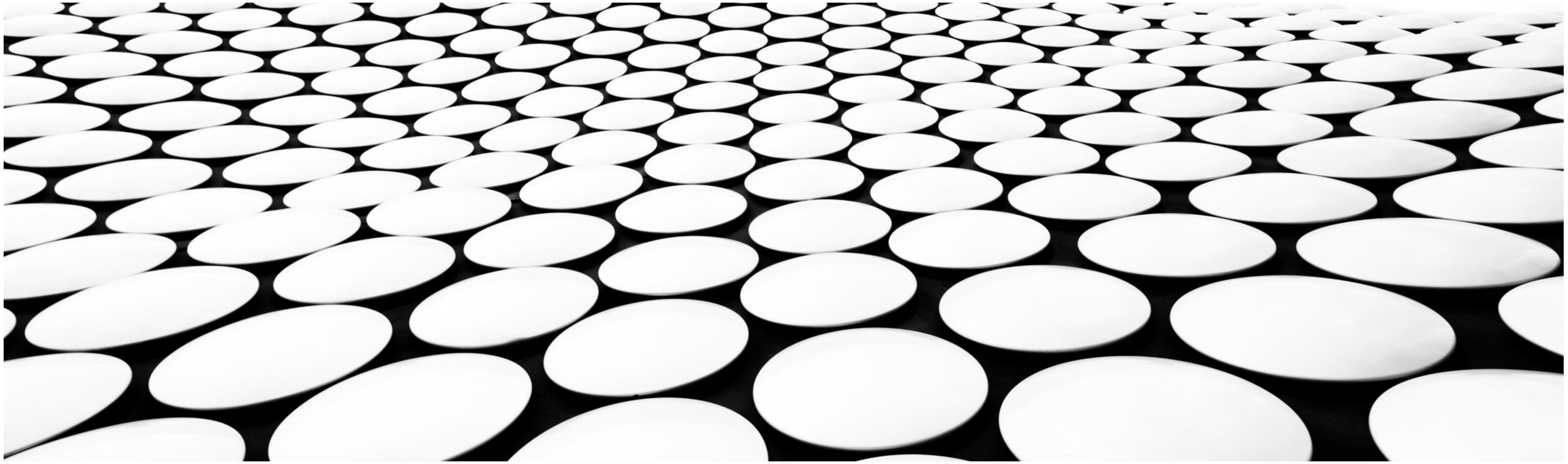
MVC : MODEL, VIEW, CONTROLLER

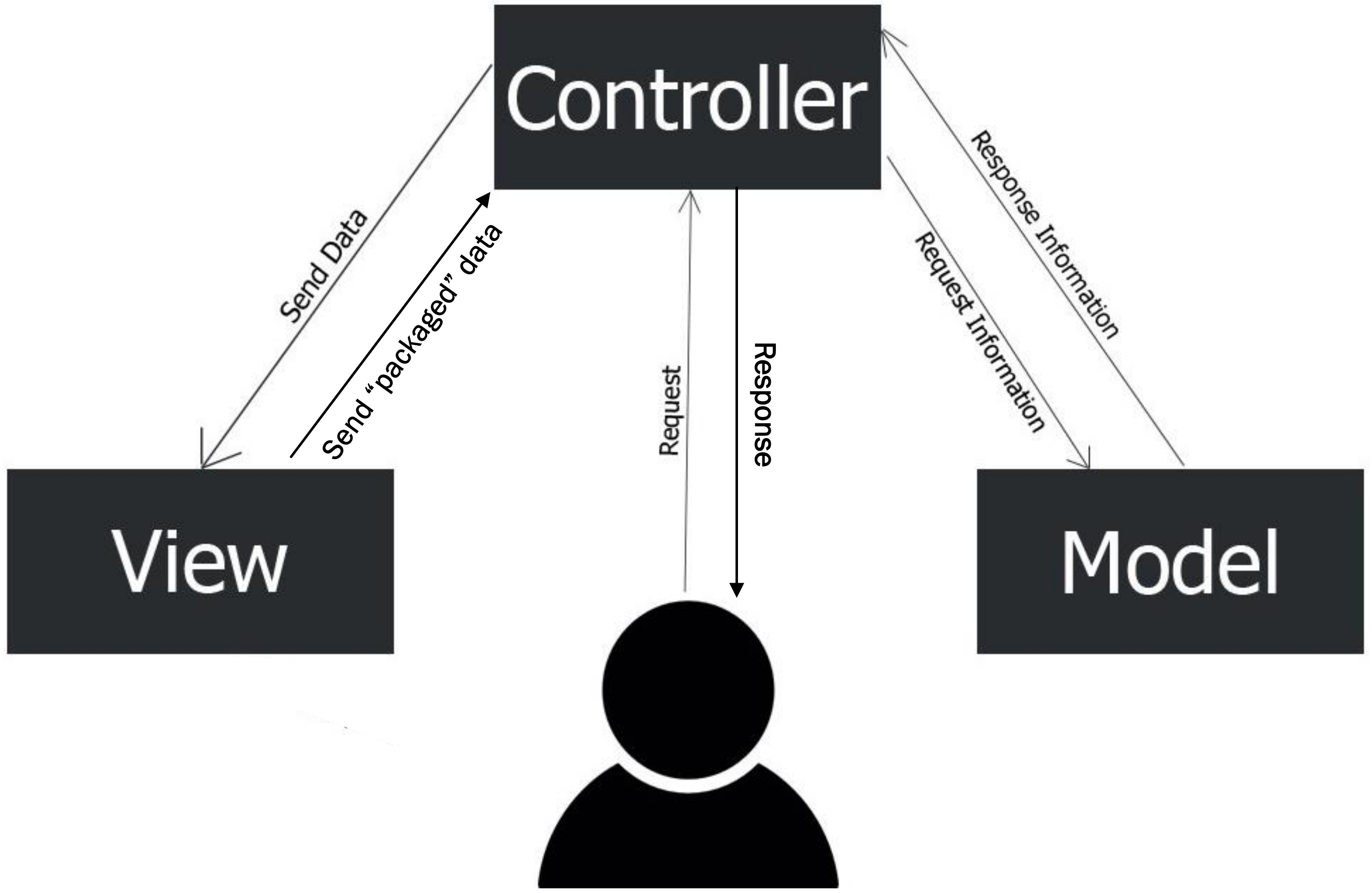
UNDERSTANDING THE COMPONENTS THAT MAKE UP A FULL STACK APPLICATION



WHAT'S MVC?

