WHAT IS MVC ARCHITECTURE?

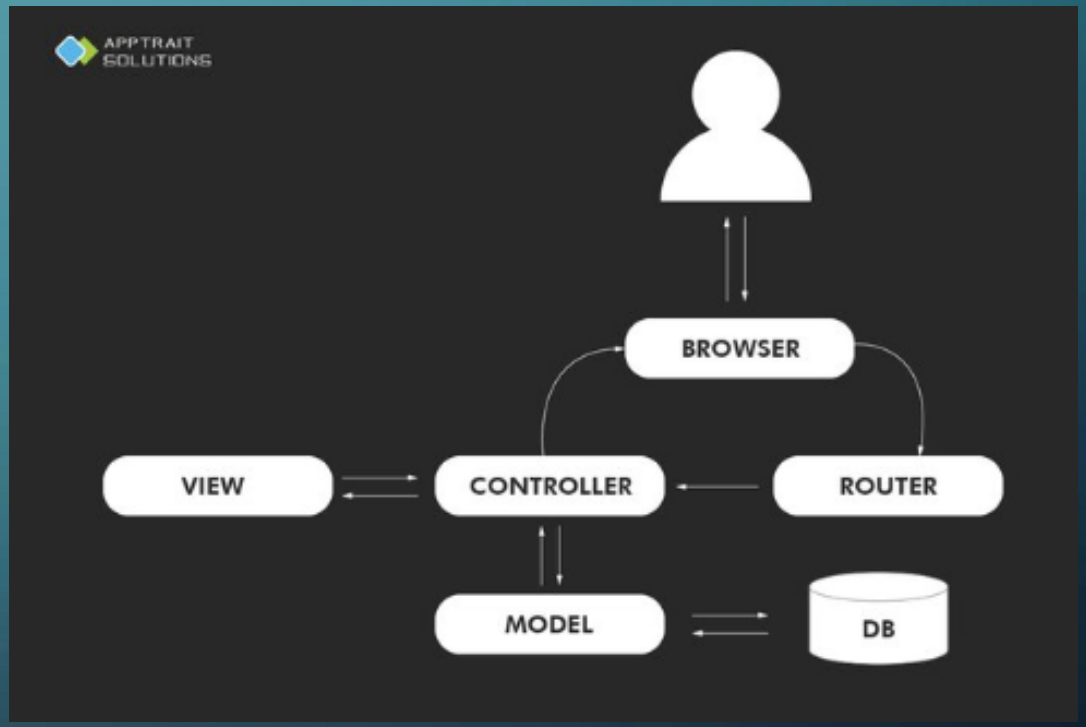
BY MARK (PING_PONG17)

MVC: MODEL, VIEW, CONTROLLER

- MVC is a way to ORGANIZE your backend code.
- Similar to the 'Golden Rule' of the Separation of Concerns with our front-end projects.
- This organization lets us:
  - Have multiple developers work at the same time on different parts of the project.
  - Creates an easily changeable application.
  - Makes the code more predictable and easier to work with.

HOW DOES IT WORK?

- Separate your code into three main categories.
- Each category is built to handle a specific aspect of an application.
- Each category can be independently changed without modifying the others.



THE CATEGORIES

Model

- Concerned with the data and database of an application.
- Can add or retrieve data from the DB.
- Speaks only with the Controller, never interacts with the View.

View

- Concerned with the presentation of data and User Interface.
- Defines how the user sees and interacts with an application.
- Only speaks with the Controller/Router never the Model.

Controller

- Enables the interconnection between View and Model.
- Determines actions to take when receiving input from User.

A CLOSER LOOK AT THE MODEL

- Models are implemented with database systems.
  - MongoDB, MySQL, PostgresSQL, etc.
- Models define what data is available for the View to render.
- It can respond only to the Controller's requests to add data to the DB or deliver data from the DB.
- Models can incorporate Object Data Models to give them consistent structure and validation.

MODELS, MONGOOSE, AND MONGODB.

