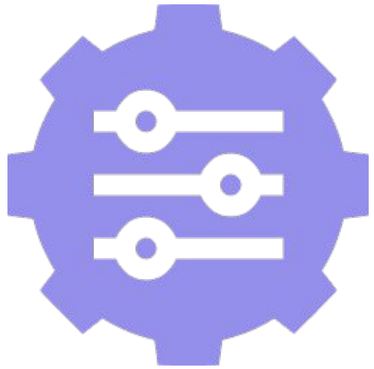
Introduction to MVC

Relevel
by Unacademy

What is MVC

The Model-View-Controller is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. This is used in combination with the front end and back end. Each of these components is built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development frameworks to create scalable and extensible projects.



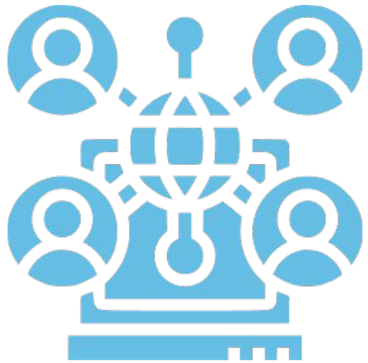
MVC Basics

What is the model?

The Model component corresponds to all the data-related logic that the user works with. This includes organising the data from the database in a presentable format and sending it as a response. This can represent either the data transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it, and update it back to the database or use it to render data.

What is a view?

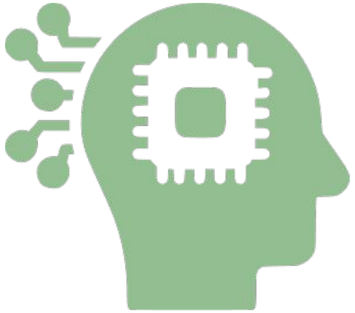
The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc., that the final user interacts with.



MVC Basics

What is the controller?

Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component, and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.



