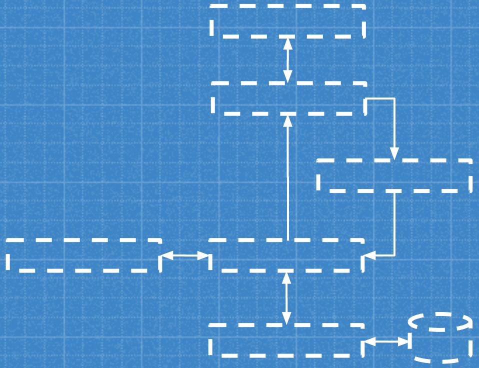


BUILDING WITH MVC

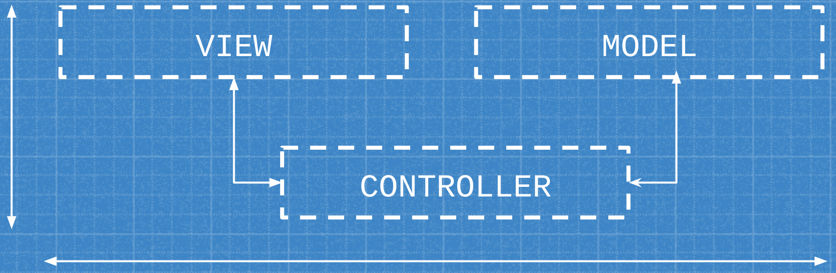
BY EVAN SHAW

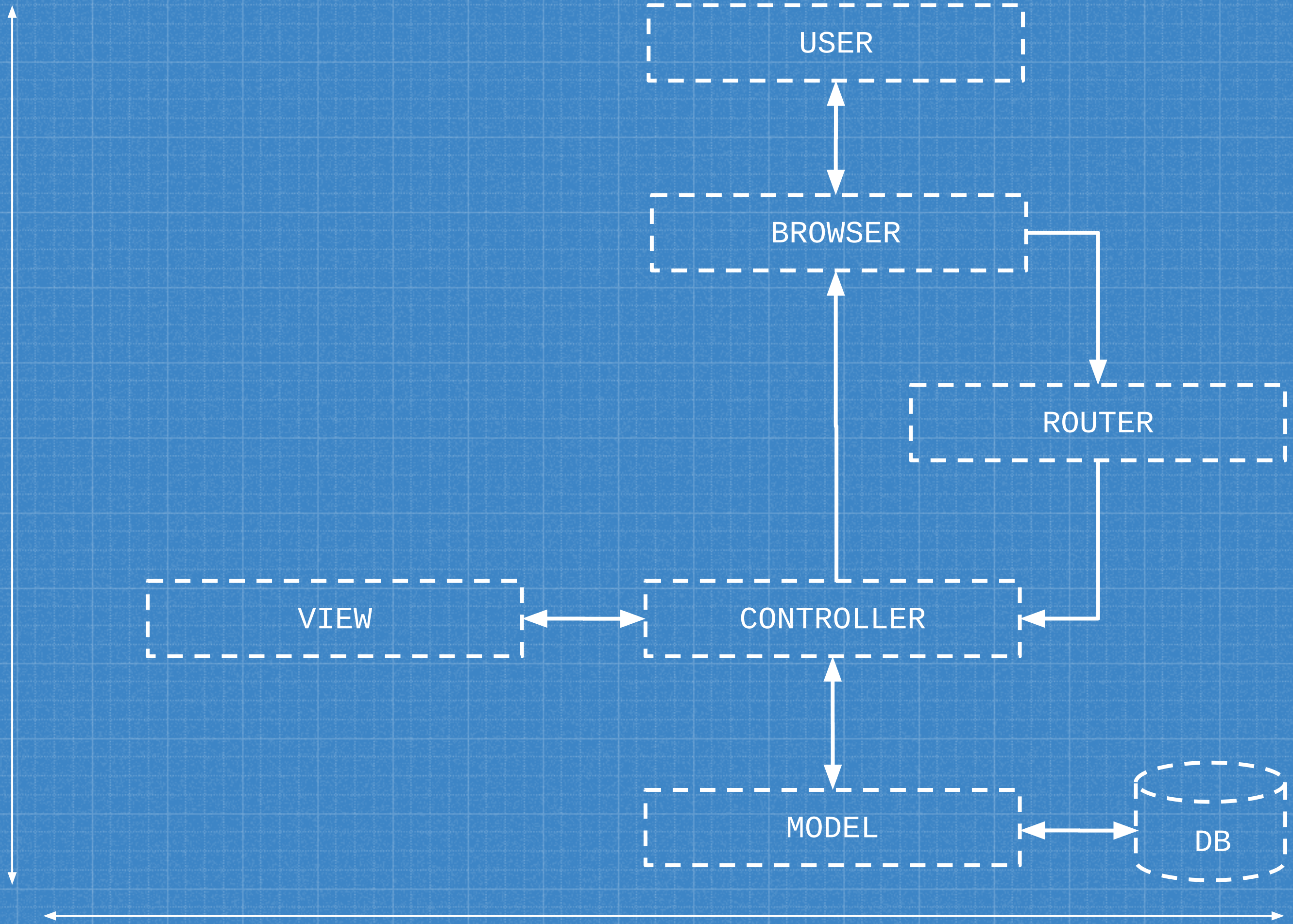
WHAT IS MVC?