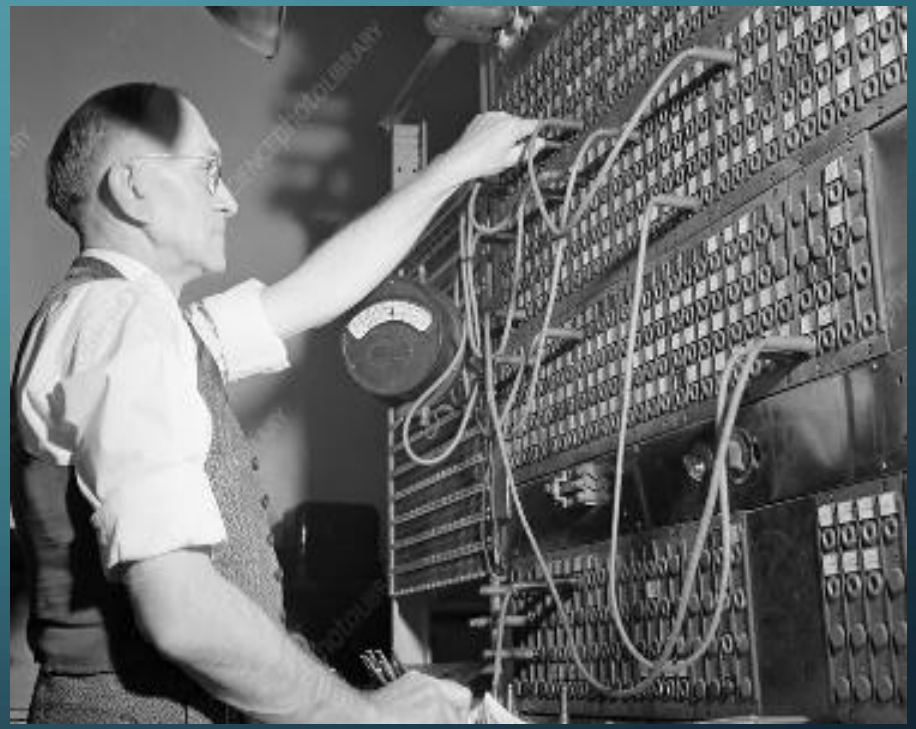
- Mongoose is a tool that works with MongoDB that adds structure to your database.
- That structure is called “schemas”. A schema is a blueprint to which all your data must conform.
- Schemas are then compiled into ‘models’ that use the blueprint to construct new instances of documents.
- Every object passed into the database will have this same structure, making your data consistent and predictable.

A CLOSER LOOK AT THE VIEW

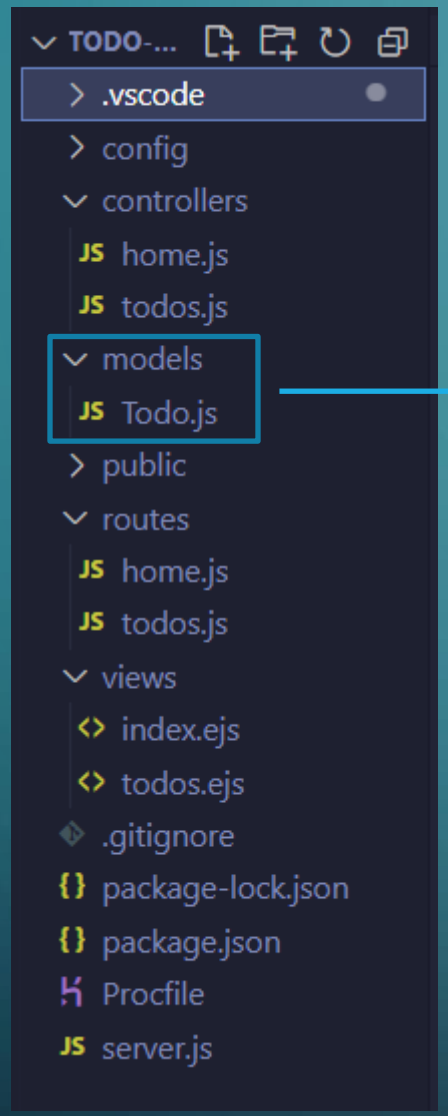
- The view renders the actual look of the application. The part that the user will see and interact with.
- Views can use templating engines to dynamically render HTML and CSS. Examples include EJS, Handlebars, Pug, Nunjucks, etc.
- The View takes data sent from the Controller to use in the templating engine.
- The View only responds and sends request through the Controller.

A CLOSER LOOK AT THE CONTROLLER

- The Controller is like the switchboard operator of the application.
- It takes in an input from the client interacting with the View and decides on an action.
- Uses Request types (GET, CREATE, PUT, DELETE) and routes (‘/’) to determine what to ask for and respond with.



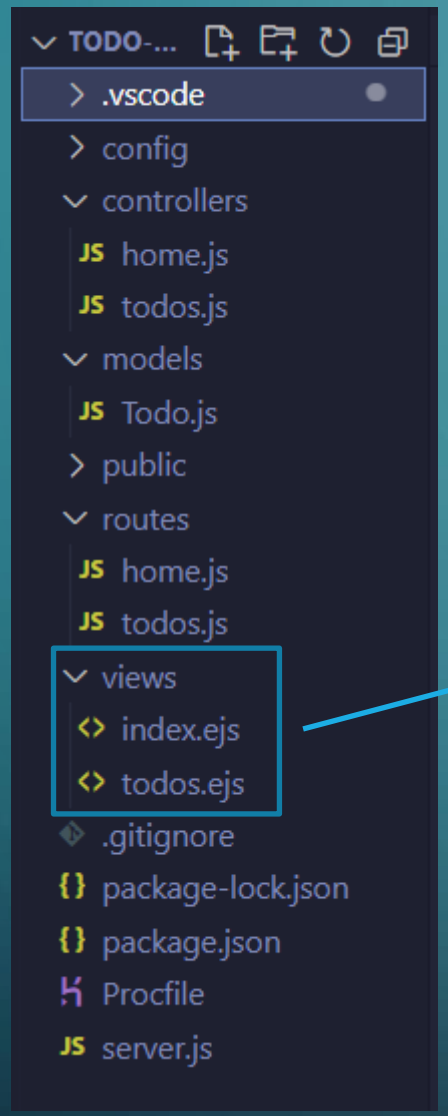
FILE STRUCTURE



- Model – includes anything to do with the interaction with our Database, including our Mongoose schema.

```
const mongoose = require('mongoose')

const TodoSchema = new mongoose.Schema({
  todo: {
    type: String,
    required: true,
  },
  completed: {
    type: Boolean,
    required: true,
  }
})

module.exports = mongoose.model('Todo', TodoSchema)
```