A pattern to organize components of an application into distinct sections based on their functionality/purpose

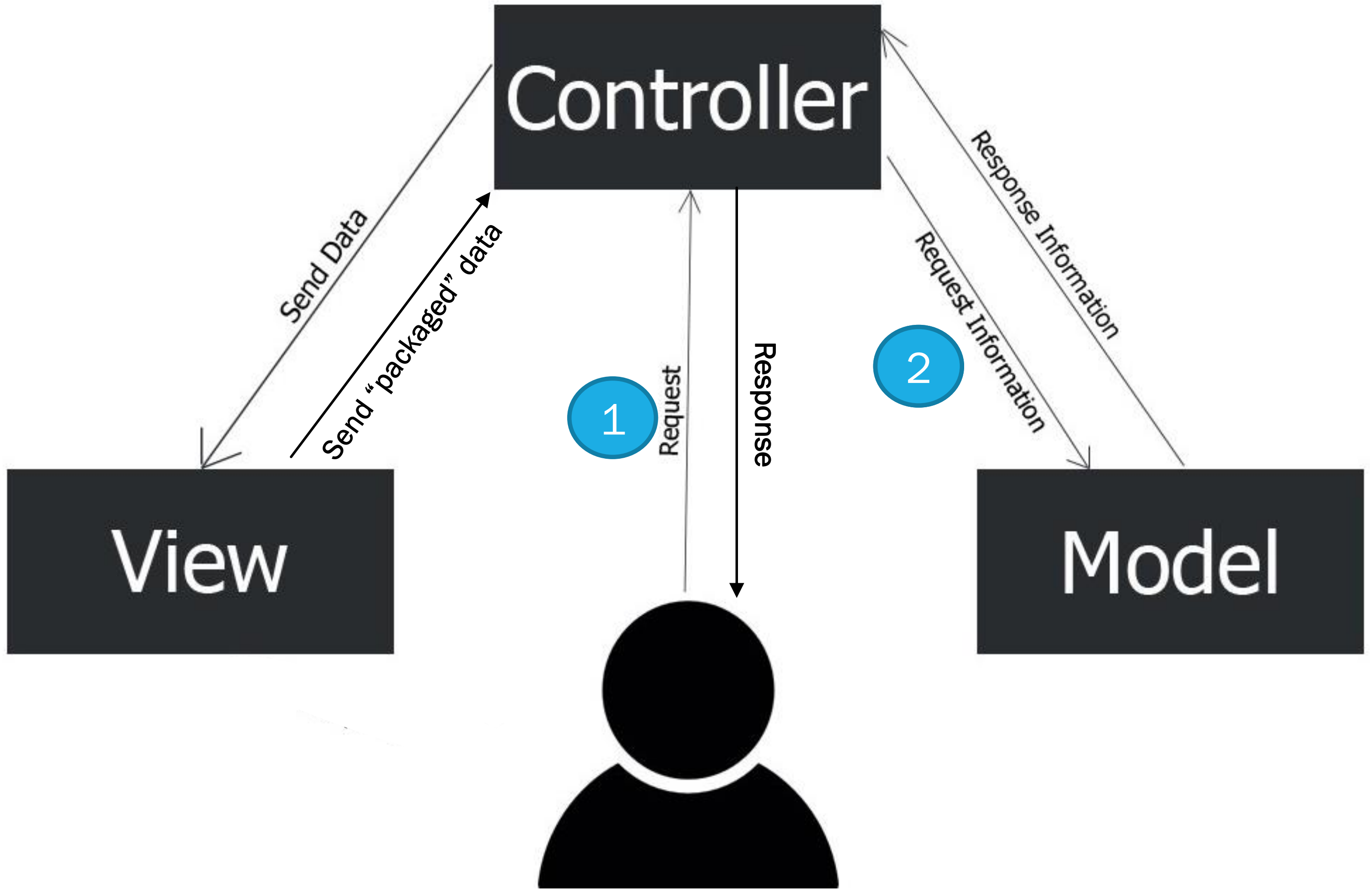
- M – Model
- V – View
- C – Controller



UNDERSTANDING THE CONTROLLER

The controller stands in the forefront of the end user's interaction with the application.

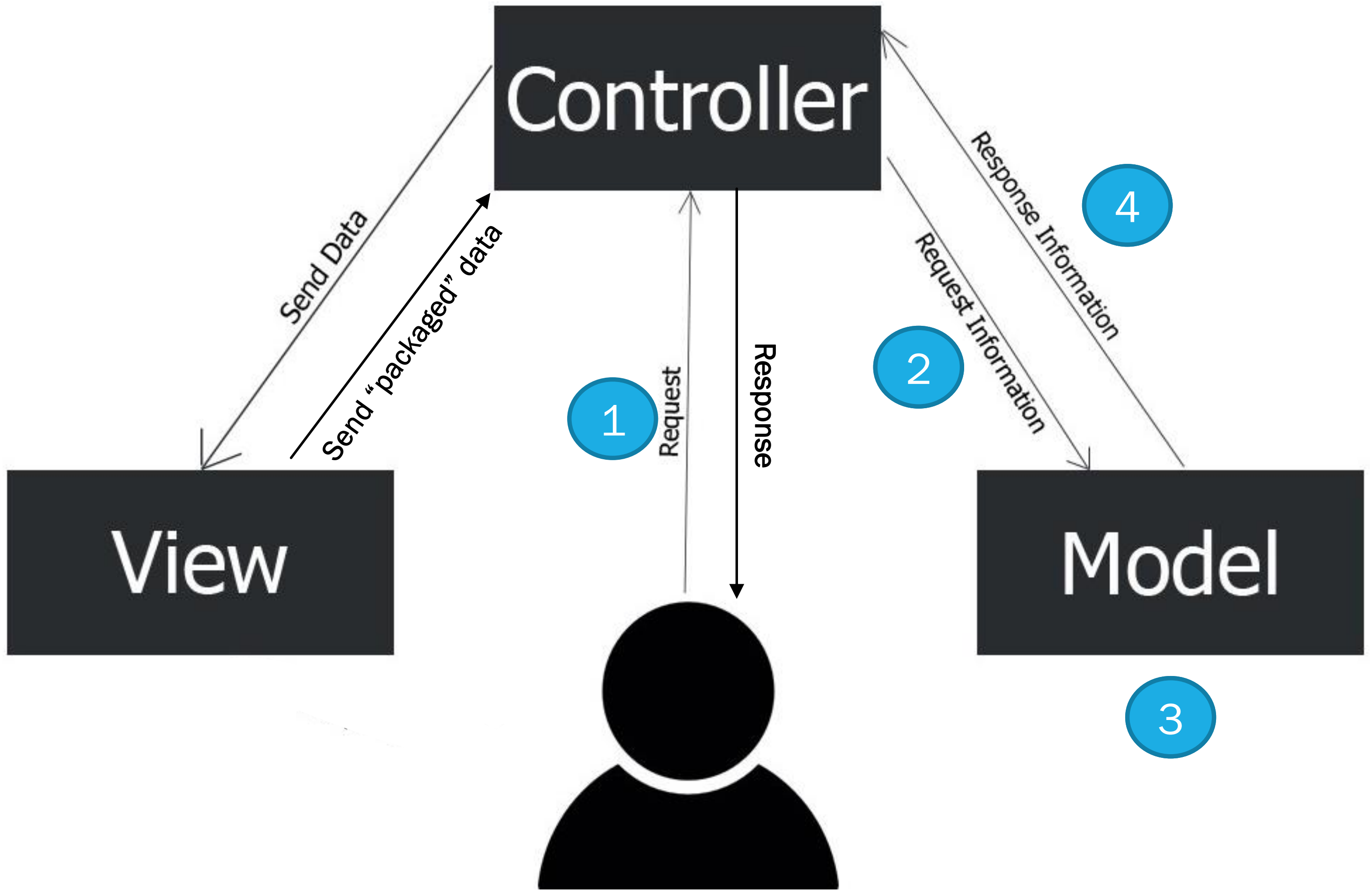
1. The controller accepts the user input
2. Handles the request by "translating" it into a command directed towards the model (retrieves data organized by the model)



UNDERSTANDING THE MODEL

The model outlines how raw data is organized in the database.

