven't got the level of abstraction quite right.

So, how do we make code reusable?

- Split large chunks of code into reusable components
- You will end up with a lot of files
- But the files will contain reusable components
- And if you’re working in a group, individuals can be responsible for individual components

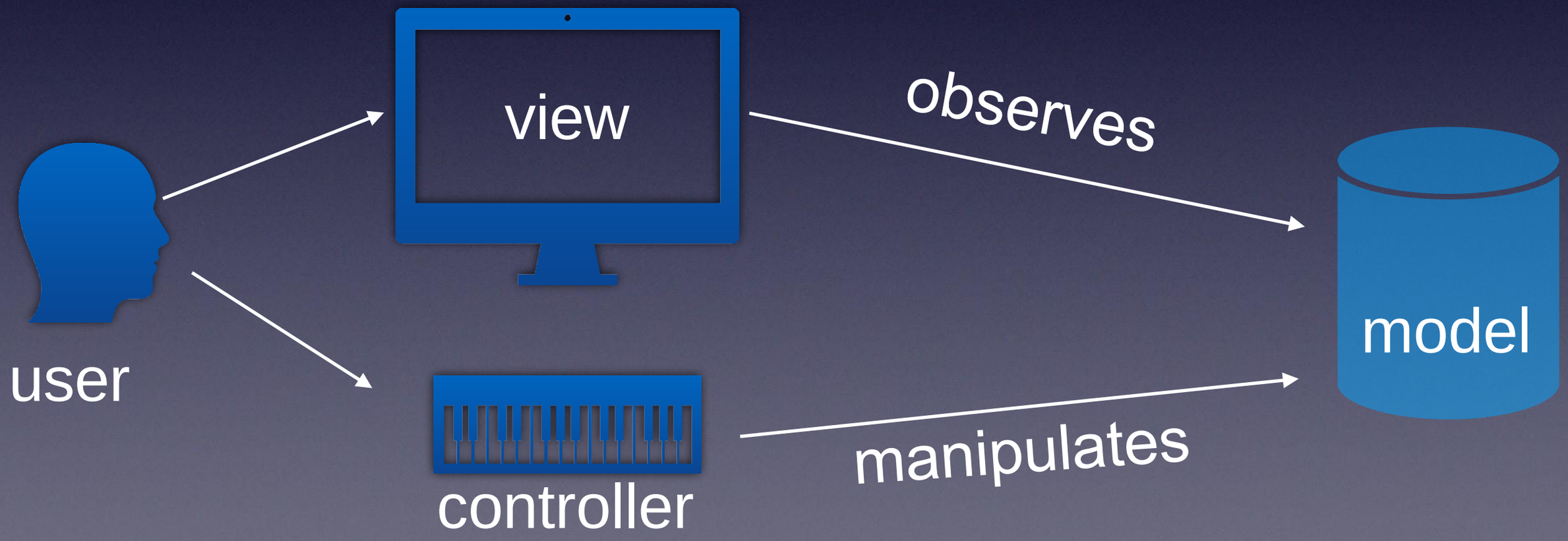
And how do we make working in teams easy?

- Use version control
- Define a code architecture or software design pattern
- Even better, use an existing software design pattern