FILE STRUCTURE

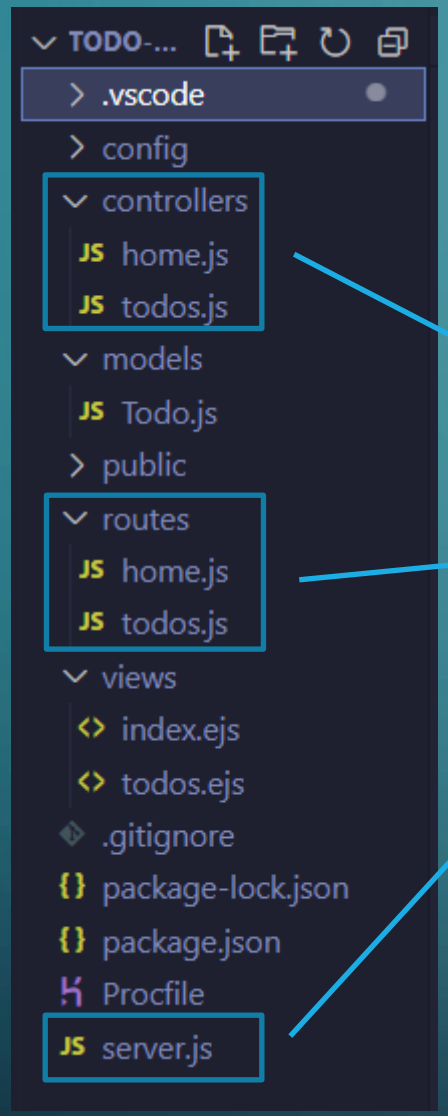


- Views – include anything to do with the presentation and user interaction with the application.

views > index.ejs > html

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width,
       initial-scale=1.0">
7      <title>Document</title>
8  </head>
9  <body>
10     <h1>Create A Todo List By Clicking The Button Below</h1>
11     <a href="/todos"> New Todo List</a>
12 </body>
13 </html>
```

FILE STRUCTURE



- Controller – includes anything to do with the flow of data across the application.

settings.json M | server.js

server.js > ...

```
1  const express = require('express')
2  const app = express()
3  const connectDB = require('./config/database')
4  const homeRoutes = require('./routes/home')
5  const todoRoutes = require('./routes/todos')
6
7  require('dotenv').config({path: './config/.env'})
8
9  connectDB()
10
11 app.set('view engine', 'ejs')
12 app.use(express.static('public'))
13 app.use(express.urlencoded({ extended: true }))
14 app.use(express.json())
15
16 app.use('/', homeRoutes)
17 app.use('/todos', todoRoutes)
18
19 app.listen(process.env.PORT, ()=>{
20     console.log('Server is running, you better catch it!')
21 })
```

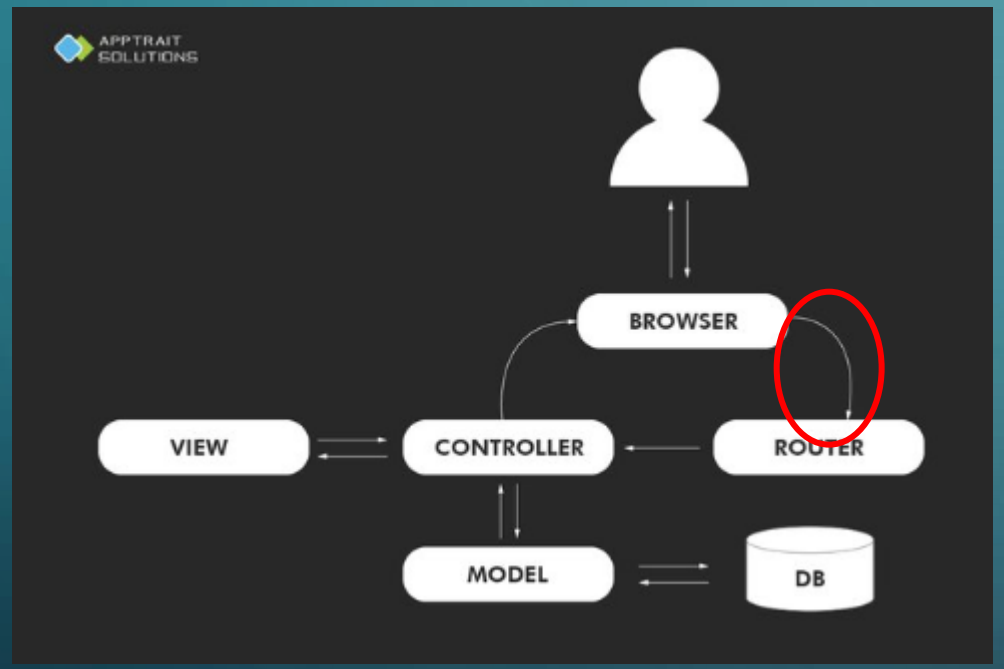
controllers > home.js > <unknown>

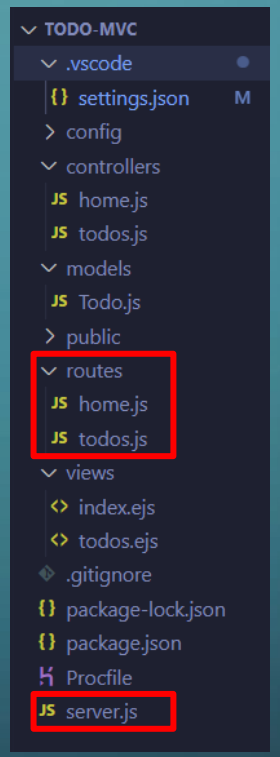
```
1  module.exports = {
2      getIndex: (req,res)=>{
3          res.render('index.ejs')
4      }
5  }
```

routes > home.js > ...