MVC Basics

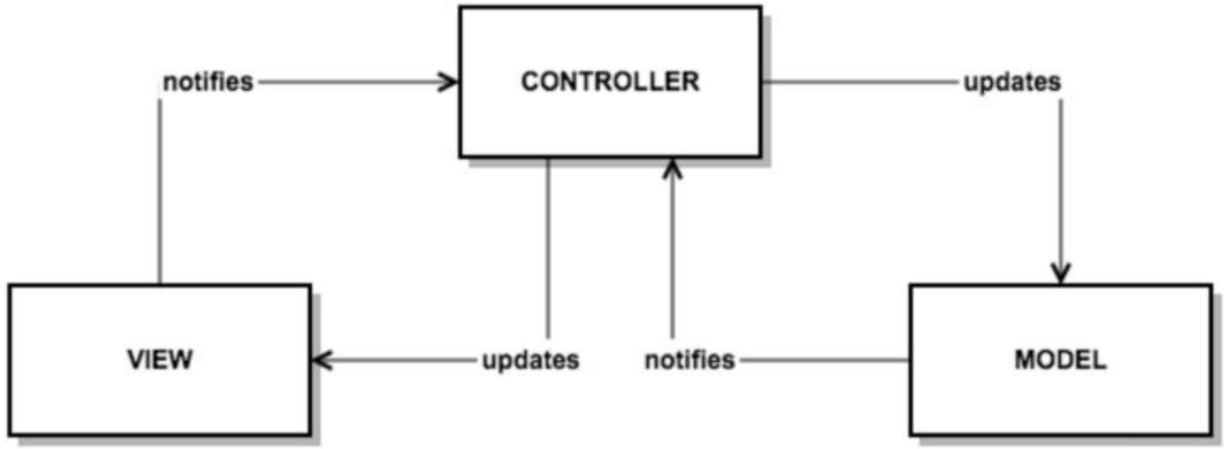


Fig. 1. Apple's Model-View-Controller architectural overview

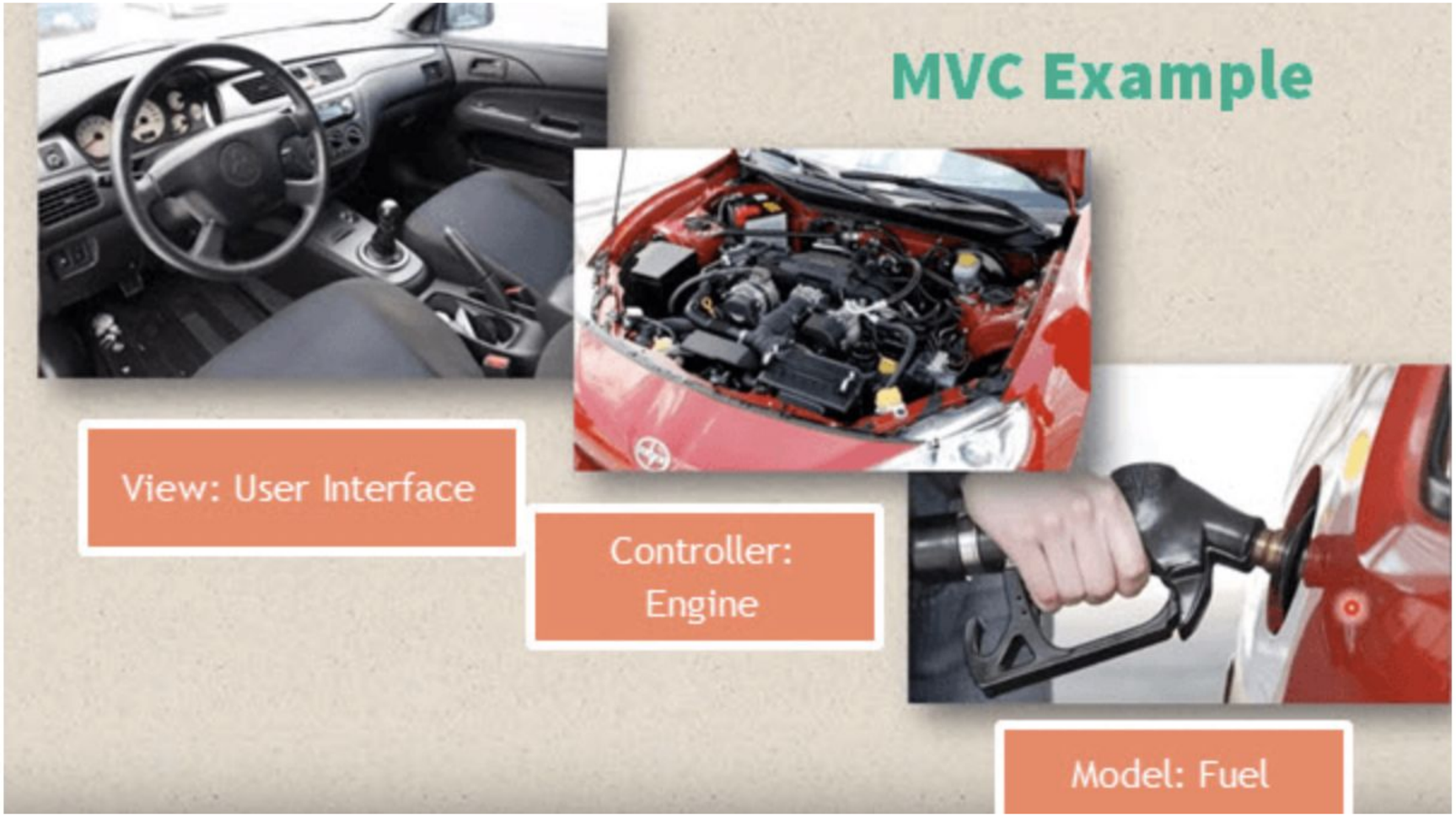
Example of MVC

The car driving mechanism is another example of the MVC model.

