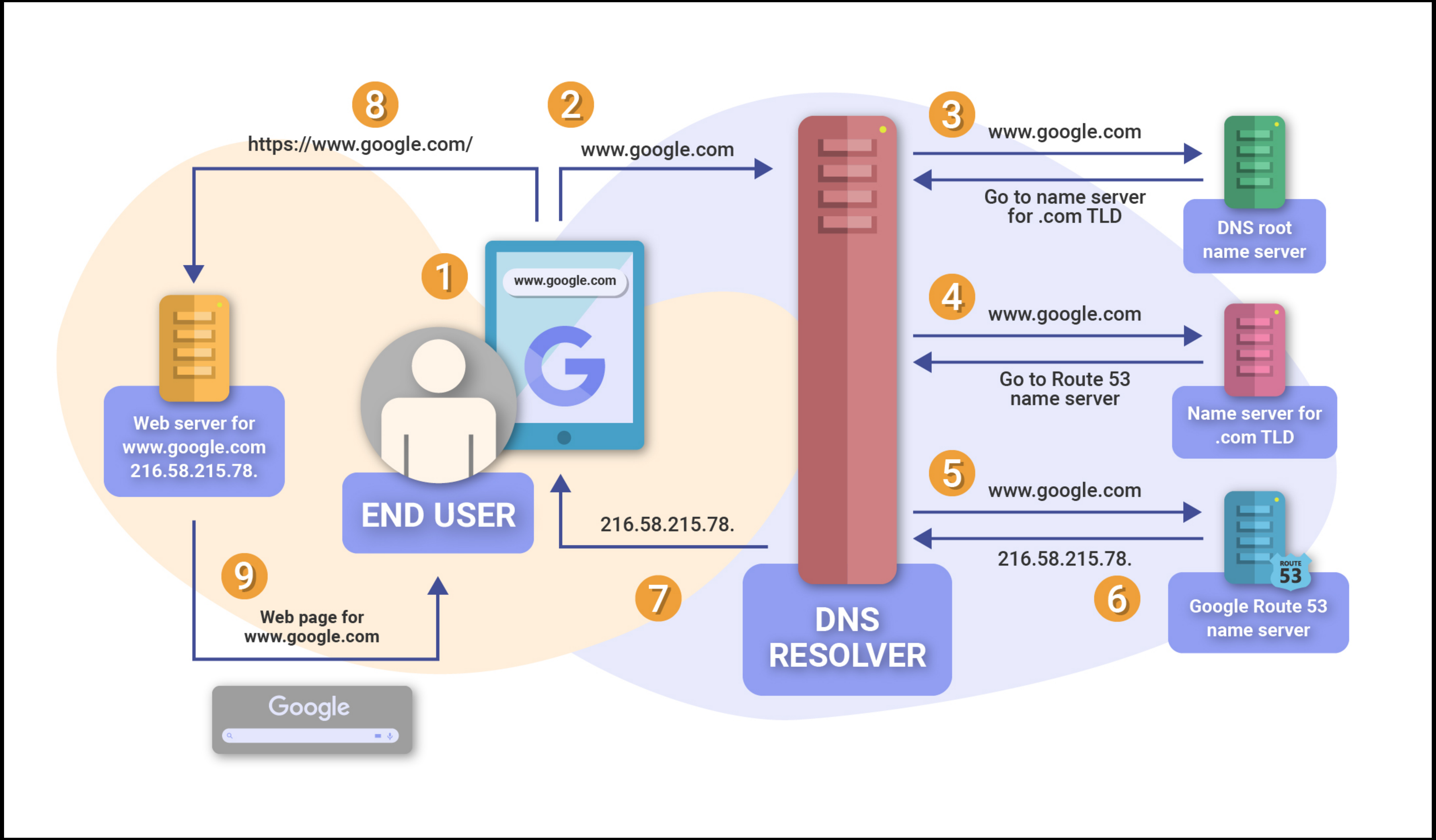
Backend Development - P1

Basics in Depth

- Love Babbar