```
1  const express = require('express')
2  const router = express.Router()
3  const homeController = require('../controllers/home')
4
5  router.get('/', homeController.getIndex)
6
7  module.exports = router
```

WALK THROUGH MVC USING OUR TODO APP



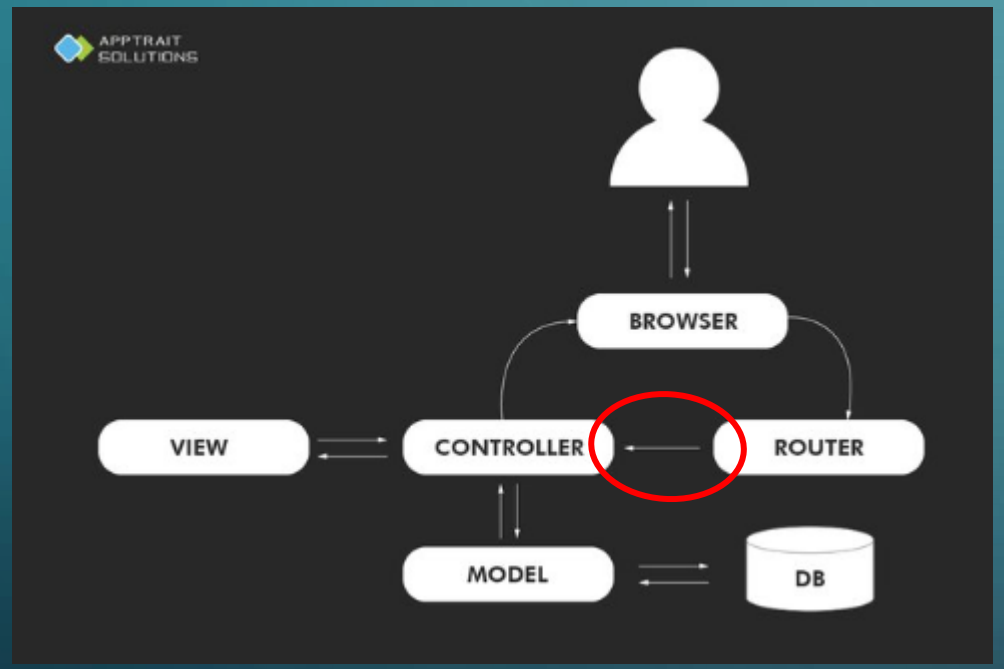


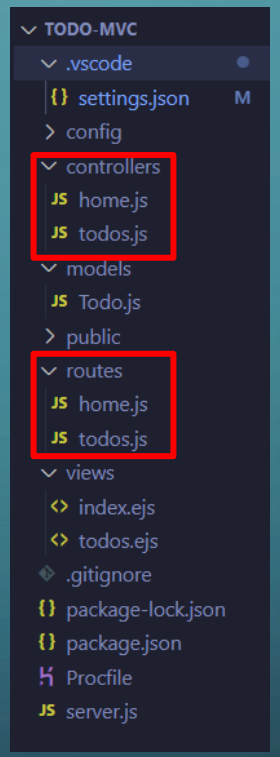
Client makes a request to add a todo item. This request is sent to the server.js which sends it to the router.