- Every car consists of three main parts.
- The user interface's view includes gear lever, panels, steering wheel, brake, etc.
- The controller is the mechanism which is Engine
- The model means the storage, which is the Petrol or Diesel tank

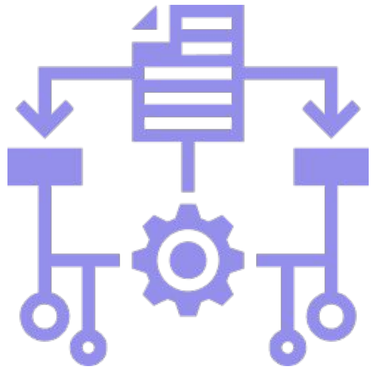
Car runs from the engine take fuel from storage but only uses mentioned user interface devices.

Example of MVC

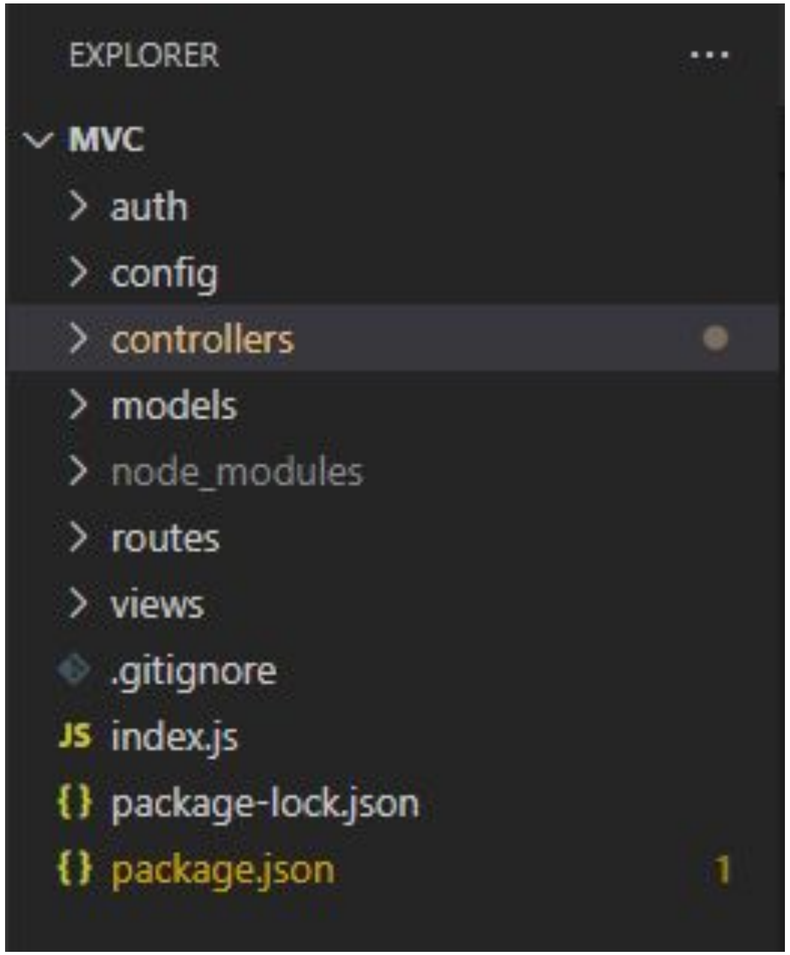


MVC using Node and Express

- We can have a basic approach to work with nodes and express while building an MVC. Now let’s try to clarify why using an MVC is useful and how it can turn your standard Express/Node.js App into a high-level app just by wrapping around an MVC Architecture.
- The code will contain only one home page and some templates. Here we are trying to create a login app with the MVC architecture. This is an example to give clarity around how it works.
- Since the Architecture comprises three main parts (Controller, Model, and View) so under the src/ folder, we need to have the other subparts in different folders.
- The code can be accessed through link https://github.com/VJ28/mvc_login_app



MVC using Node and Express



Data Modelling

The Model-Viewer-Controller is a dominant paradigm for application development. The MVC approach separates key common concerns for organized, manageable application code.