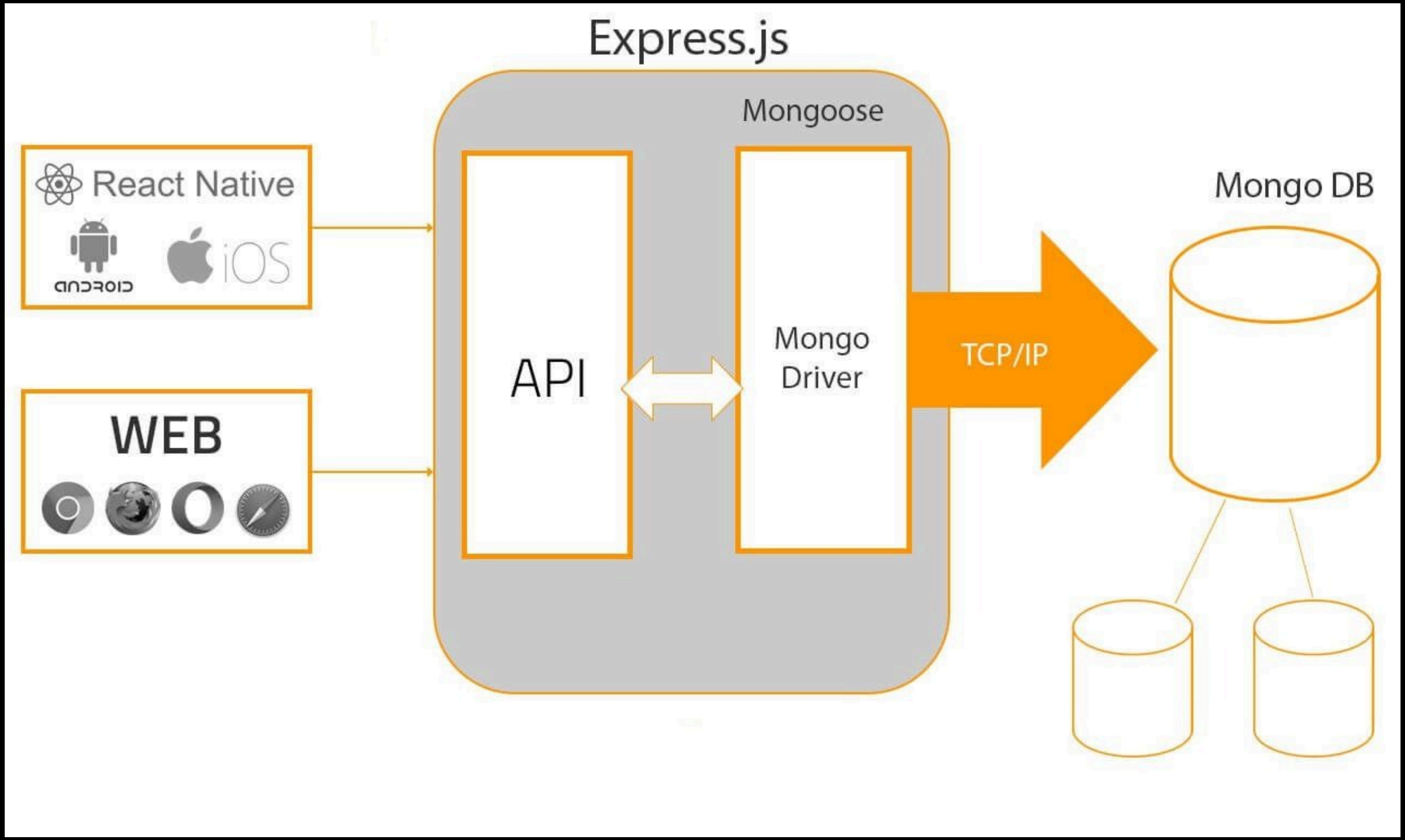
How Internet works ?



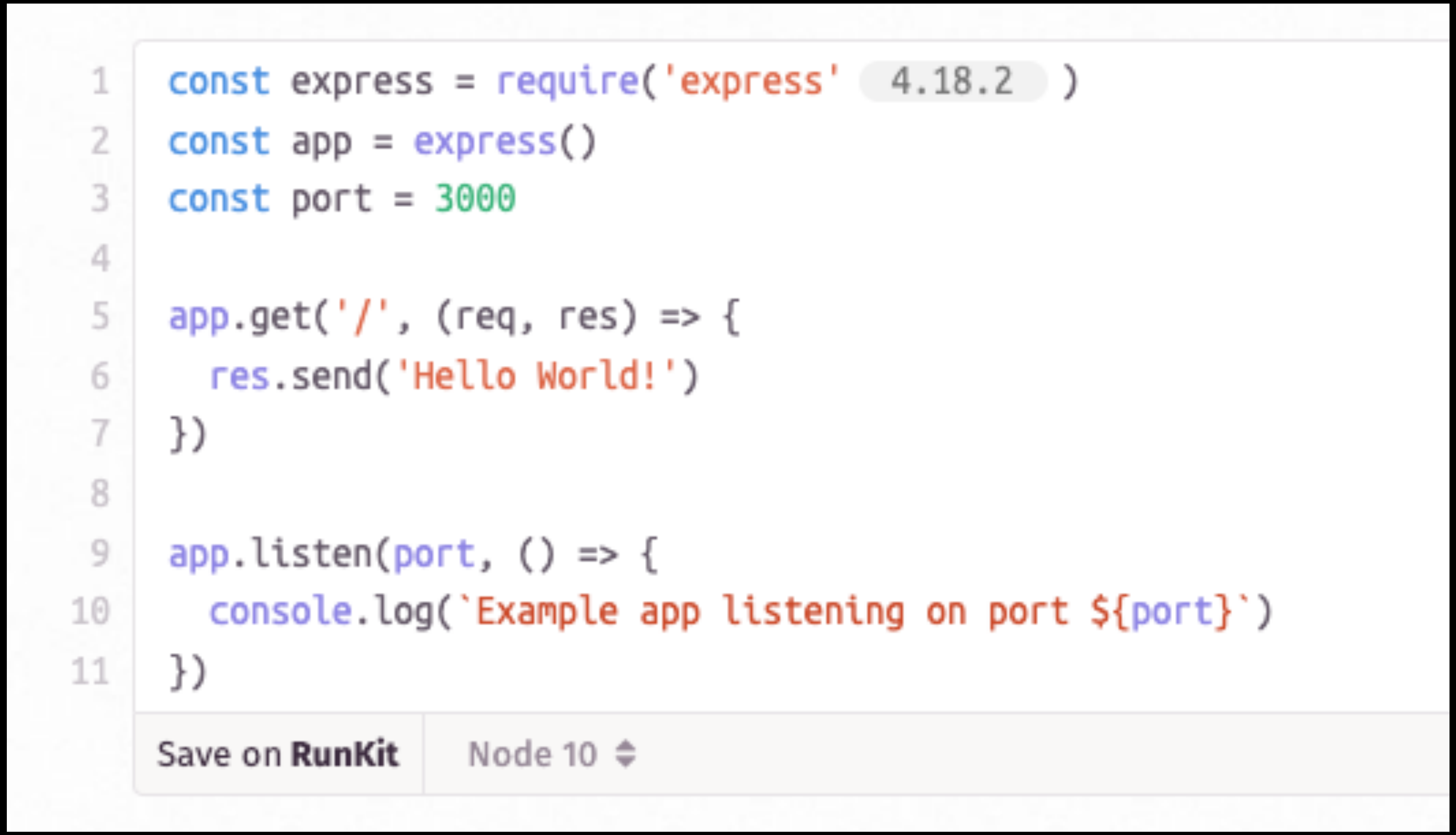
What is Express ?

