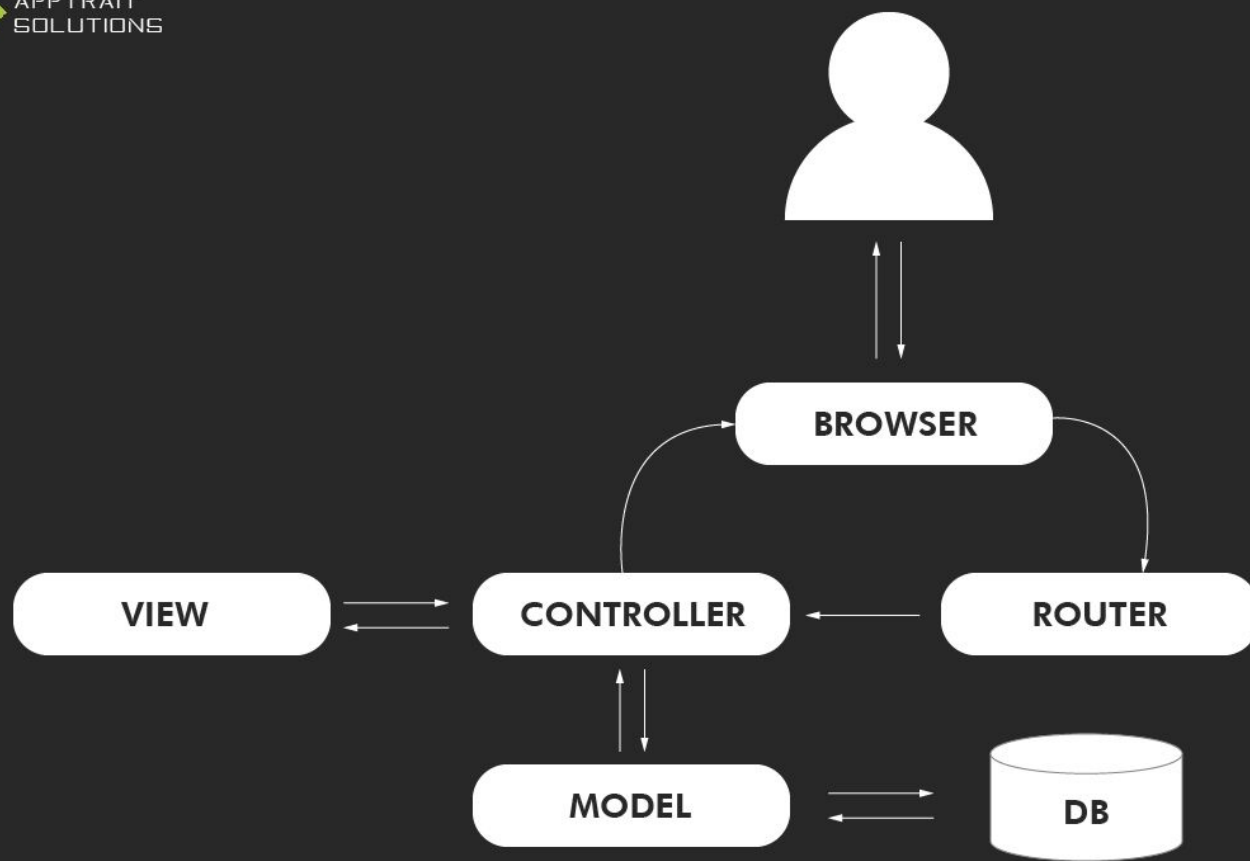
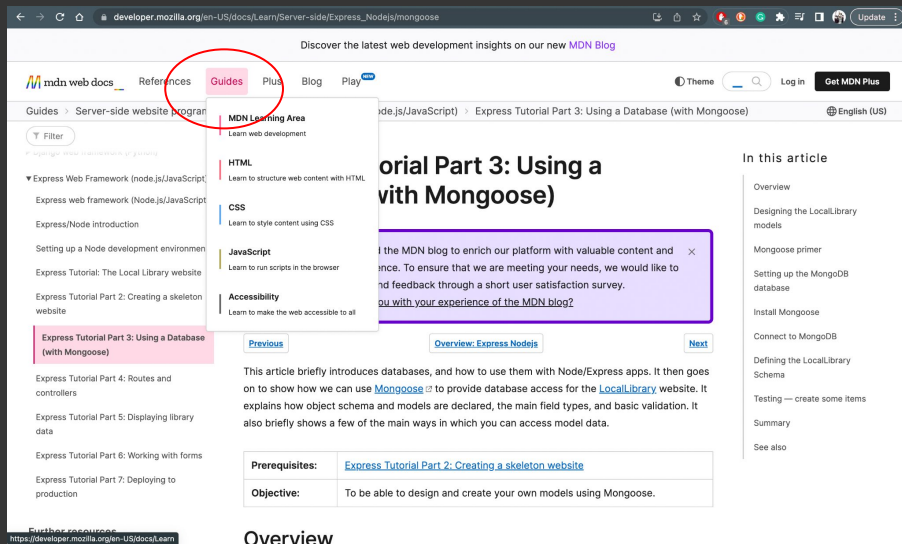