The model classes represent domain-specific data and business logic in the MVC application. It represents the shape of the data as public properties and business logic as methods.

We would be defining all the Models in the Model folder.

What is ORM?

Object-relational mapping is the process of mapping between objects and relational database systems. Multiple databases use data in different ways, and object-relational mapping maintains objects when the sources and app they access can change with time. ORM is used to handle the passage of data between different databases.

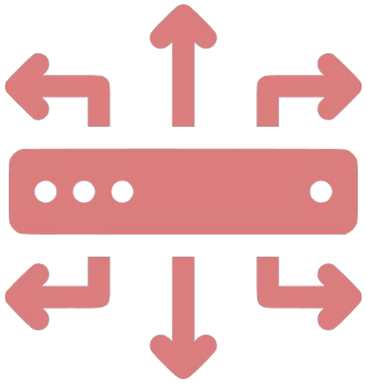
Why do we need ORM?

ORM helps to implement the principle of Do Not Repeat Yourself.
It helps with marshalling. ORMs generally allow fetching complete objects from the database rather than row objects to wrap yourself.

Routing in Express

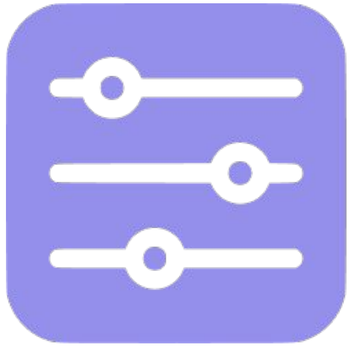
Routing is very crucial. It defines the URL structure that can be used to interact with the web application.

Express apps use routers that are containers for a bunch of middleware. The middleware holder can be used on a certain route, which enables to place the logic in separate files and bring them together.



Route handler Callback Function

Assume a /login route, the callback will be stored in separate "controller" modules for Users. It can be started with a /controller folder followed by different controller files/modules for handling each of the models:

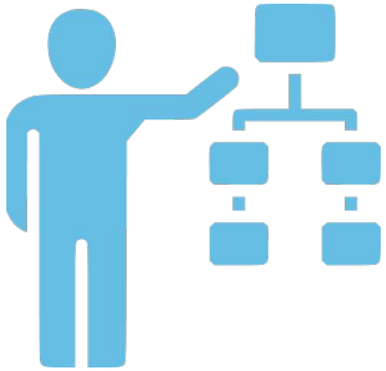


/controllers
    dashboardController.js
    loginController.js

It can be access here -
https://github.com/VJ28/mvc_login_app/blob/main/controllers/loginController.js

Code Explanation:

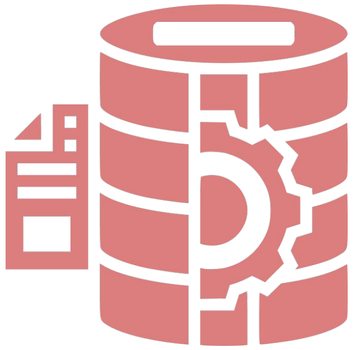
We have different functions which handle login, register or return html as when called. During login, it first needs to verify if the passed credential is correct and later needs to check in db if the user exists. For registration, it must check if password and confirmPassword are the same and then check if the user already exists, else if everything is fine create a new user.



Setting up Database

You can install MySQL client and Sequelize by executing the following command:

```
npm install sequelize mysql2
```



Then we can add a models directory with a file called index.js, which will contain the database and Sequelize.js setup. Code can be accessed here - https://github.com/VJ28/mvc_login_app/blob/main/models/index.js

First, we include the modules that we’re going to use:

- Sequelize allow us to create a new Sequelize instance and connect to the db
- We also define a User model and add it into our db object to make it accessible outside.
- require("./User.js")(sequelize, Sequelize)
- The db object which corresponds to the database is exported, and it corresponds to the database methods for each model. We can access it when we want to fetch the data from the database.

Thank You!