```
app.use('/todos', todoRoutes)
```

```
router.post('/createTodo', todosController.createTodo)
```

WALK THROUGH MVC USING OUR TODO APP



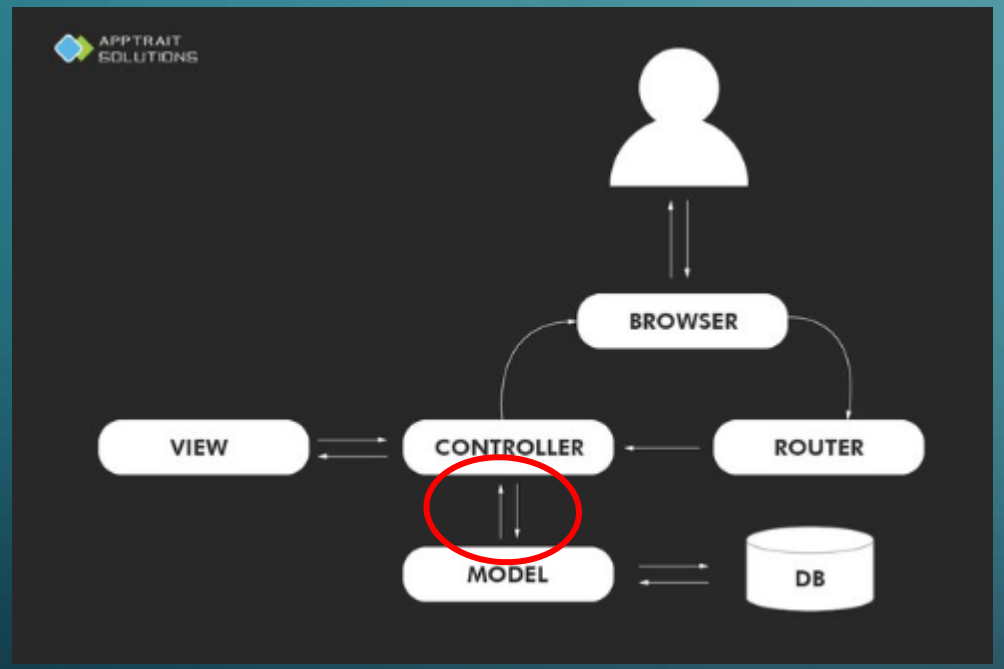


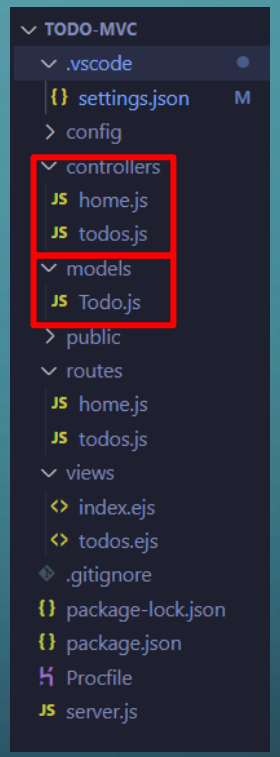
In the todos router the request is sent to the todos Controller responsible for creating a new todo.

```
router.post('/createTodo', todosController.createTodo)
```

```
createTodo: async (req, res)=>{
    try{
        await Todo.create({todo: req.body.todoItem, completed: false})
        console.log('Todo has been added!')
        res.redirect('/todos')
    }catch(err){
        console.log(err)
    }
},
```

WALK THROUGH MVC USING OUR TODO APP

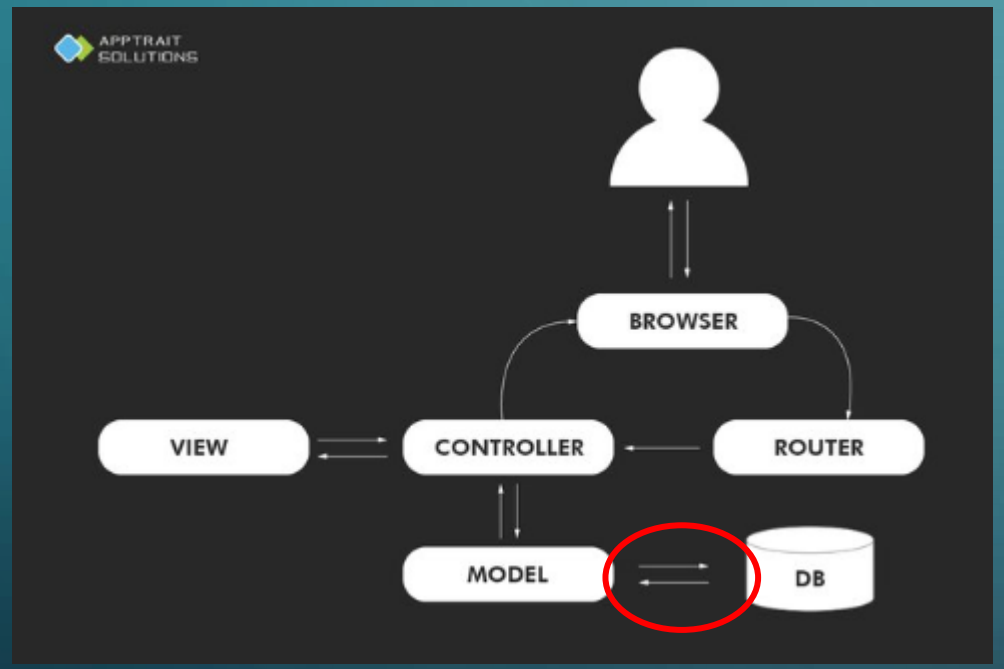


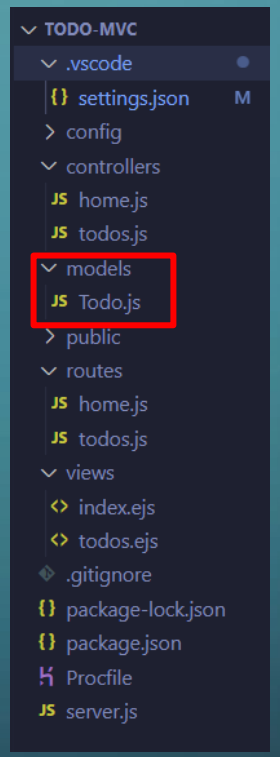


The Controller awaits a request to the model, passing the data in the schema's defined structure.