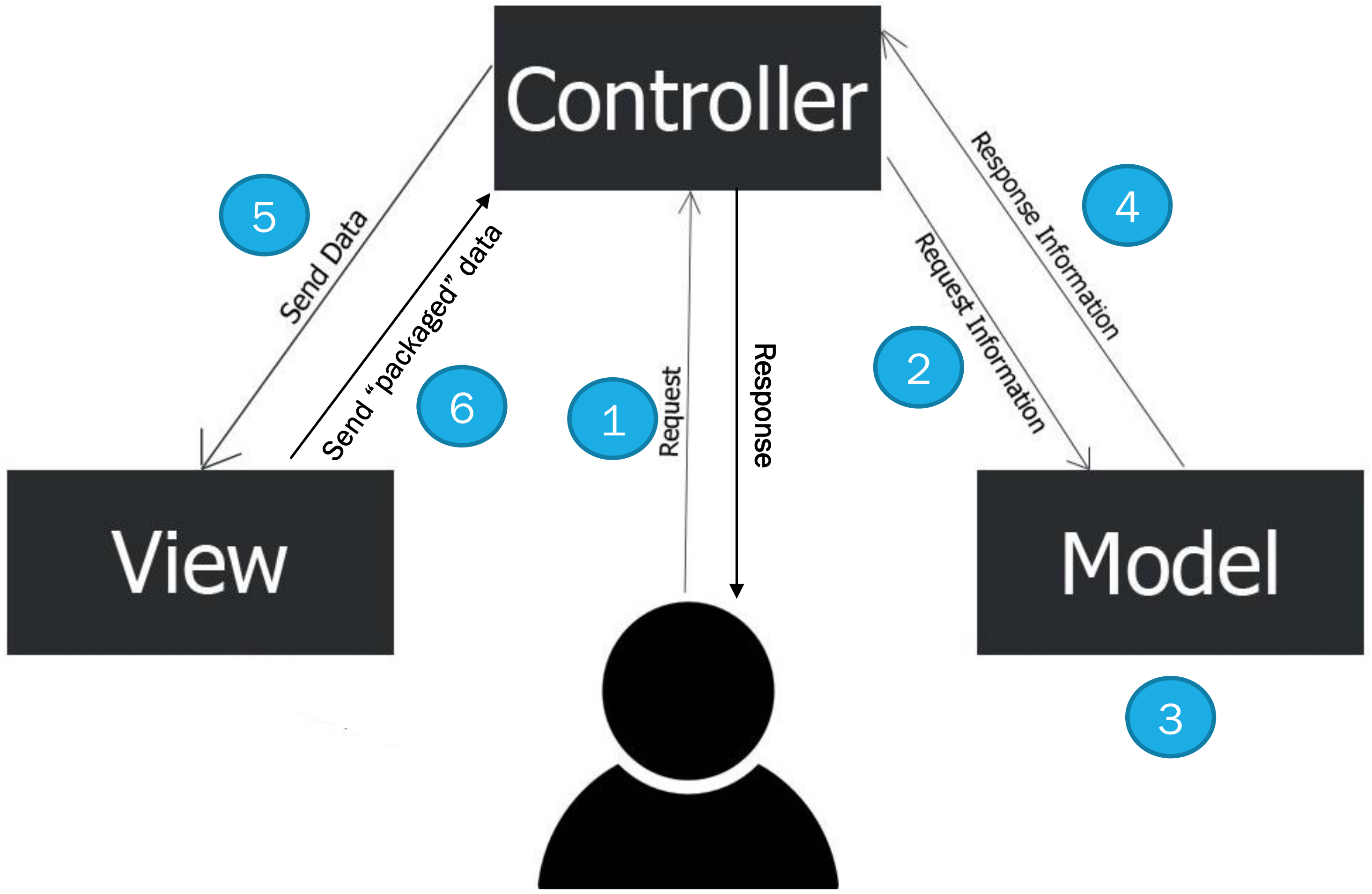
3. The controller interacts with the database via the model – this entails performing CRUD (CREATE, READ, UPDATE, DELETE) operations to data elements
4. It then returns the data from the database in response to the controller's request



UNDERSTANDING THE VIEW

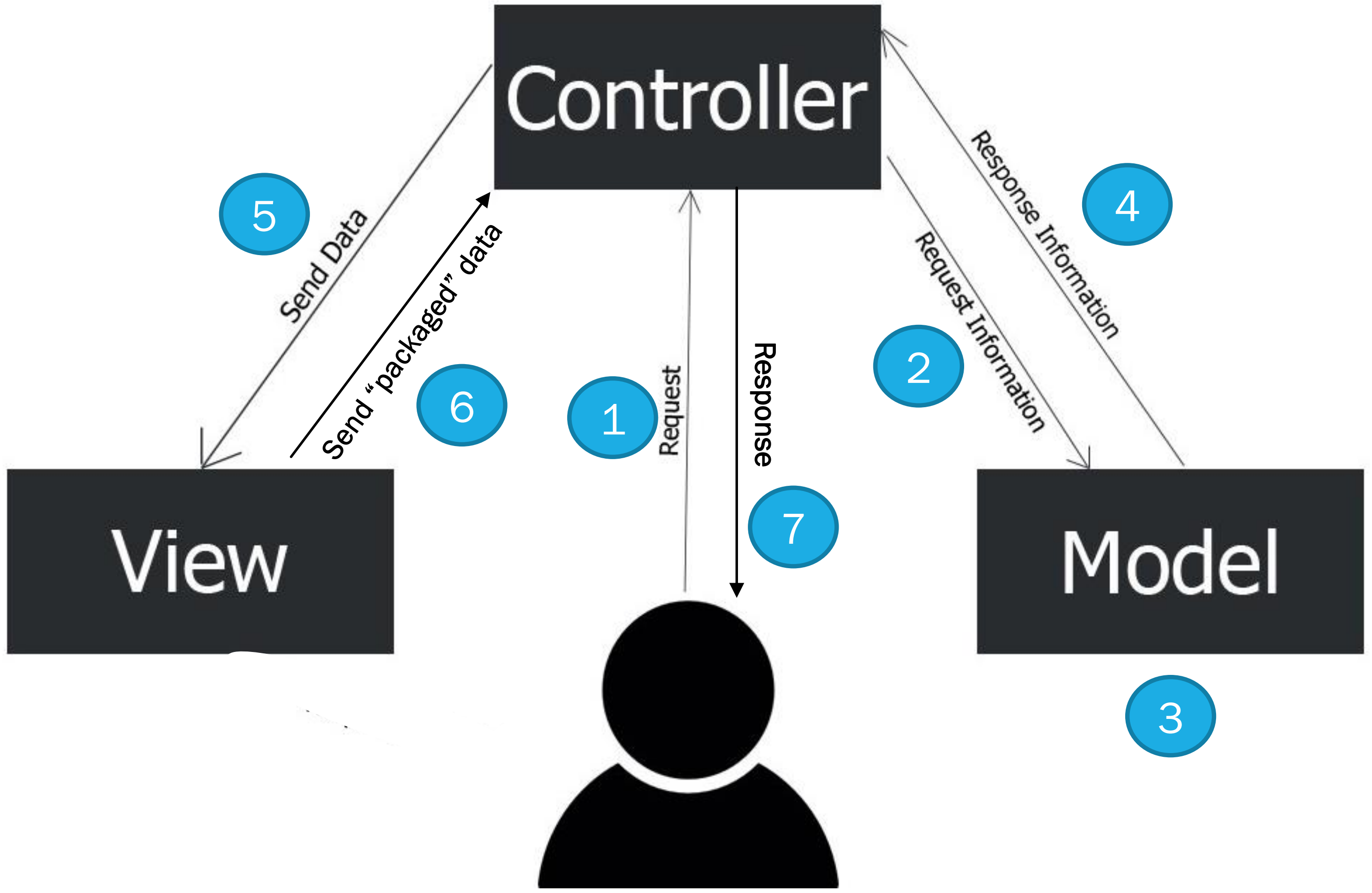
The view provides a template for how the data is to be displayed