AN ARCHITECTURAL PATTERN

That separates an application into three main logical components:

1. **Model**
   a. Handles the data logic
      i. Connected to the DB
      ii. Thinks ill of the View.
2. **View**
   a. Manages the display.
      i. Generates the UI
      ii. Only HTML/CSS
      iii. Sworn enemy to Model.
3. **Controller**
   a. The middleman.
      i. An intermediary that collects data from the **Model**, processes it, and tells the **View** what to do with it.
      ii. Multiple Controllers all centralized in server.js.




USER
BROWSER
ROUTER
VIEW
CONTROLLER
MODEL
DB

1

MODEL

Understanding the Model and it’s friend, the database.

THE MODEL

- The Model is the only component of MVC that talks to the DB. Sending and retrieving data as requested by the Controller. Working with both SQL based DBs and NoSQL DBs.
- Database interfaces such as Mongoose can be considered part of the model since their primary function is talking to the DB.

Mongoose:

- ➔ An ODM (Object Data Modeling)library for MongoDB.
- ➔ A schema based solution to model data.
    - ◆ The Schema creates key value pairs.
- ➔ Creates an easy to use object reference.
- ➔ Models the DB within the code.
  - ◆ Ensuring data is both consistent and reliable.

MONGOOSE SCHEMA

Using Mongoose.

- Install it.
- Require It.
- Set a variable reference to a schema.
- Create the schema.
- In order to use in other files, create an instance of the schema. We call this a ‘model’. Require this in other files.
- The model includes upto three parameters:
    - The model name
    - The schema name
    - The collection name

The Mongoose Schema:

```
const mongoose = require('mongoose');
const todoTaskSchema = new mongoose.Schema({
title: {
    type: String,
    required: true
},
content: {
    type: String,
    required: true
},
date: {
    type: Date,
    default: Date.now
}
})

module.exports =
mongoose.model('TodoTask',todoTaskSchema,'tasks')
```

2

VIEW

Seeing the View and its engines.

THE VIEW

- The View receives requests from the Controller to layout and structure the HTML via a template engine. Once a template has been completed, it’s passed back to the Controller in order to be rendered as HTML for the user.

Template languages:

- Some examples of common template languages include but are not limited to:
  - EJS
  - Handlebars
  - Pug
  - Nunjucks
  - React.
- The template language needs to be installed and required.
  - `app.set("view engine", "ejs");`
- Every template language will have its own unique syntax.

3

CONTROLLER

Understanding the Middleman

THE CONTROLLER

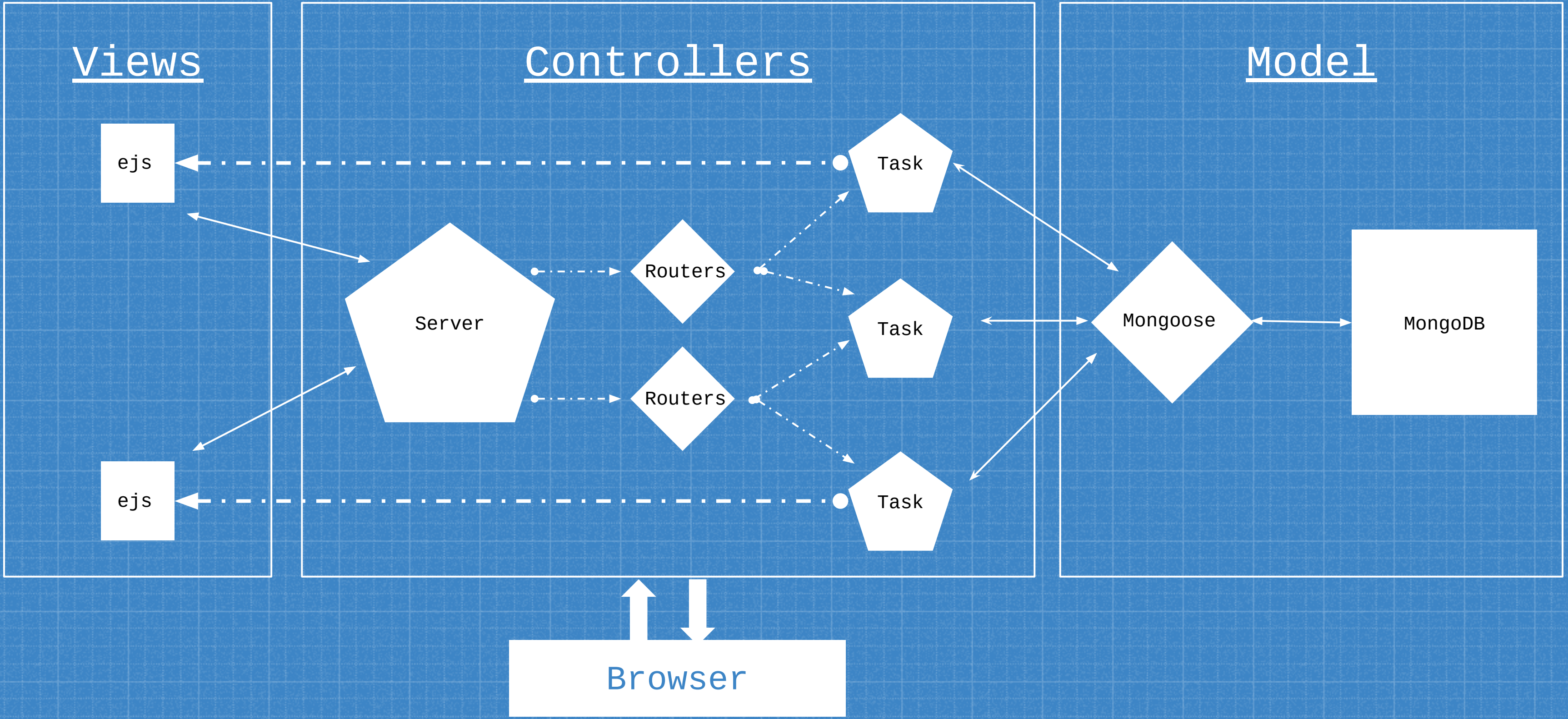
- The Controller sits between the View and Model and uses JavaScript to dictate how an application behaves while taking requests from and responding to the browser.
- While there may be multiple controllers for every application, the Controller is centralized in server.js.