```
createTodo: async (req, res)=>{
    try{
        await Todo.create({todo: req.body.todoItem, completed: false})
        console.log('Todo has been added!')
        res.redirect('/todos')
    }catch(err){
        console.log(err)
    }
},
```

WALK THROUGH MVC USING OUR TODO APP

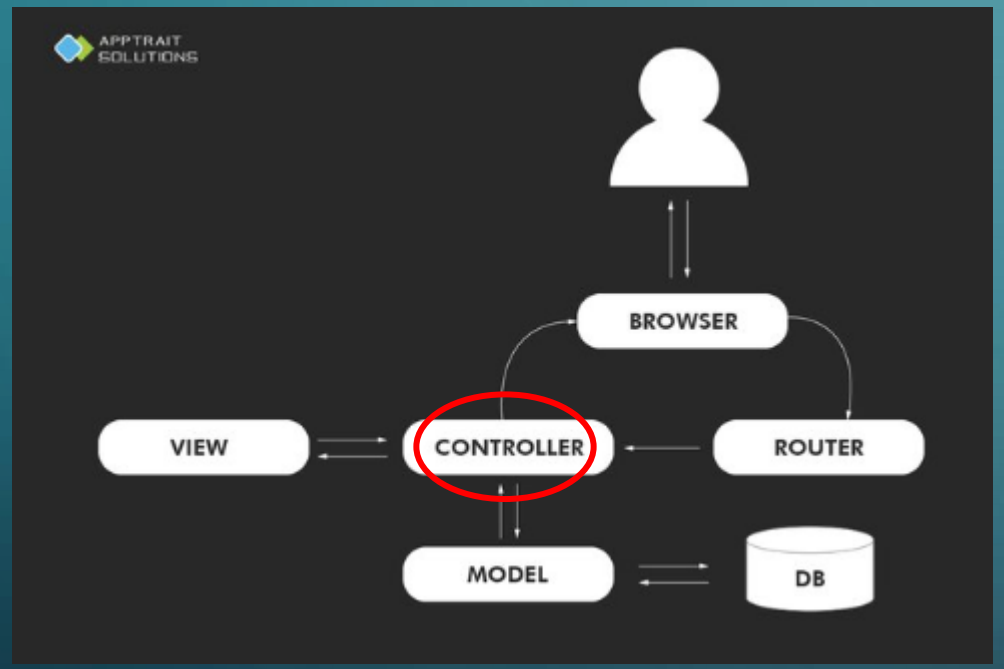


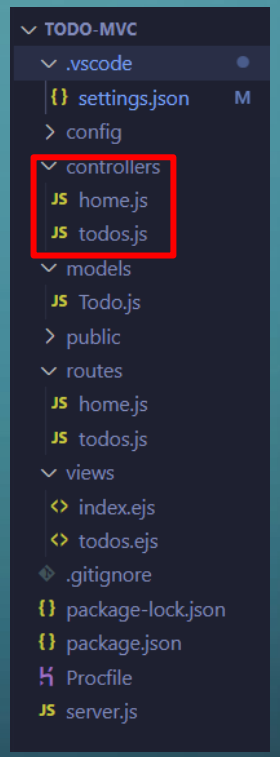


The Model creates the new todo document using the mongoose schema and adds it to the DB.