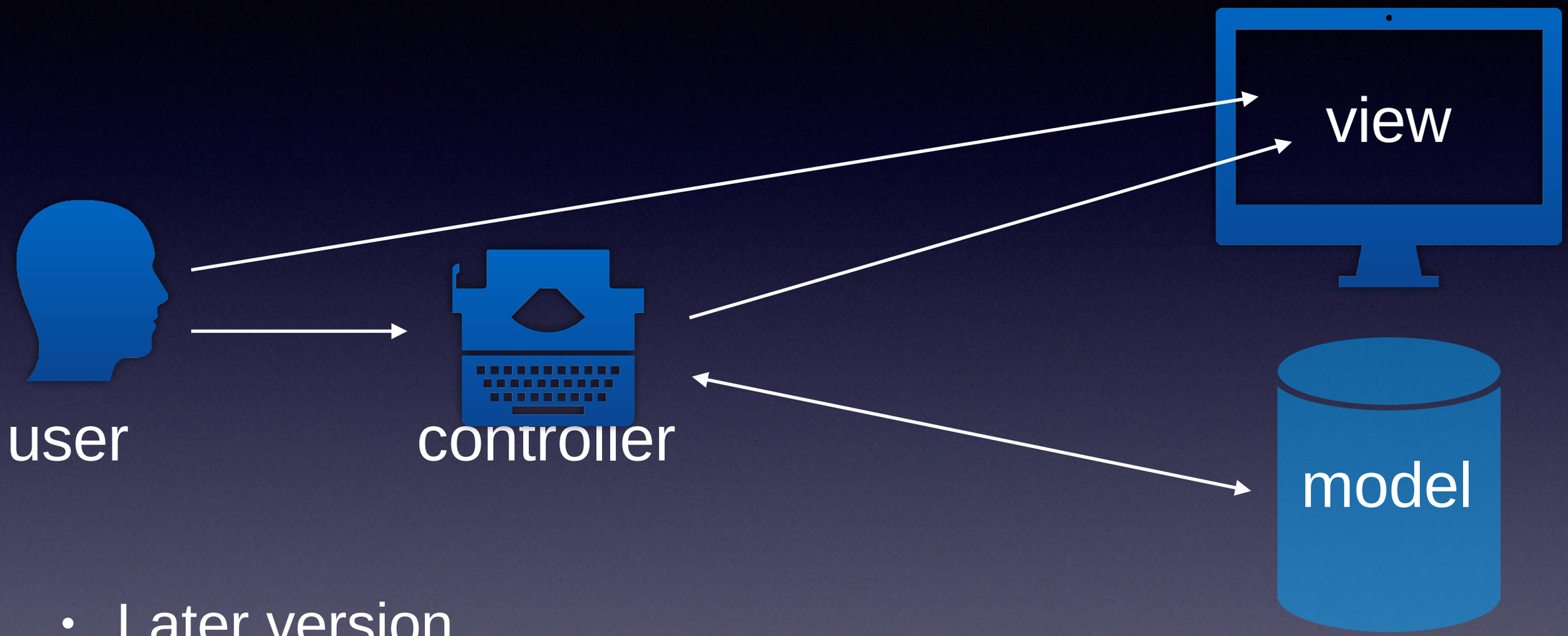
Model View Controller

What is MVC?

It’s a **software architecture pattern** which is used to separate the model, view and controller.