- Express is a back-end web application framework for Node.js. It provides a set of tools and features that enable developers to build server-side applications that handle HTTP requests and responses, connect to databases, and perform other tasks. Express is designed to be flexible and modular, allowing developers to add middleware functions and customize the application to meet their specific needs.

Let’s learn:


Express.js
Mongoose
React Native
android
iOS
WEB
API
Mongo
Driver
TCP/IP
Mongo DB

Create a Application



```
const express = require('express'  4.18.2 )
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Port & Start a Server

- In the context of web development with Express, a "port" refers to a number that identifies a specific communication endpoint on a computer. When we start a server with Express, we need to specify which port the server should listen on, so that client applications can connect to it and communicate with it over the internet.

- The reason we need to specify a port is that a single computer can potentially run multiple server processes, each serving different applications or services. By specifying a port number, we can ensure that requests are routed to the correct server process and application.

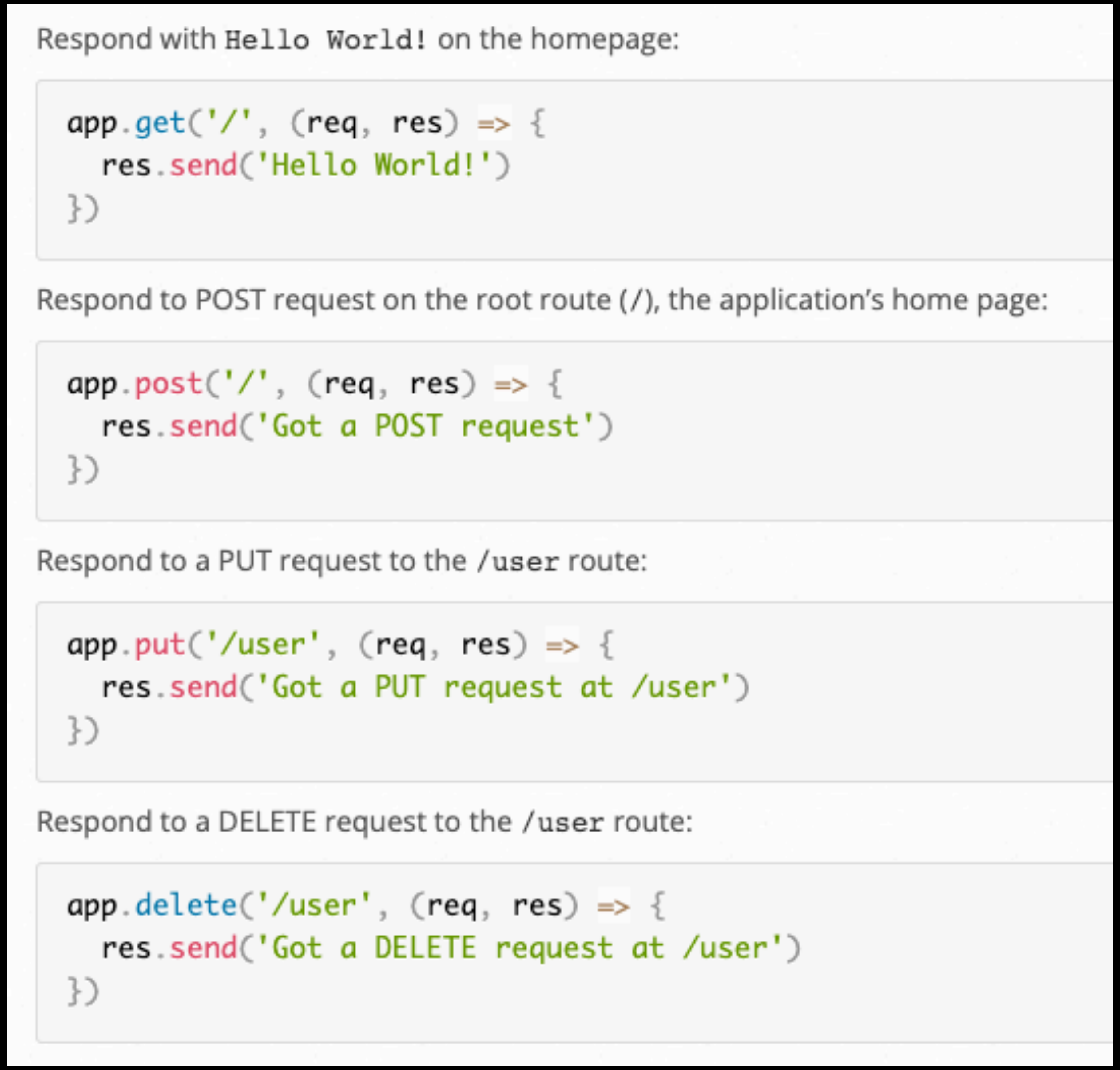
- By default, the port number used by Express is 3000, but this can be changed to any available port number that is not already in use by another process on the computer. To specify a different port number, we can use the app.listen() method

Middleware:

In Express, middleware refers to a function that processes incoming HTTP requests and can perform various actions such as modifying the request or response objects, invoking the next middleware function in the chain, or sending a response to the client

- Middleware functions can be used to perform a variety of tasks, such as:
- Logging: Middleware can be used to log information about incoming requests and outgoing responses.
- Authentication: Middleware can be used to authenticate users, check if they are authorized to access certain resources, and redirect them to login pages if necessary.
- Parsing: Middleware can be used to parse incoming request bodies, such as JSON, XML, or form data.
- Error handling: Middleware can be used to handle errors and exceptions that occur during the request/response cycle.
- Middleware functions are added to an Express application using the app.use() method, and they are executed in the order in which they are added to the middleware chain. This allows for modular and flexible application design, as different middleware functions can be added or removed as needed.

Routes



Respond with `Hello World!` on the homepage:

```
app.get('/', (req, res) => {
  res.send('Hello World!')
})
```

Respond to POST request on the root route (/), the application's home page:

```
app.post('/', (req, res) => {
  res.send('Got a POST request')
})
```

Respond to a PUT request to the `/user` route:

```
app.put('/user', (req, res) => {
  res.send('Got a PUT request at /user')
})
```

Respond to a DELETE request to the `/user` route:

```
app.delete('/user', (req, res) => {
  res.send('Got a DELETE request at /user')
})
```

Mounting

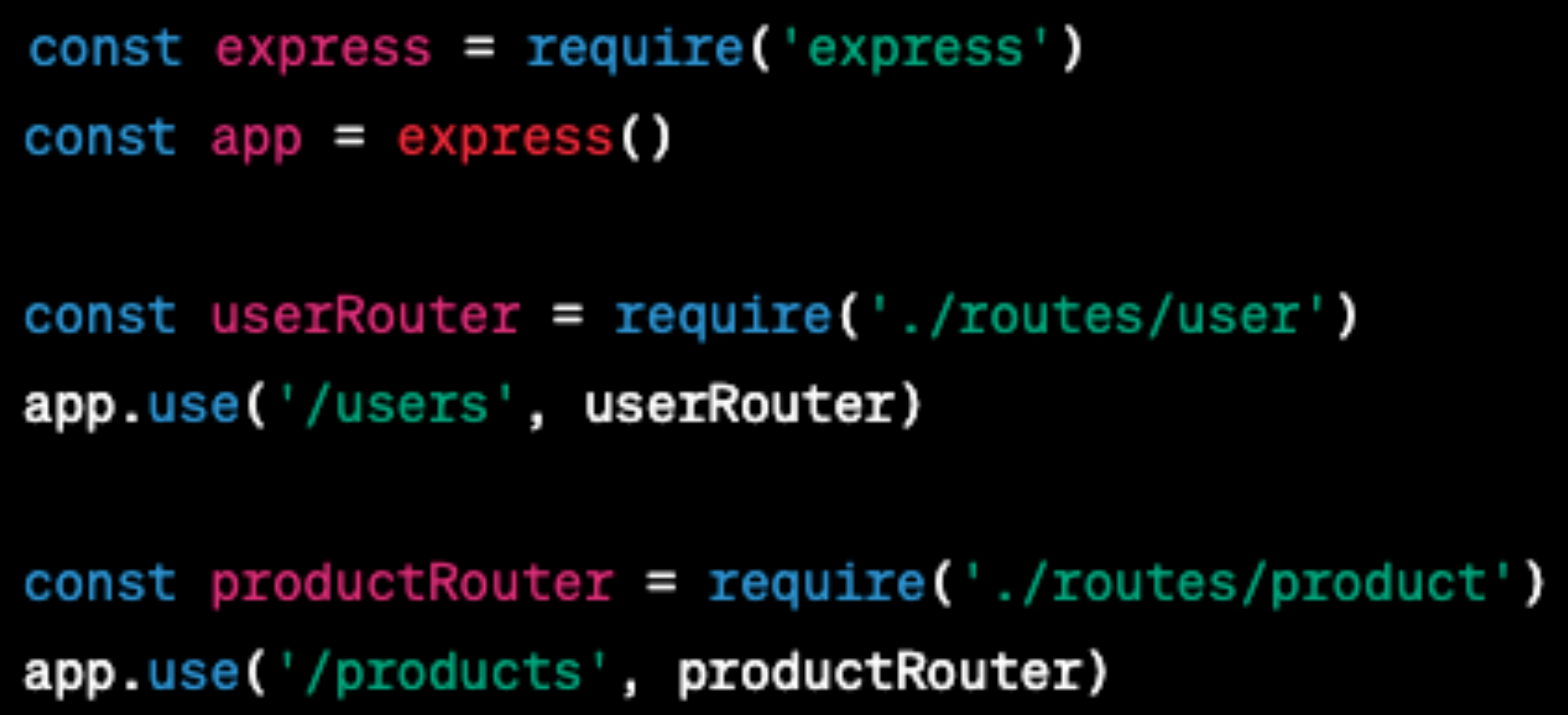
In Express.js, "mounting" refers to the process of attaching middleware or sub-applications to specific paths in the main Express application. This allows us to create a modular and flexible application structure, where different parts of the application are responsible for handling different routes and functionalities.

- When we mount middleware or sub-applications in Express, we specify the path at which they should be mounted using the app.use() method. Here's an example:

- In this example, we have two separate router objects, userRouter and productRouter, each defining their own routes and middleware functions. We then mount these routers on specific paths using the app.use() method, so that any incoming requests that match those paths are routed to the appropriate router and middleware.

- So, any incoming requests to the /users path will be handled by the middleware and routes defined in the userRouter object, while requests to the /products path will be handled by the productRouter.

- By using mounting, we can create a more organized and modular application structure, with each part of the application responsible for handling a specific set of routes and functionalities. This can make our code easier to maintain and scale over time, as we can add or remove modules without affecting the rest of the application.



```
const express = require('express')
const app = express()

const userRouter = require('./routes/user')
app.use('/users', userRouter)