```
const mongoose = require('mongoose')

const TodoSchema = new mongoose.Schema({
  todo: {
    type: String,
    required: true,
  },
  completed: {
    type: Boolean,
    required: true,
  }
})

module.exports = mongoose.model('Todo', TodoSchema)
```

WALK THROUGH MVC USING OUR TODO APP

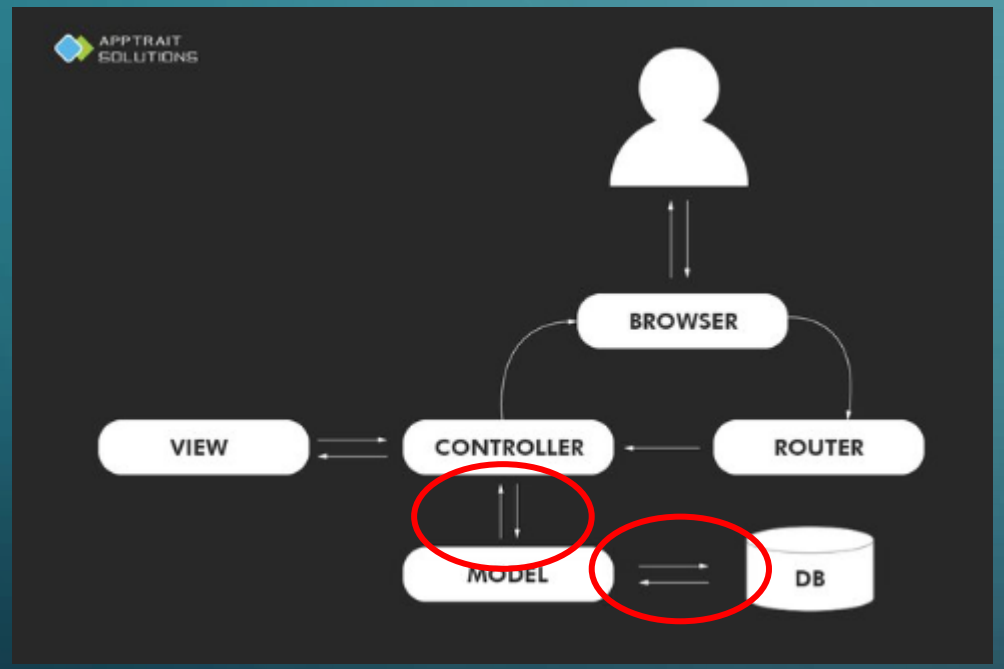


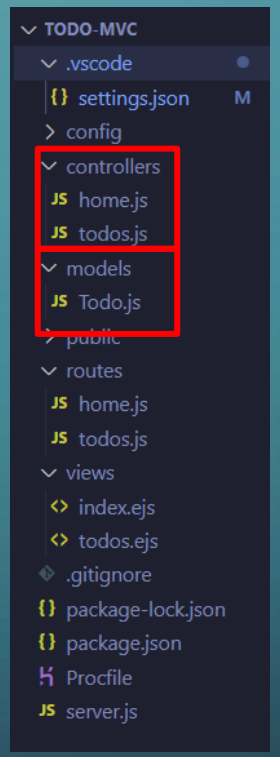


The Controller responds with a redirect and refreshes the /todos page. Creating a Get request on the /todos route.

```
module.exports = {
    getTodos: async (req,res)=>{
        try{
            const todoItems = await Todo.find()
            const itemsLeft = await Todo.countDocuments({completed: false})
            res.render('todos.ejs', {todos: todoItems, left: itemsLeft})
        }catch(err){
            console.log(err)
        }
    },
```

WALK THROUGH MVC USING OUR TODO APP

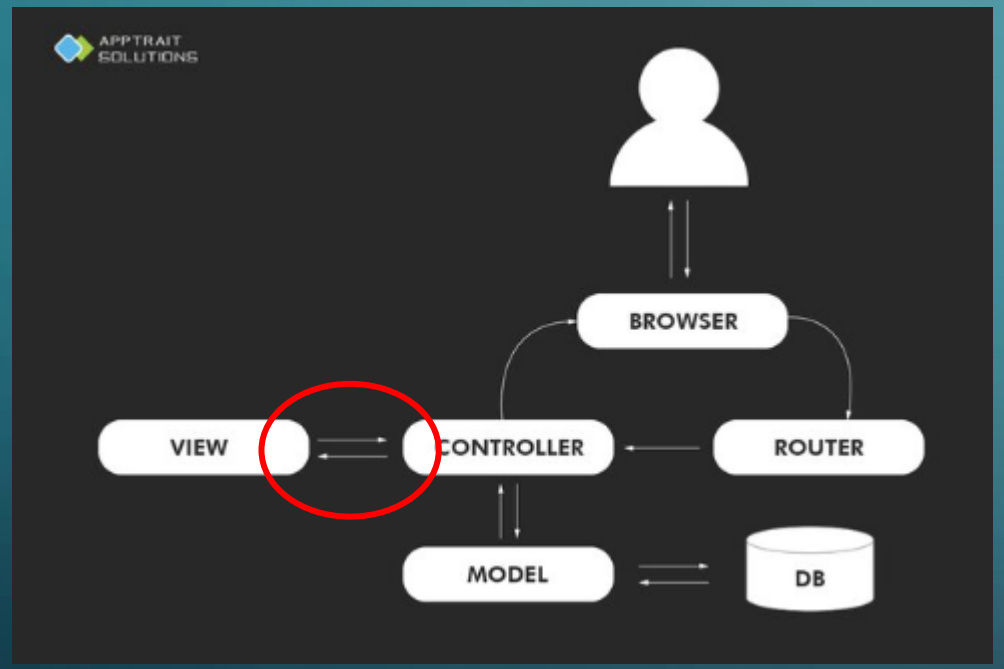


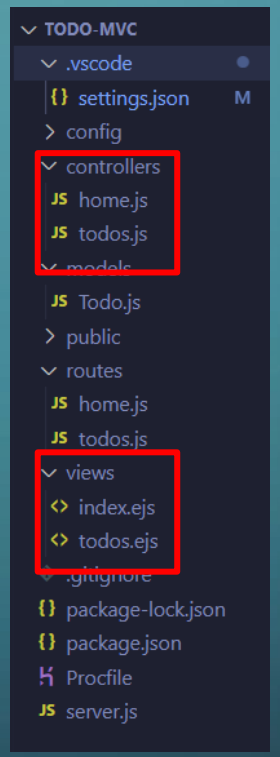


The Controller awaits data from the Model, finding all the todos as well as counting the number of them.