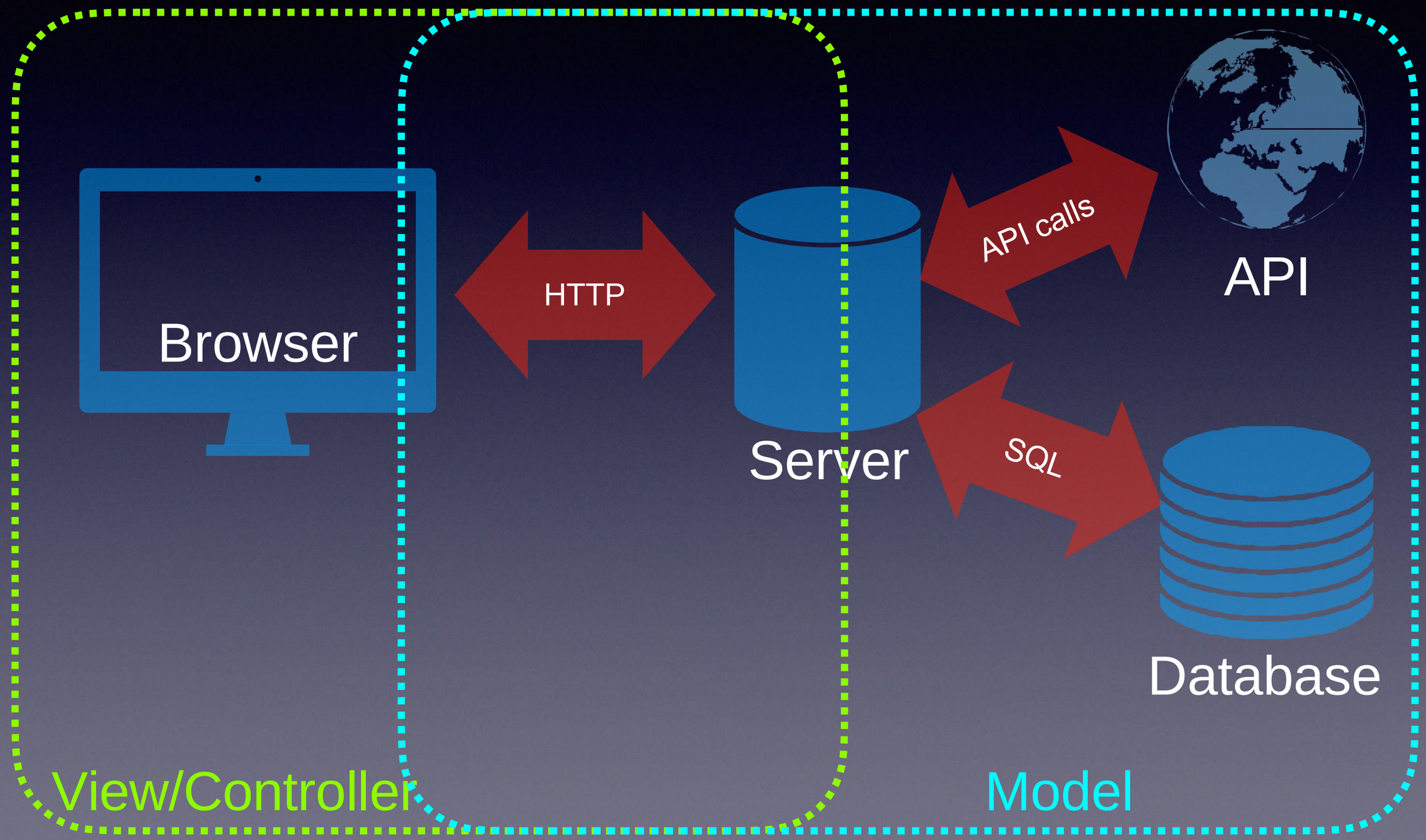


But you might also see it like this:



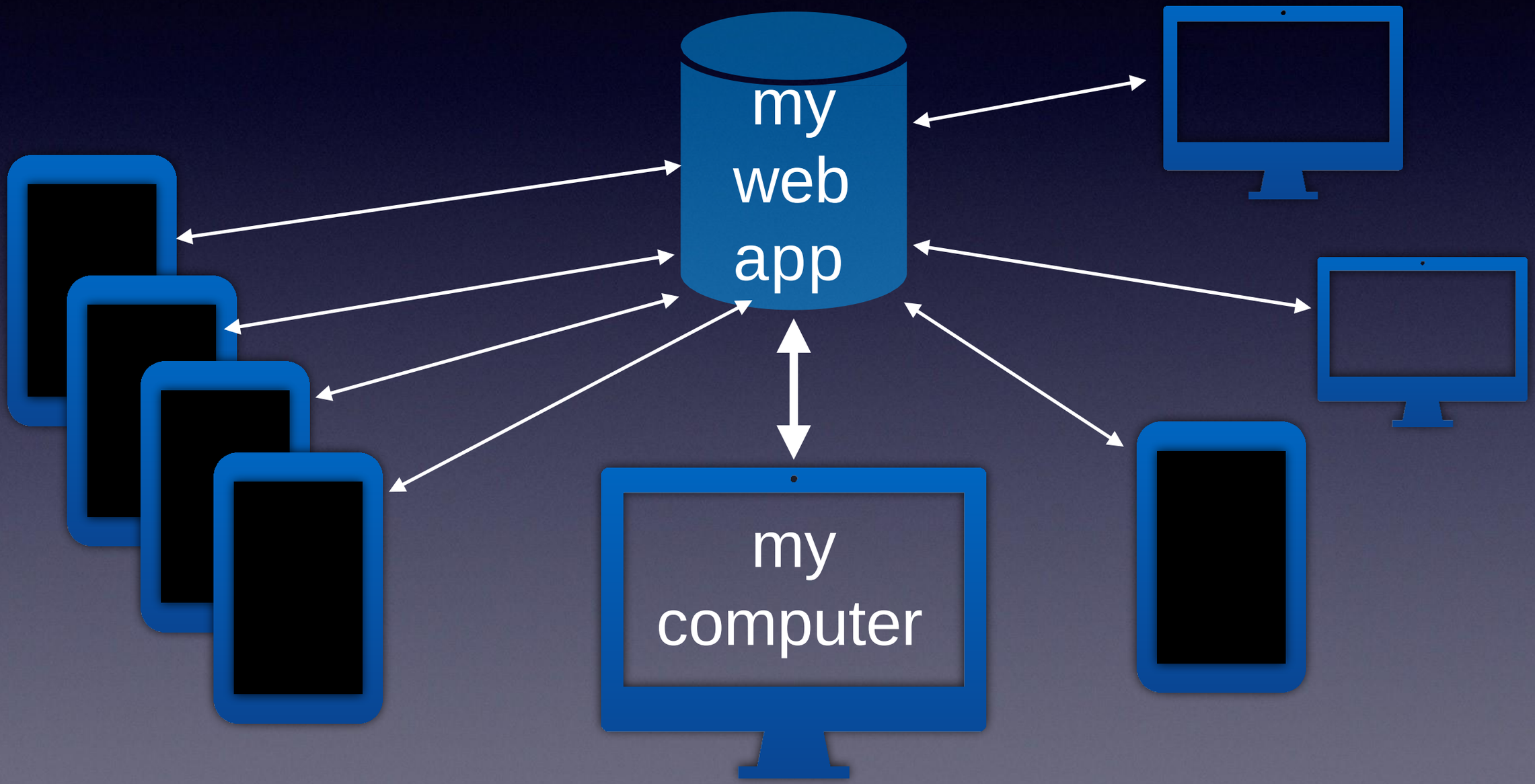
- Later version
- Controller sits between user, view and model.
- View and model never interact directly.