const productRouter = require('./routes/product')
app.use('/products', productRouter)
```

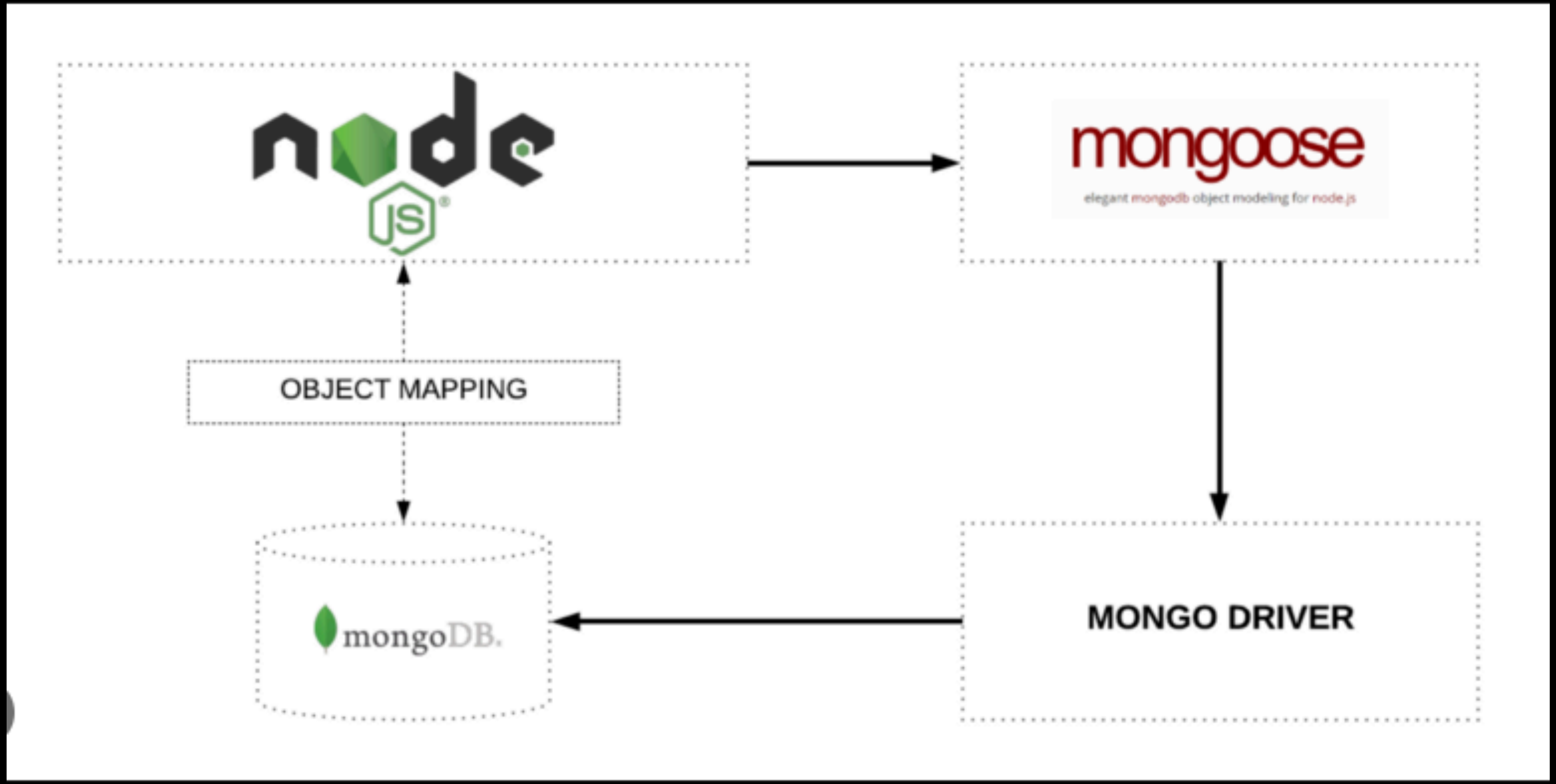
Testing API Routes:

What is MongoDB ?

- MongoDB is a NoSQL document-oriented database that is often used in MERN stack applications. The MERN stack refers to a set of technologies that are commonly used together to build web applications: MongoDB, Express.js, React, and Node.js.

- MongoDB is a popular choice for MERN stack applications because it provides a flexible and scalable database solution that can handle large amounts of data. Unlike traditional SQL databases, MongoDB stores data in JSON-like documents with dynamic schemas, which can make it easier to work with complex data structures.

- In MERN stack applications, MongoDB is typically used as the database layer of the application. This means that data is stored and retrieved from MongoDB using Node.js and the Mongoose library, which provides an object modeling interface for MongoDB.



What is Mongoose ?

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js that provides a higher-level abstraction layer for working with MongoDB. It simplifies the process of interacting with MongoDB databases by providing a schema-based solution to model and validate data.

Mongoose provides a way to define data models in Node.js that are mapped to MongoDB collections. With Mongoose, developers can define data schemas that enforce strict data types and validation rules, allowing for more structured and reliable data storage.

- Here are some of the features that Mongoose provides:
- Schema definition: Developers can define schemas for data models using Mongoose's schema definition syntax, which includes data type validation, custom validation, and other options.
- Data querying and manipulation: Mongoose provides a simple and intuitive way to query and manipulate data in MongoDB using a fluent API that includes methods like find(), findOne(), save(), and update().
- Middleware: Mongoose allows developers to define middleware functions that can be executed before or after specific operations, such as validating data, encrypting passwords, or triggering notifications.
- Connection management: Mongoose provides a connection manager that handles connection pooling and reconnection in case of network failures.
- Plugins: Mongoose allows developers to extend its functionality with plugins that can provide additional features, such as full-text search or geospatial queries.

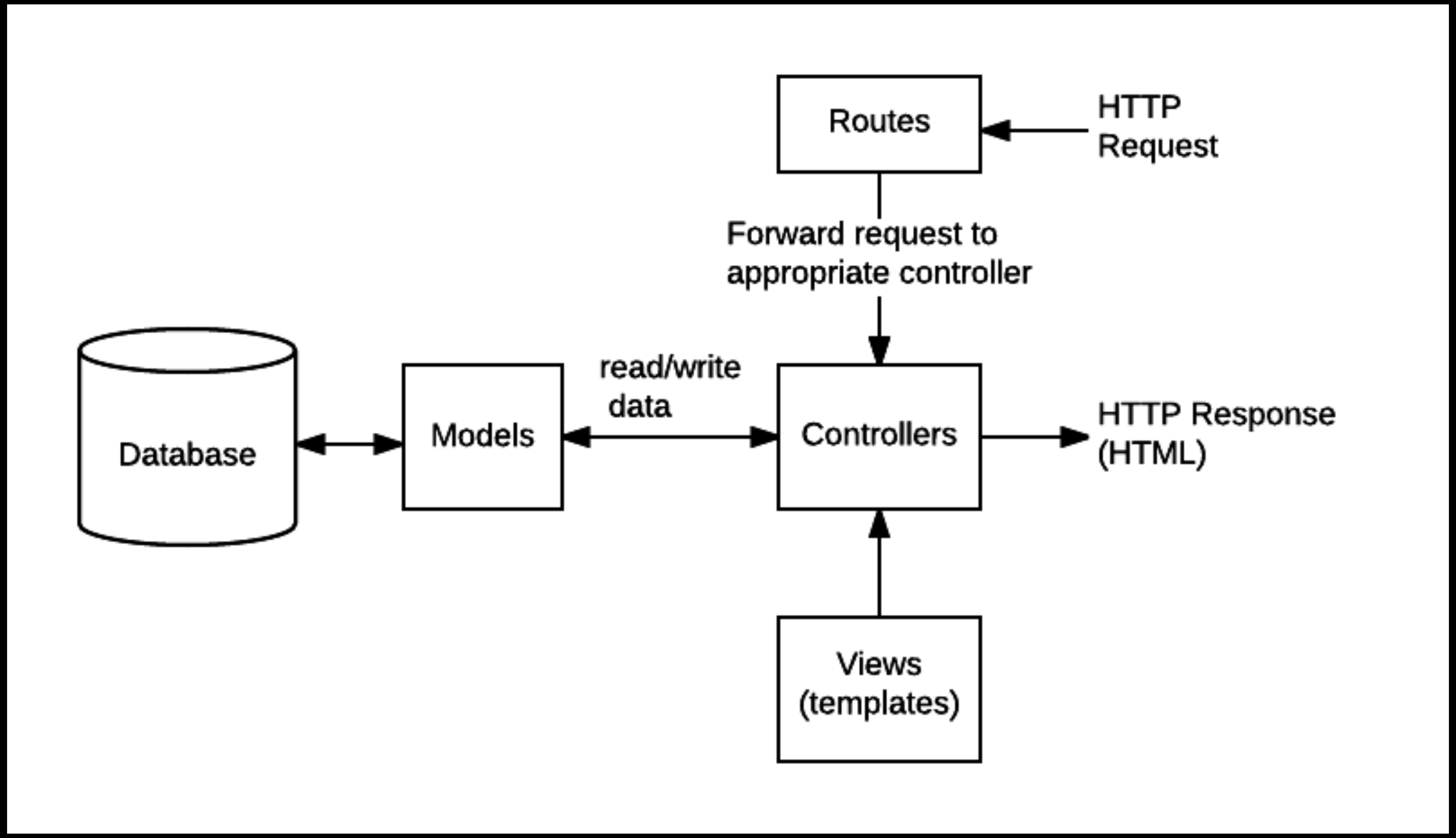
What is ODM ?

- An Object Data Modeling (ODM) library is a software tool or library that provides an abstraction layer on top of a database, allowing developers to work with their data using object-oriented paradigms instead of SQL queries.
- ODM libraries are typically used with NoSQL databases, such as MongoDB, which store data in a format that is more naturally represented as objects rather than in a relational format. By using an ODM library, developers can work with data in a way that more closely resembles their programming language of choice, making it easier to map data to objects, perform validation, and work with relationships between data.
- Some popular ODM libraries for working with NoSQL databases include Mongoose for MongoDB, ODM for Couchbase, and Doctrine for PHP.