5. The controller interacts with the view, which packages the data that was sent by the controller
6. The view sends this newly formatted data back to the controller



UNDERSTANDING THE CONTROLLER

7. The controller renders the data that was formatted by the view to the end user



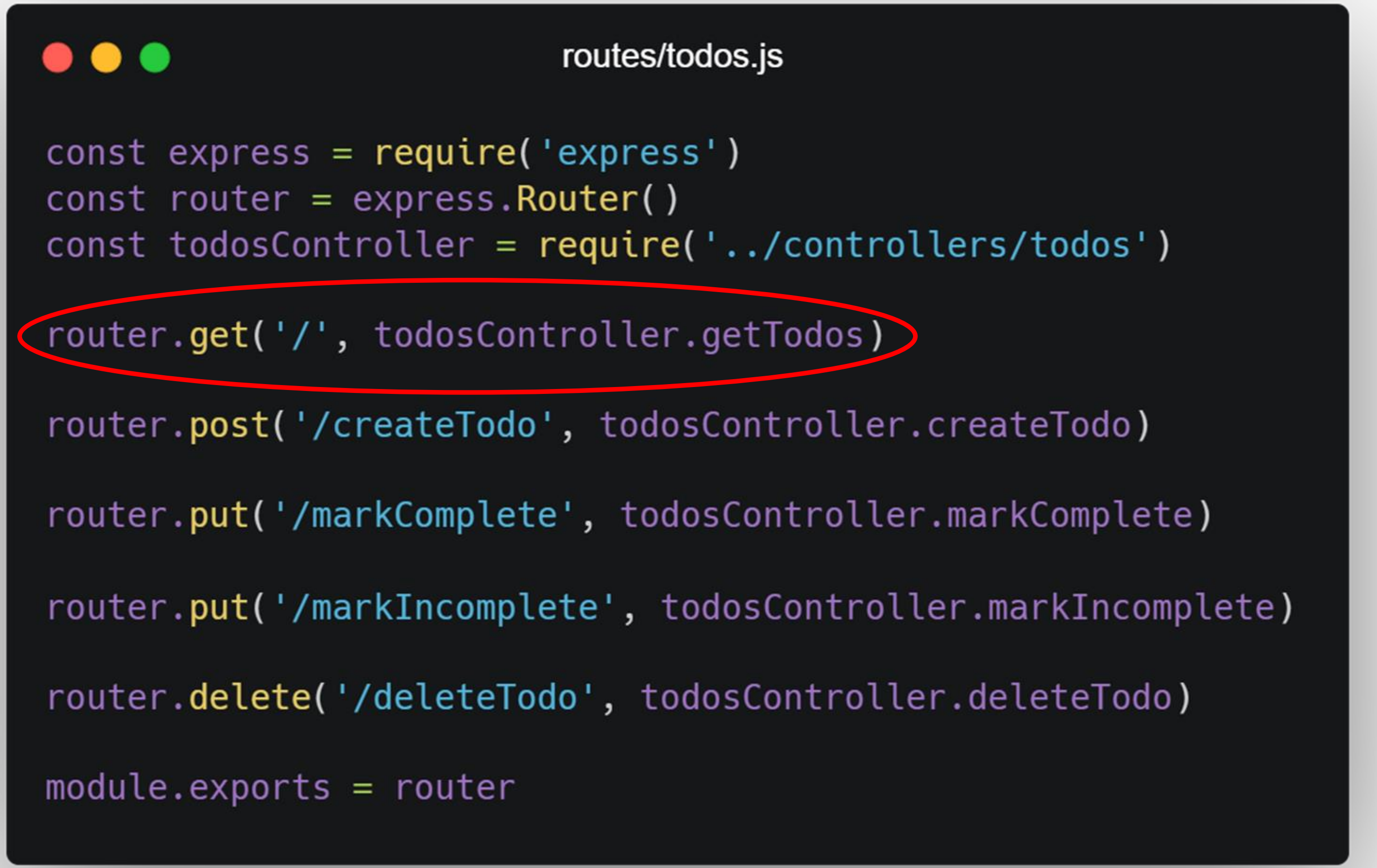
MVC IN THE REAL WORLD

A DEMO USING THE TO-DO LIST APPLICATION

ROUTE

While a Router is **not** a Controller, it directs the browser request to the corresponding controller action so that the correct response is returned for a given URL.

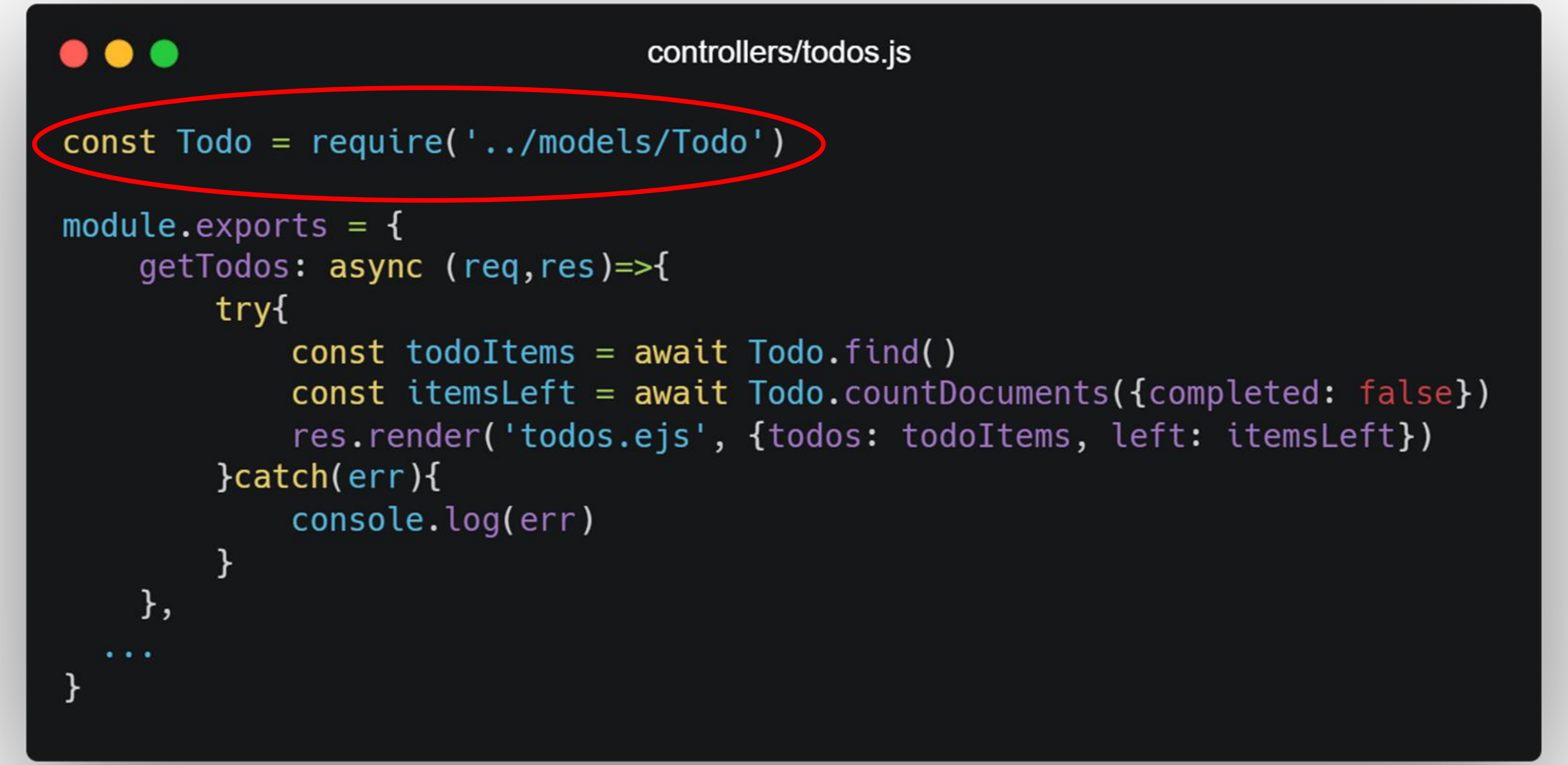
Here, accessing the homepage of the application sends a GET request to the todosController controller and calls the getTodos function



```
const express = require('express')
const router = express.Router()
const todosController = require('../controllers/todos')

router.get('/', todosController.getTodos)

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

router.put('/markComplete', todosController.markComplete)

router.put('/markIncomplete', todosController.markIncomplete)

router.delete('/deleteTodo', todosController.deleteTodo)

module.exports = router
```