MVC?


Browser
HTTP
Server
API calls
API
SQL
Database
View/Controller
Model

Why use MVC?

Why use MVC?



MVC

Different platforms (think mobiles vs desktop) have:

- different views (big screen, small screen)
- different user controls (keyboard, touchscreen)

Model

Responsible for:

- Managing application data (can interact with database)
- Takes input from the controller
- Represents the application core
- Only “knows” about itself

View

- Displays the data
- Observes information from the model (sometimes via the controller)
- Can be an HTML page, but is anything that displays the data

Controller

- Handles user interaction
- Gets data from user input
- Sends data to the model
- Gets data from the model (and handles errors)

MVC - Dependence Hierarchy

- **Model** only knows about itself
- **View** only knows about the data from the model
- **Controller** knows about the model and the view and can act as an intermediary

Dependence Hierarchy

- The **model** can persist between platforms
- Even if the web app is accessed via a smart phone and a desk top, the **model** doesn’t need to change (but the **view** does)
- The **controller** will gather data from the user interface that can be used to inform the **view**

MVC in practice (React)

- React does not *really* handle the **model**
- User input can update the view via view controllers (and one way binding)
- **Data flows from parent components to children components only (not upwards)**
- But there is no application state: the **model** is not saved by React alone, and the flow of data is only from the user interface to the view

Aside…React Parents

Parent components (like web page is parent of form component).

Two forms won’t update each other.

Key word: Redux (it’s a database for storing the states of ALL the components and letting them interact).

MVC in practice (Angular)

If you set up an Angular (>2.0) project, you'll see a clear example of MVC.

- The "View" code is in .html files
- The "Controller" is the associated TypeScript file
- The "Model" is the application state

Advantages of MVC

- Clear separation between presentation and business logic
- Facilitates parallel development
- Easy to maintain
- Easy to upgrade
- Objects are separate from each other

Disadvantages

- **Some** functionality may be duplicated
- Not worth it for small applications
- All things data goes via the model (can be inefficient for data access)

How do I use MVC?

- Consider what functionality is related to the view and what is related to the model, divide the code among the files appropriately

- Don’t fill your html files up with JavaScript that is unrelated to the “View”

- Consider using a framework (Angular?) that implements MVC by default

Other software design considerations

- Working in a team
- Finding/creating a team that works

“The Joel Test”

- Created by Joel Spolsky (one of the founders of StackOverflow)
- Provides a quick and easy check to see if a software development team is “healthy”
- Highlights some good practice
- https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code/

The Joel Test

More about software team management than coding, but two things are particularly important for us:

- Do you use source control?
- Can you make a build in one step?

Do you use source control?

- The original “Joel’s Test” is old - he talks about CVS, but nowadays we might talk about git.

- It allows individual coders to work together asynchronously

- It also allows mistakes to be rolled back, so you can be fearless

Can you build it in one step?

When you’re debugging, you need to know:

- You’re editing the right file
- Your edits are making it through to the system you run

Well, **can** we build in one step?

Do you know the commands to “build” your server from “nothing”? Are these right?

- git clone <my repository>

- cd <my repository path>

- npm install

- node index.js

Lab

- We’re going to look at a structured back end that actually does implement a user sign up and log in system using Express and Node.
- You will also use curl to test your back end
- You can run the system **locally**. You should get this lab (or an equivalent) working before moving on to its follow up.
- At one point, you might need **4 terminals** open: one to run the server, one to run the database, one to look inside the database (mongo shell) and one to shout curl commands.

Project Pigeon