- Overall, ODM libraries are a powerful tool for working with NoSQL databases, providing a way to work with data using object-oriented paradigms, which can improve productivity, reduce errors, and improve code maintainability.

What is Nodemon ?

- Nodemon is a tool for Node.js that monitors changes in a project's source code and automatically restarts the Node.js application when a change is detected. This is useful during development, as it allows developers to see the changes they've made without having to manually stop and restart the Node.js application every time.
- Nodemon works by watching the files in a project directory and reloading the application whenever it detects a change. By default, Nodemon will watch for changes in .js, .json, and .html files, but it can be configured to watch other file types as well.
- To use Nodemon, developers need to install it as a development dependency in their Node.js project, and then start the application using the nodemon command instead of the node command.

Optimal Folder Structure



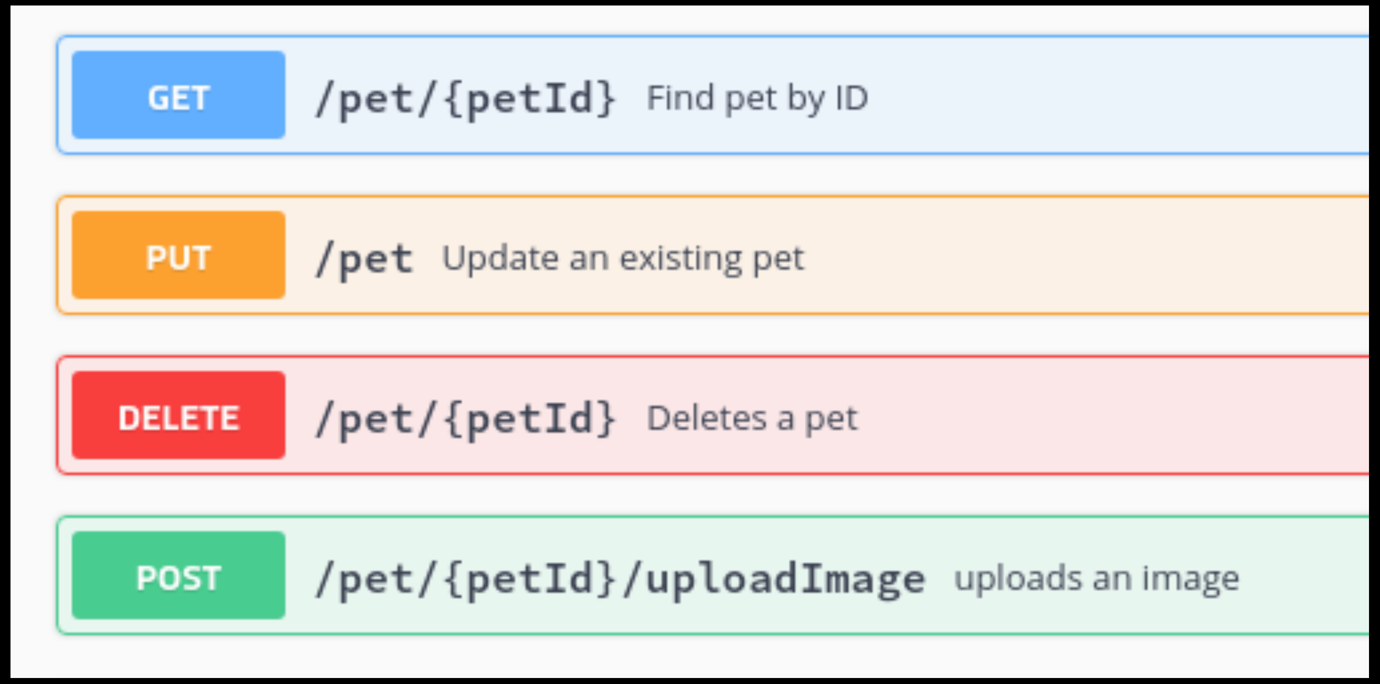
What is MVC ?

MVC stands for Model-View-Controller, which is a software architectural pattern used in web development to separate the application's concerns into three interconnected components.

- Here's a brief overview of each component:
- Model: This component represents the data and business logic of the application. It defines the structure and behavior of the data and manages access to it. In a web application, this might include database operations, data validation, and application-specific business rules.
- View: This component defines the presentation and user interface of the application. It is responsible for rendering data to the user in a way that is easy to understand and interact with. In a web application, this might include HTML templates, CSS stylesheets, and JavaScript code that handles user interactions.
- Controller: This component acts as an intermediary between the model and the view. It handles user requests, interacts with the model to retrieve data, and passes that data to the view for rendering. It also handles user input and updates the model accordingly.
- By separating the application's concerns into these three components, the MVC pattern promotes a modular and maintainable codebase. Changes to one component can be made without affecting the others, which makes it easier to modify and update the application over time.
- MVC is used widely in web development frameworks such as Ruby on Rails, Django, Laravel, and Express.js. By following the MVC pattern, developers can build scalable, maintainable, and testable web applications.

Different types of Routes:

- In the URL "/car/:id", the :id portion is known as a route parameter in Express.js. Route parameters are a way of capturing a variable part of a URL and passing it to the server as a parameter. In this case, the :id parameter represents a unique identifier for a specific car resource.
- When a request is made to a URL that matches this pattern, Express.js captures the value of the id parameter and passes it to the server as a parameter. For example, if a request is made to "/car/123", Express.js would capture the value "123" as the value of the id parameter.
- In the server-side code, developers can access the value of the id parameter using req.params.id, where req is the request object passed to the server-side function. Developers can then use this value to query a database or perform some other action specific to the car resource with the corresponding id.