THE ROUTER

- ➔ When a request is received, the controller reads the URL sent with that request and passes it to the correct router which handles requests with the appropriate URL.
- ➔ Every time the Controller receives a request, it looks at the URL to determine what router correctly manages that route.
- ➔ Paths may be routed multiple times.

Using MVC Structure

- A simple application with two views, a server, and multiple controllers. Using Mongoose as the UI for MongoDB.



MVC in action - Liking an image.

1. You Click a button which increases the likes on your Moms post.
   - ➔ This action creates a request with data built into it.

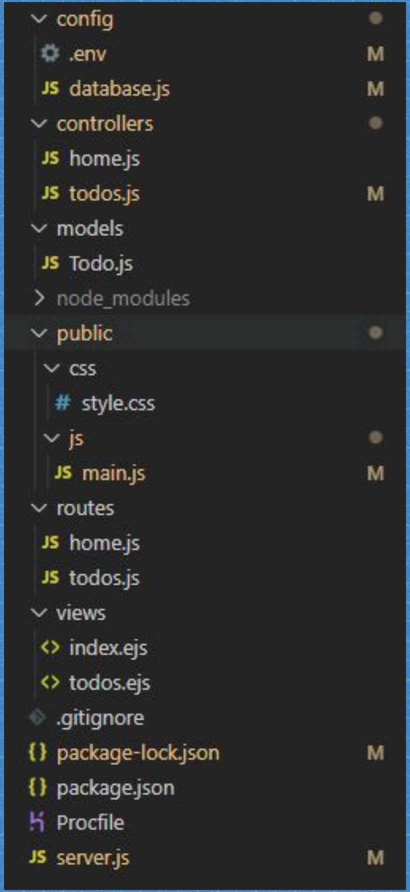
2. The Router hears this request, it looks at the information sent with the request and ‘routes’ it to the Model based on the URL which was sent with the click of the button.

3. The Model then receives this request to update the DB with the new like count and responds back to the Controller with the updated information.

4. Once the Controller receives the response from the Model, the Controller then responds back to the client and ‘instructs’ it to refresh the page. *and passes the response data to the View.

5. The Client gets a successful response from the Controller and refreshes the page. This creates a new request that is routed to the View.

6. The View then renders the new data and passes it back to the Controller who pushes and renders the template for the Browser where the new like is displayed for the User.



```
config
    .env                M
    database.js         M
controllers
    home.js
    todos.js            M
models
    Todo.js
node_modules
public
    css
        style.css
    js
        main.js         M
routes
    home.js
    todos.js
views
    index.ejs
    todos.ejs
.gitignore
package-lock.json       M
package.json
Procfile
server.js               M
```

MVC Code Structure

- MVC Components are partitioned into their own individual folders
    - This allows the server to become more organized with multiple controllers outside the server.js file.
- Organized routes allow data to be easily traced by following the required branches.
- Mongoose schemas provide a reference to the data structure allowing team collaboration without DB access.

WHY USE MVC?

ADVANTAGES:

- Separation of concerns.
- Team collaboration.
- Modular and reusability.
- Ease of testing.
- Organization
- Targeted testing.
- Fast troubleshooting.

DISADVANTAGES:

- High complexity.
- MVC must have strict rules over methods.
  - An appropriate reaction from the controller.
- Views may fall behind updating while rendering when overburdened with multiple requests

Thanks!

ANY QUESTIONS?

You can find me at:

@evan_shawAZ
evanshaw.az@gmail.com