Model, View, Controller (MVC)

Taine Jarvis



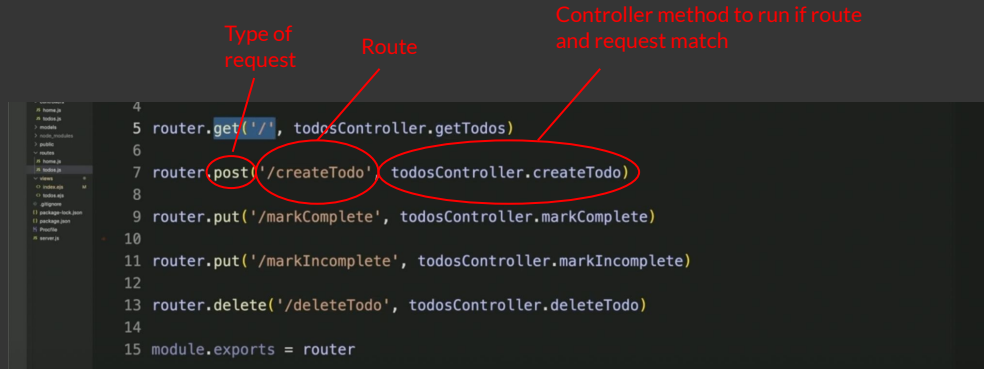
Browser

- The user will make a request to the server through a link or route (' / ', ' /login')
- Clicking on the guide link highlighted below will make a request
- The route for the request is in the url (' /mongoose')



Router

- The router looks at the route and the type of request that has been made
- The requests include post, get, put, delete
- The type of request and route dictate what controller and method will be used
- The request is then passed to the matching controller method



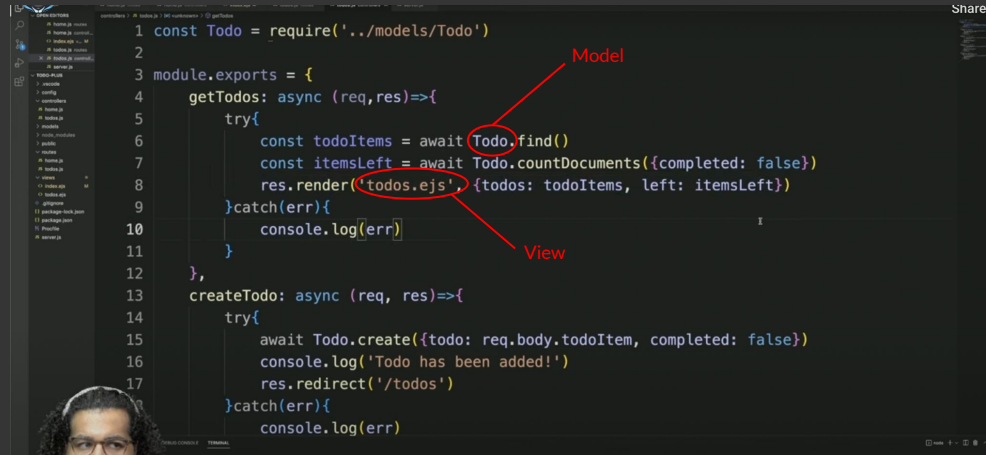
The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The code is in JavaScript and defines an Express.js router. Three red annotations with arrows point to specific parts of the code:

- Type of request**: Points to the `post` method in `router.post`.
- Route**: Points to the `"/createTodo"` string in `router.post`.
- Controller method to run if route and request match**: Points to the `todosController.createTodo` function in `router.post`.

```
4
5 router.get('/', todosController.getTodos)
6
7 router.post('/createTodo', todosController.createTodo)
8
9 router.put('/markComplete', todosController.markComplete)
10
11 router.put('/markIncomplete', todosController.markIncomplete)
12
13 router.delete('/deleteTodo', todosController.deleteTodo)
14
15 module.exports = router
```

Controller

- The controller handles all the requests sent from the router
- If a display is needed the controller communicates with the view
- If data is needed for a operation the controller will communicate with the model