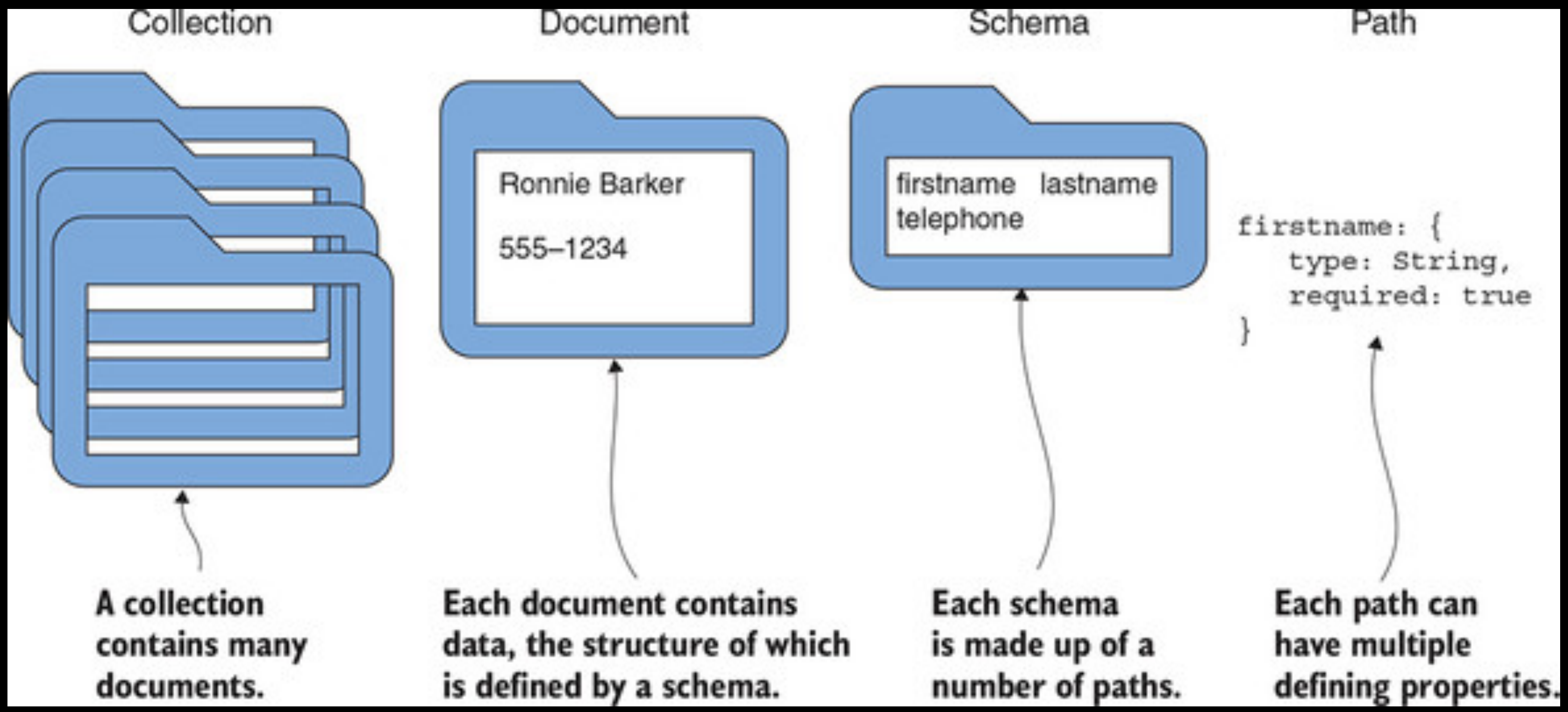
```
GET     /pet/{petId}  Find pet by ID
PUT     /pet  Update an existing pet
DELETE  /pet/{petId}  Deletes a pet
POST    /pet/{petId}/uploadImage  uploads an image
```

How to export ?

- In an Express backend application, there are several ways to export functions or objects:
- Exporting a function or object directly: This is the simplest way to export a function or object from a module. For example, to export a function named myFunction, you can simply write module.exports = myFunction;. To export an object, you can write
  - module.exports = myObject;.
- Exporting multiple functions or objects: If you need to export multiple functions or objects from a module, you can attach them to the exports object. For example, to export two functions named myFunction1 and myFunction2, you can write
  - exports.myFunction1 = myFunction1; exports.myFunction2 = myFunction2;.
  - Similarly, to export two objects named myObject1 and myObject2, you can write
  - exports.myObject1 = myObject1; exports.myObject2 = myObject2;.
- Exporting a default function or object: You can also export a default function or object from a module using the module.exports syntax. For example, to export a default function named myFunction, you can write module.exports = myFunction;. To export a default object, you can write
  - module.exports = myObject;.
- These are some of the common ways to export functions or objects in an Express backend application. The choice of syntax depends on the specific use case and the developer's preference.

Schema, Models & Documents:

- In Mongoose or MongoDB, schema, models, documents, and collections are key concepts used to define and work with data.
- Schema: A schema is a blueprint that defines the structure of a document in MongoDB. It describes the fields or properties that a document can have, along with their data types, default values, and validation rules. Schemas help enforce data consistency and make it easier to work with data.
- Models: A model is a representation of a MongoDB collection. It is created from a schema and provides an interface for interacting with the data in the collection. Models can be used to perform CRUD (create, read, update, delete) operations on documents.
- Documents: A document is an instance of a model and represents a single record in a MongoDB collection. It contains fields with values that correspond to the properties defined in the schema.
- Collection: A collection is a group of related documents in MongoDB. It is similar to a table in a relational database and can be queried and manipulated using MongoDB commands.
- Together, these concepts form the basis of working with data in MongoDB using Mongoose. Schemas define the structure of the data, models provide an interface for working with the data, and documents represent individual records within a collection. Collections group related documents together for efficient querying and manipulation.


Collection
Document
Schema
Path
Ronnie Barker
555–1234
firstname lastname
telephone
A collection contains many documents.
Each document contains data, the structure of which is defined by a schema.
Each schema is made up of a number of paths.
Each path can have multiple defining properties.