CONTROLLER

Before defining any method, the Controller imports data from the Todo model

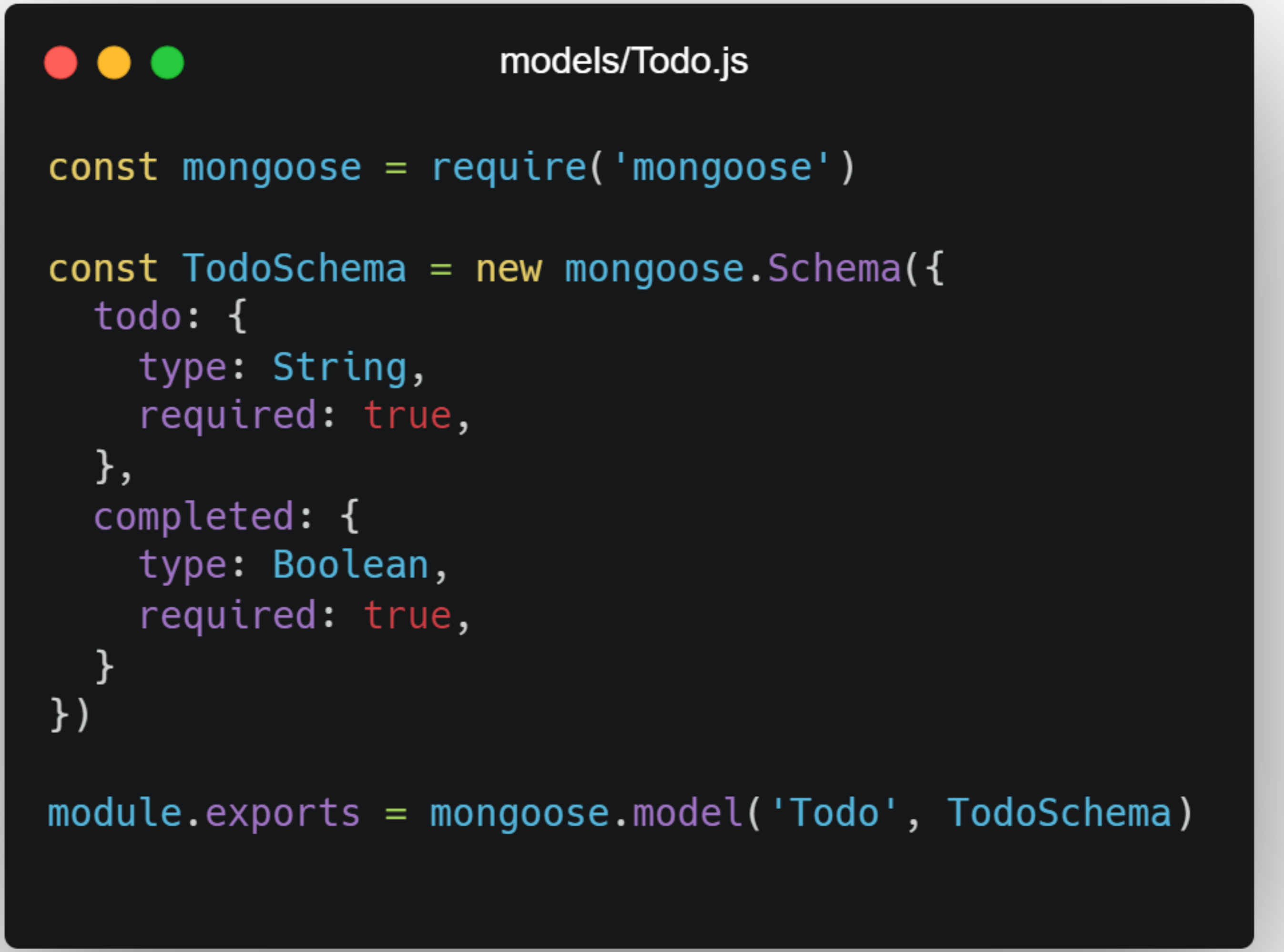

controllers/todos.js

```
const Todo = require('../models/Todo')

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)
        }
    },
  ...
}
```

MODEL

The Model outlines how data is organized in the database. In this application, the structure that it provides for each Todo (task) is the name of the todo and a completed value

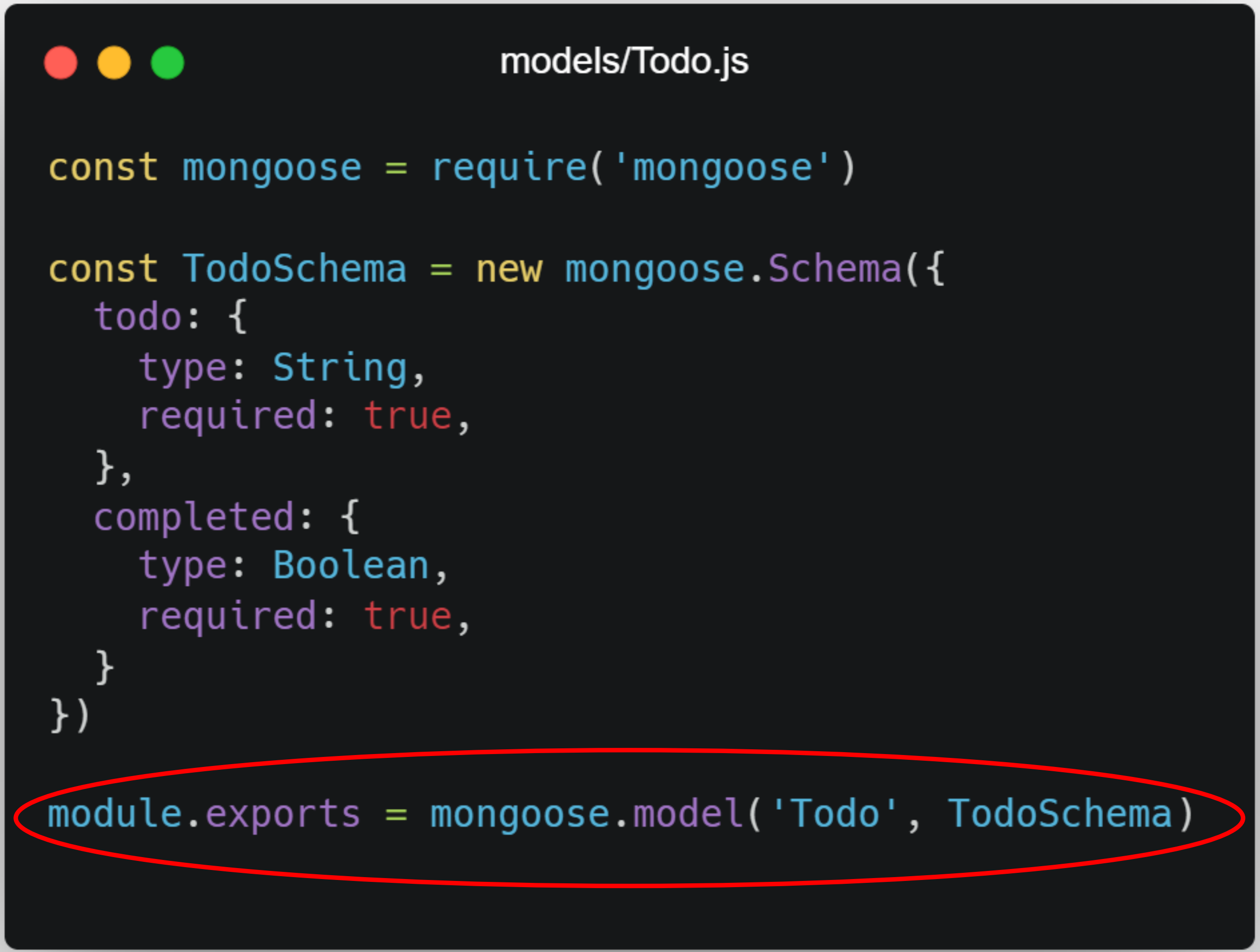
For instance, a “Take the dog out” task that you have not completed would have the todo value of “Take the dog out” and completed value of false.