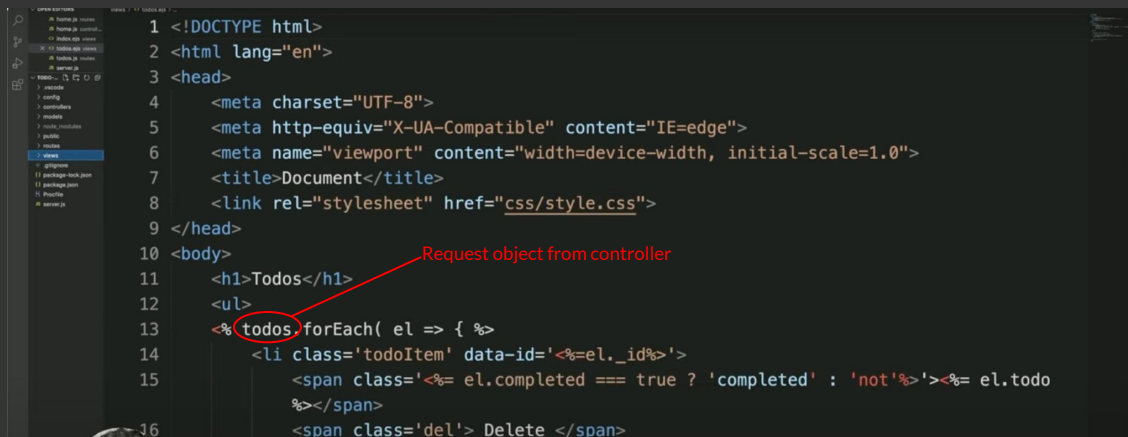


```
1 const Todo = require('../models/Todo')
2
3 module.exports = {
4   getTodos: async (req, res) => {
5     try {
6       const todoItems = await Todo.find()
7       const itemsLeft = await Todo.countDocuments({completed: false})
8       res.render('todos.ejs', {todos: todoItems, left: itemsLeft})
9     } catch (err) {
10      console.log(err)
11    }
12  },
13   createTodo: async (req, res) => {
14     try {
15       await Todo.create({todo: req.body.todoItem, completed: false})
16       console.log('Todo has been added!')
17       res.redirect('/todos')
18     } catch (err) {
19       console.log(err)
20     }
21   }
22 }
```

The screenshot shows a code editor with a file explorer on the left. The code is a Node.js controller file. Two annotations are present: a red arrow labeled "Model" points to the `Todo` object in line 1, and another red arrow labeled "View" points to the `res.render` call in line 8. The `res.render` call is also circled in red.

View

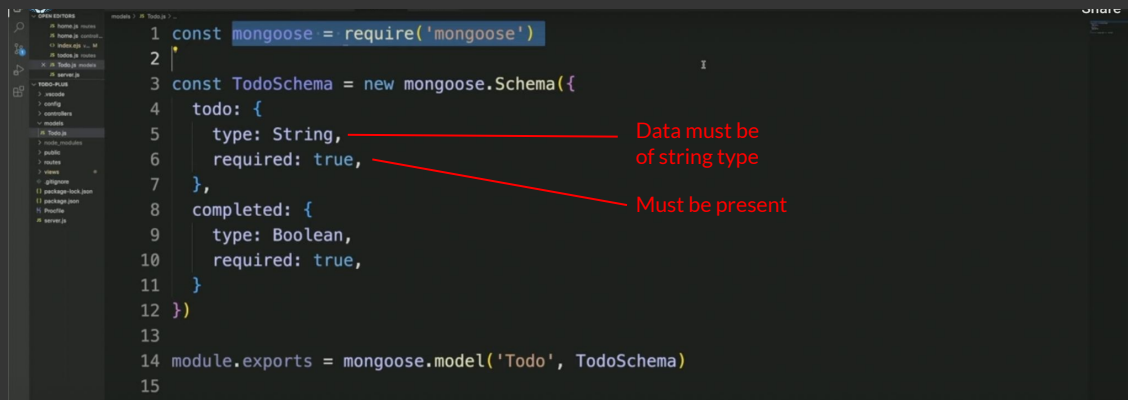
- Contains our HTML, CSS, or anything associated with the display
- Receives requests from the controller and renders out our display. In the example code Embedded Javascript is used.



```
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   <link rel="stylesheet" href="css/style.css">
9 </head>
10 <body>
11   <h1>Todos</h1>
12   <ul>
13     <% todos.forEach( el => { %>
14       <li class='todoItem' data-id='<%=el._id%>'>
15         <span class='<%= el.completed === true ? 'completed' : 'not'%>'><%= el.todo
16         <span class='del'> Delete </span>
```

Model

- The model is responsible for communicating with the database
- The database used is MongoDB and we are utilising mongoose for object modeling
- Mongoose provides a way to create schemas which create a pre-set layout for our data entries which promotes consistency



```
1 const mongoose = require('mongoose')
2
3 const TodoSchema = new mongoose.Schema({
4   todo: {
5     type: String,
6     required: true,
7   },
8   completed: {
9     type: Boolean,
10    required: true,
11  }
12 })
13
14 module.exports = mongoose.model('Todo', TodoSchema)
15
```

Data must be of string type

Must be present

Why?

- Separation of concerns. MVC separates the functionality of our applications
- Makes code easier to maintain
- Allows multiple developers to work on the same project with ease
- Promotes organised programming