```
firstname: {
   type: String,
   required: true
}
```

Controllers:

- In web development backend applications, controllers play a crucial role in implementing the application's business logic and handling user requests. Here are some key reasons why controllers are significant:
- Separation of Concerns: Controllers help to separate the concerns of the application by implementing the business logic separately from other components like models, views, and routes. This makes it easier to manage the codebase and reduces the chances of introducing bugs or errors.
- Handling User Requests: Controllers are responsible for handling user requests and delegating the appropriate actions to other components like models or services. For example, when a user makes a request to create a new resource, the controller validates the request data, creates a new resource using the appropriate model or service, and sends a response to the user.
- Reusability: Controllers can be reused across multiple routes or endpoints, reducing code duplication and promoting code organization. This also makes it easier to test and maintain the codebase.
- Error Handling: Controllers are responsible for handling errors that occur during the execution of a request. They can catch errors, log them, and send appropriate error responses to the user. This helps to ensure the stability and reliability of the application.
- Overall, controllers play a vital role in implementing the application's business logic, handling user requests, and ensuring the stability and reliability of the application.

Mongoose Documentation

Different types of Error/Status Codes:

- Click here

Different types of Interaction with DB:

“Ref” option in Mongoose Schema

In Mongoose, the ref option is used to establish a relationship between two models (collections) in a MongoDB database. It is used inside a Mongoose schema to specify the referenced model for a particular field.

- In this example, the authorSchema has a field called books, which is an array of ObjectId values that reference the Book model. The bookSchema has a field called author, which is an ObjectId value that references the Author model.
- By using the ref option, we can establish the relationship between the two models. This allows us to easily perform operations like querying for all the books written by a particular author, or finding all the authors of a particular book.
- The ref option also allows Mongoose to automatically populate the referenced documents when we query for the parent document. For example, if we query for a book and specify the author field to be populated, Mongoose will automatically fetch the corresponding Author document and populate the author field.
- Overall, the ref option is an important feature of Mongoose that allows us to establish relationships between models and perform related operations efficiently.

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

const authorSchema = new mongoose.Schema({
  name: String,
  age: Number,
  books: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Book' }]
});

const bookSchema = new mongoose.Schema({
  title: String,
  author: { type: mongoose.Schema.Types.ObjectId, ref: 'Author' }
});

const Author = mongoose.model('Author', authorSchema);
const Book = mongoose.model('Book', bookSchema);
```