models/Todo.js

```
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)
```

MODEL

The model exports the todo model, this allows the Controller to perform CRUD operations to data elements in the Todo database



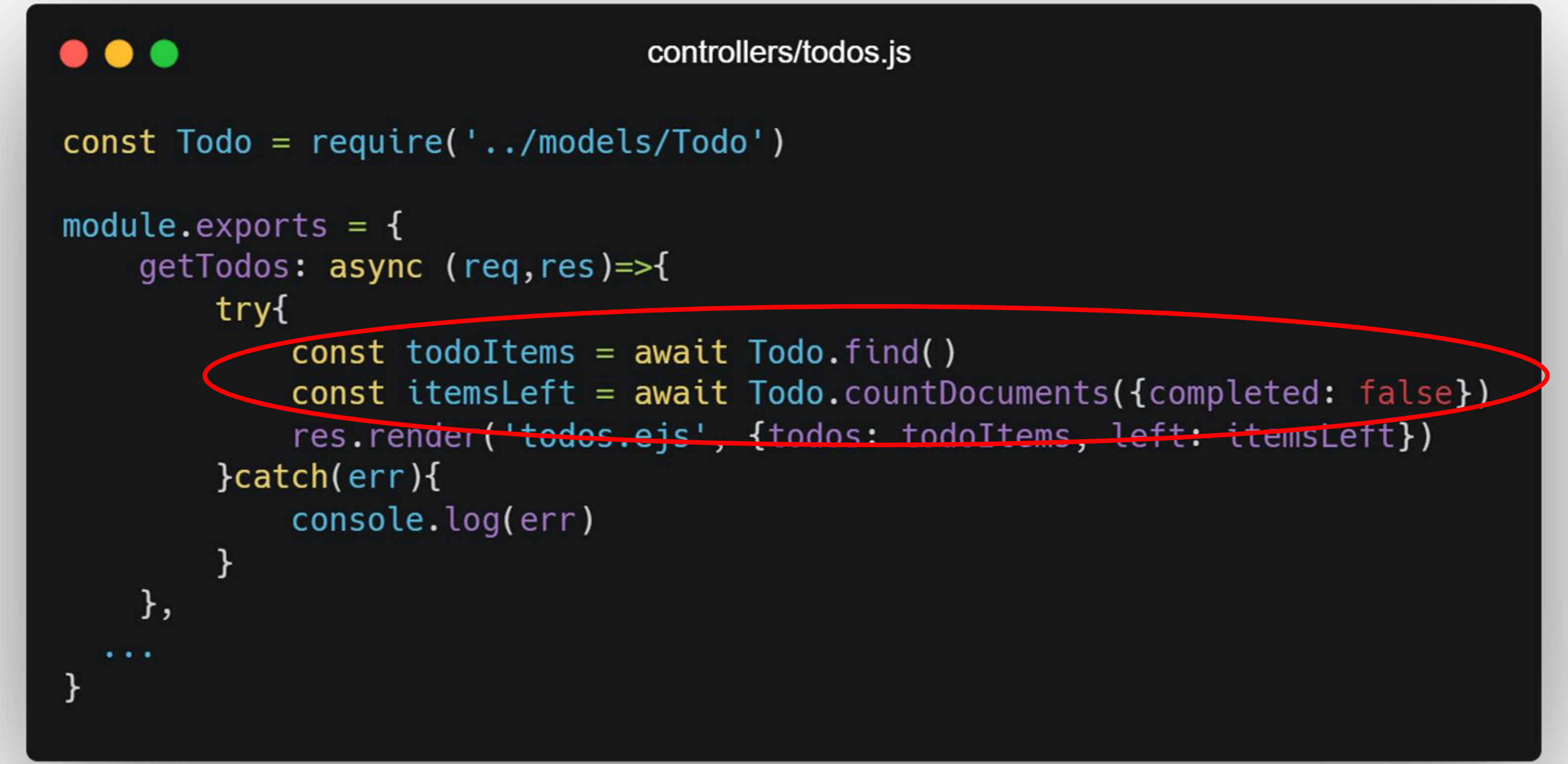
models/Todo.js

```
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)
```

CONTROLLER

The Controller then defines functions that are called to modify data elements in the database

In this example, calling getTodos results in getting the number of tasks that have not been completed from the database