```
module.exports = {
    getTodos: async (req,res)=>{
        try{
            const todoItems = await Todo.find()
            const itemsLeft = await Todo.countDocuments({completed: false})
            res.render('todos.ejs', {todos: todoItems, left: itemsLeft})
        }catch(err){
            console.log(err)
        }
    },
```

WALK THROUGH MVC USING OUR TODO APP

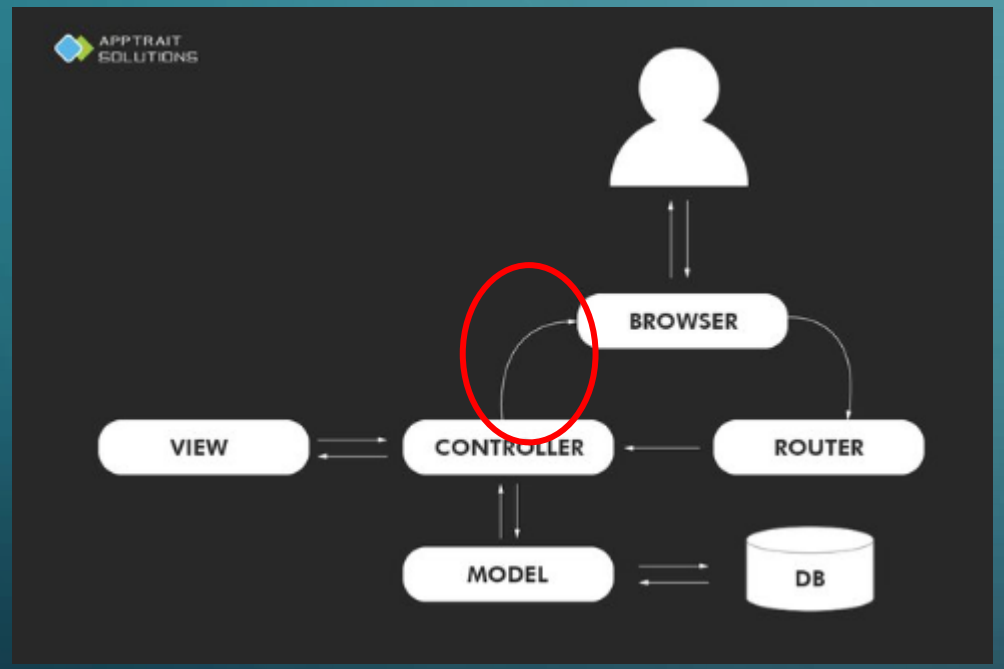


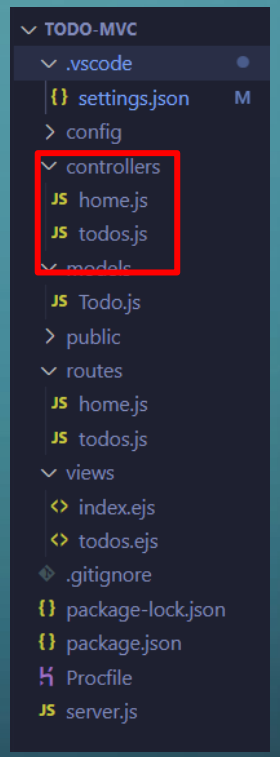


The Controller then asks the View to render an HTML page using the data from the Model and EJS, our templating engine.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <h1>Todos</h1>
    <ul>
    <% todos.forEach( el => { %>
        <li class='todoItem' data-id='<%=el._id%>'>
            <span class='<%= el.completed === true ? 'completed' : 'not'%>'><%= el.todo %></span>
            <span class='del'> Delete </span>
        </li>
    <% }) %>
    </ul>

    <h2>Things left to do: <%= left %></h2>

    <form action="/todos/createTodo" method='POST'>
        <input type="text" placeholder="Enter Todo Item" name='todoItem'>
        <input type="submit">
    </form>

    <script src="js/main.js"></script>
</body>
</html>
```

WALK THROUGH MVC USING OUR TODO APP





The render is then sent to the browser!

```
res.render('todos.ejs', {todos: todoItems, left: itemsLeft})
```

Thank you for attending my TED Talk.

-Ping_Pong17