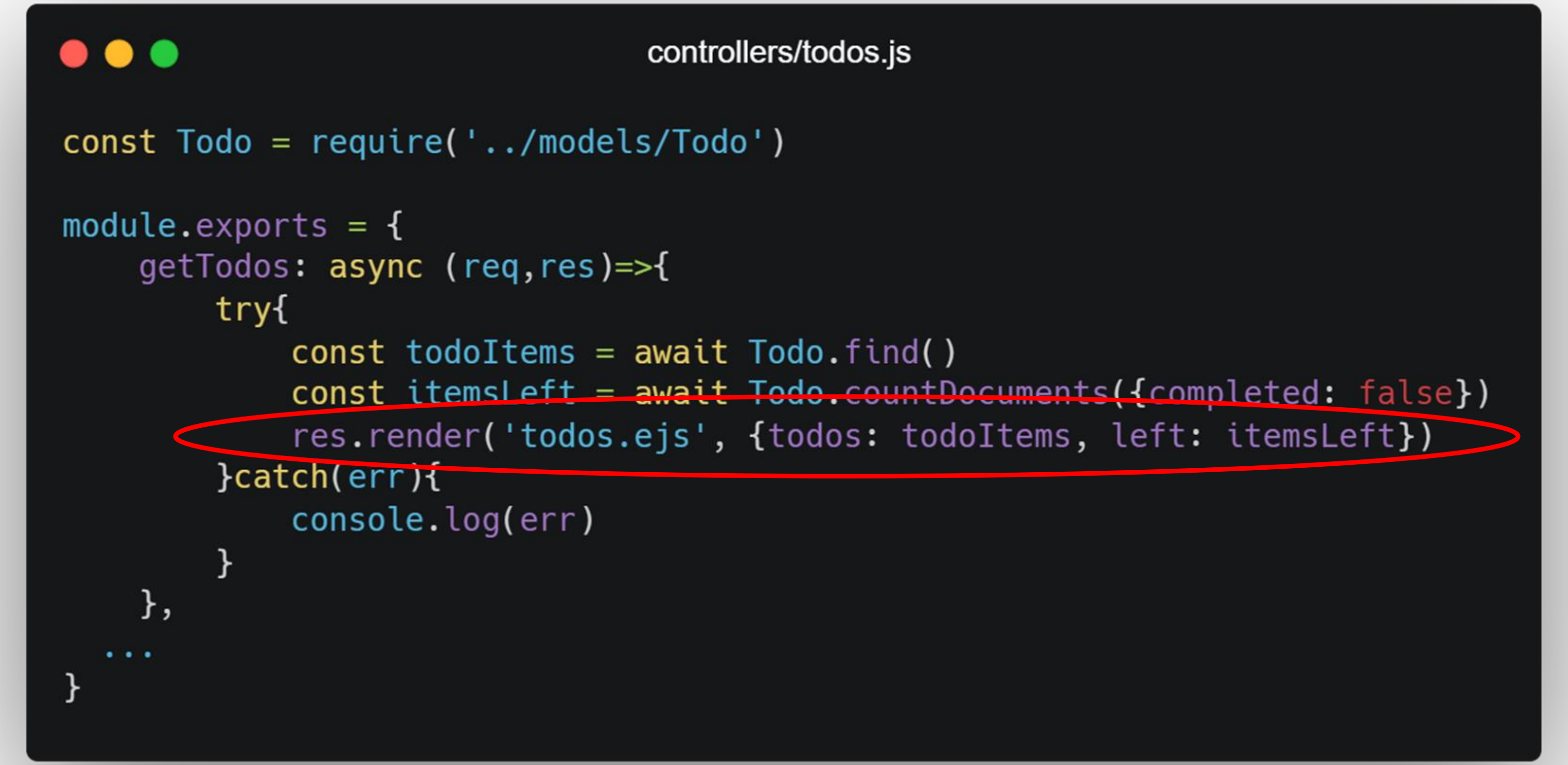


```
const Todo = require('../models/Todo')

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)
        }
    },
  ...
}
```

CONTROLLER

Next, the Controller receives the layout of how data will be presented to the end user from the View

Here, the value of “todoItems” - the tasks themselves - will be rendered where the variable “todos” is seen in the View file.

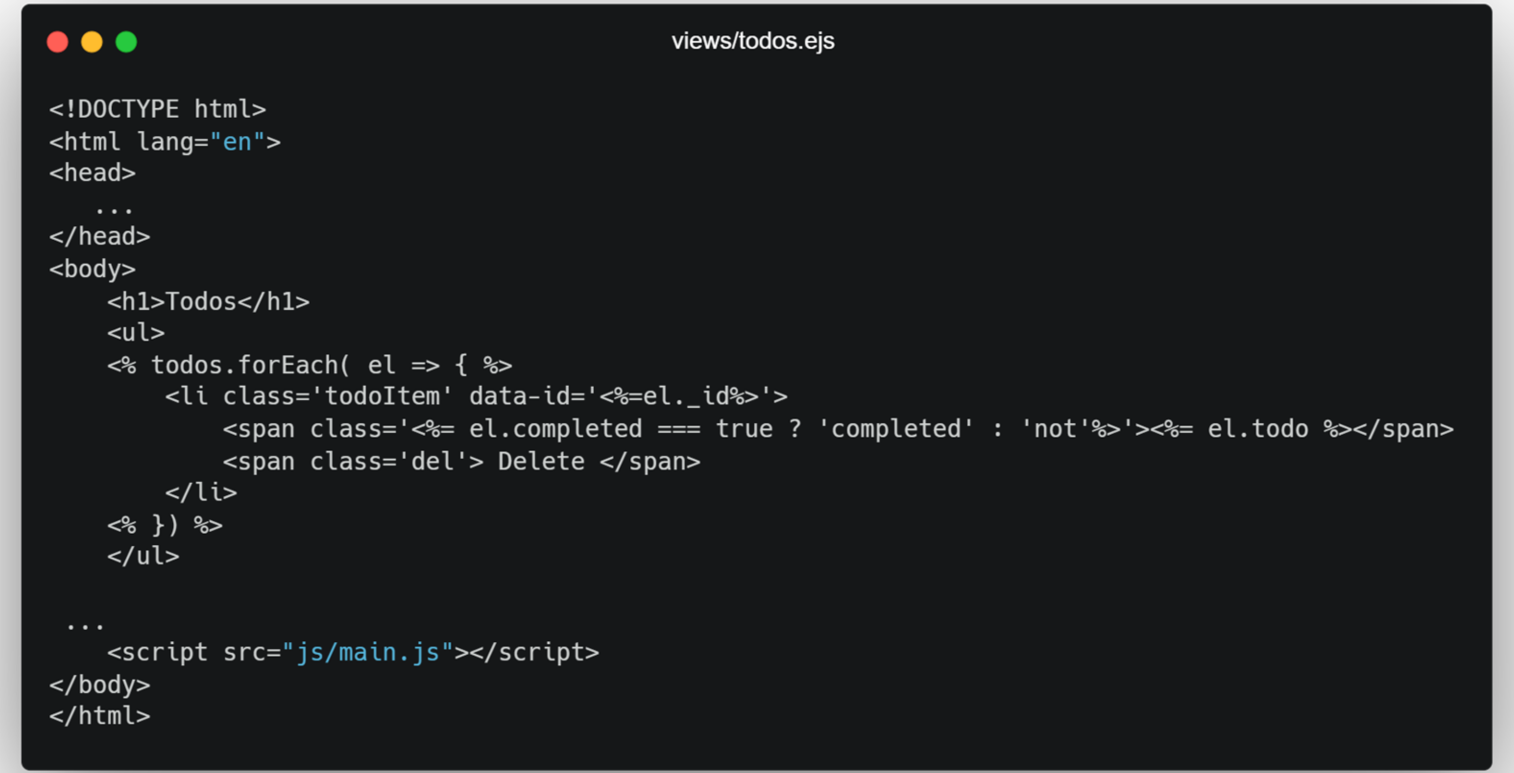

controllers/todos.js

```
const Todo = require('../models/Todo')

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)
        }
    },
  ...
}
```

VIEW

The view provides a template for how the data is to be displayed to the end user

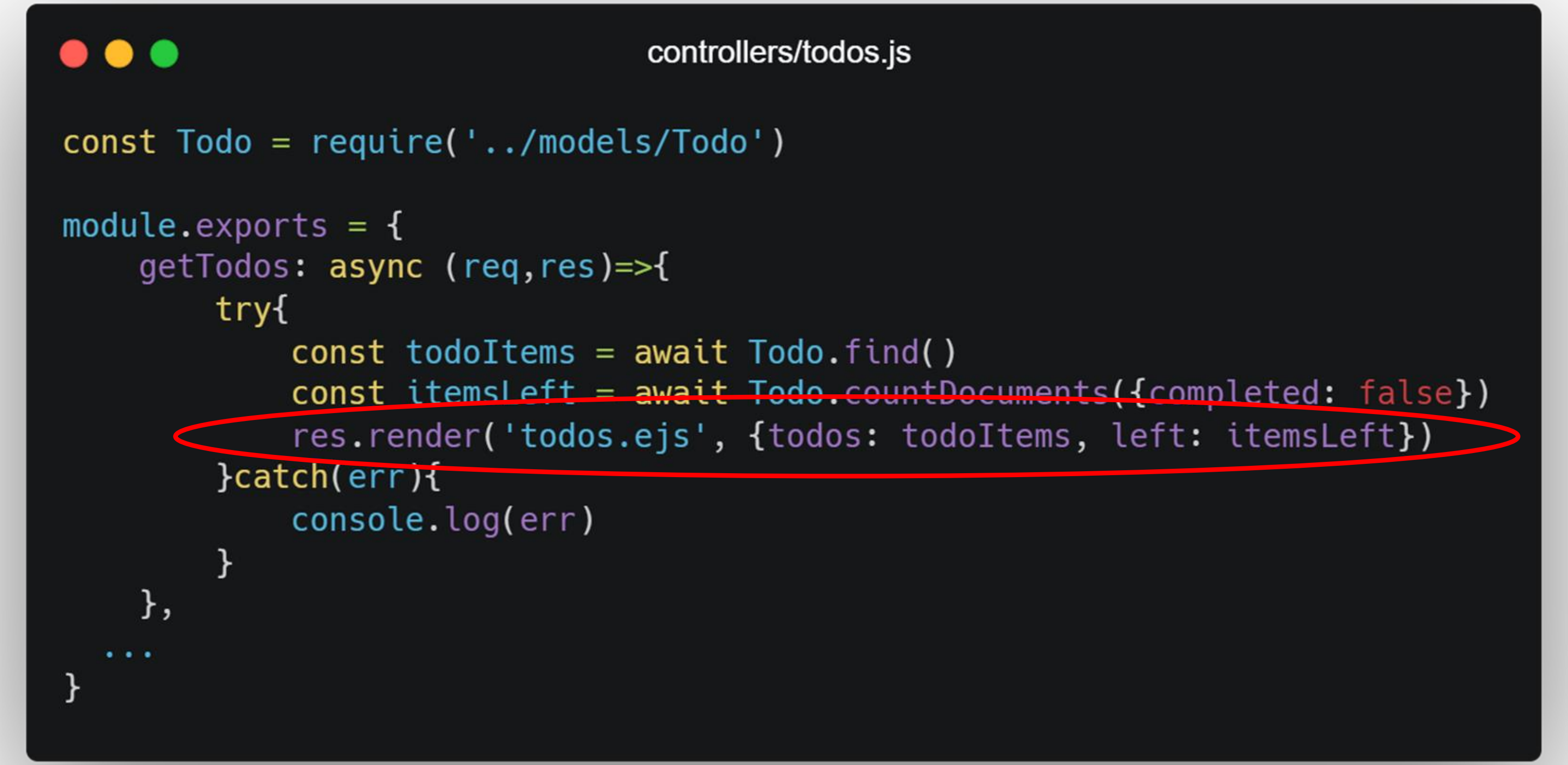
For instance, each todo (task) will be presented as a point on a list, with an <li> for the name of the task, a <span> that indicates whether the task has been completed, and a <span> that will allow the user to delete the task from the list


views/todos.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
   ...
</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>

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

CONTROLLER

Finally, the Controller will render the data in accordance with the layout specified by the View in its response to the user

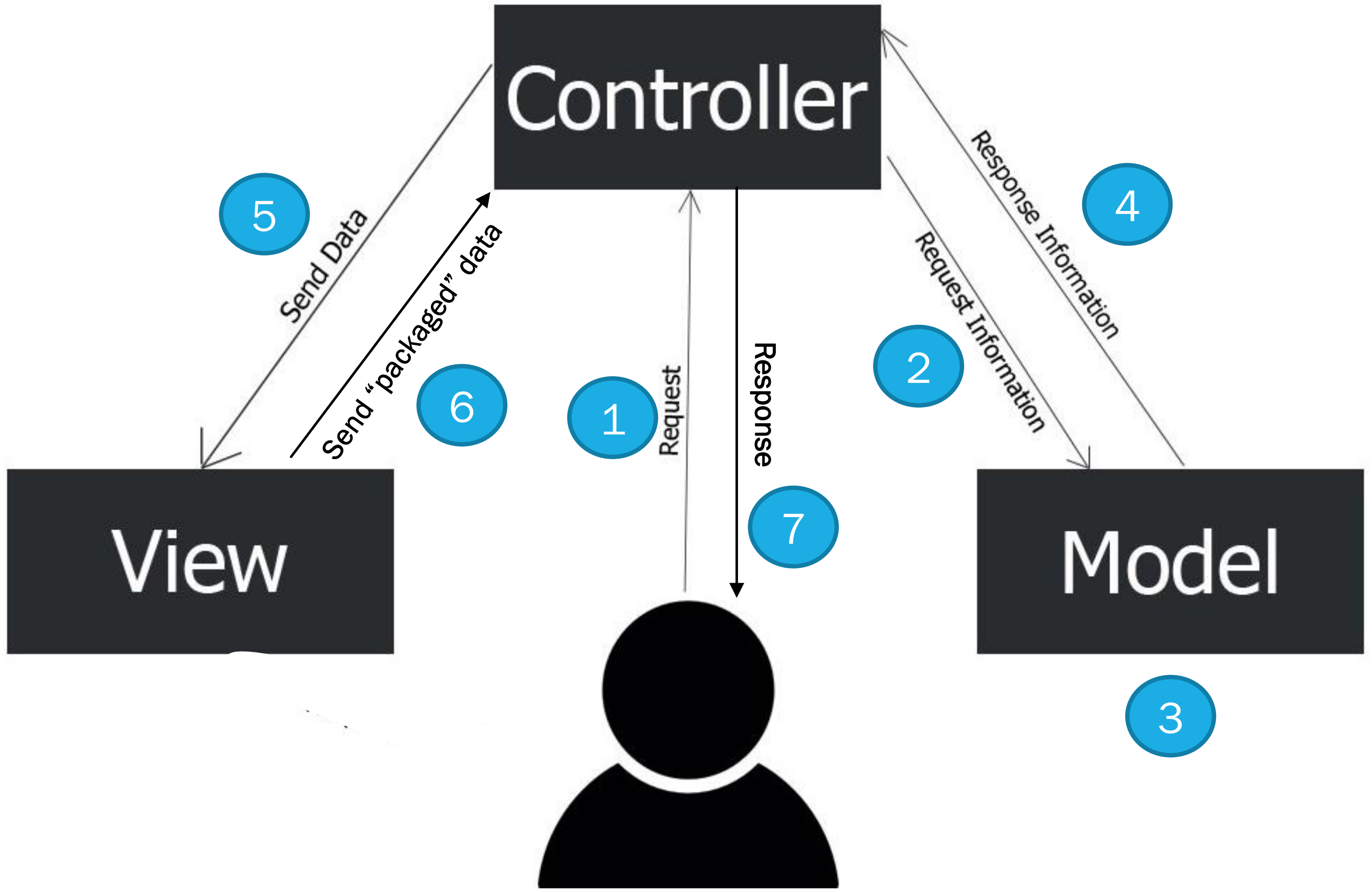

controllers/todos.js

```
const Todo = require('../models/Todo')

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)
        }
    },
  ...
}
```

THE PURPOSE OF MVC

It's all boils down to Separation of Concerns!

- Creates modularized, reusable code – each component has its own purpose
- Allows for easier debugging and collaborative efforts among multiple developers


Controller
View
Model
1
Request
2
Request Information
3
4
Response Information
5
Send Data
6
Send "packaged" data
7
Response