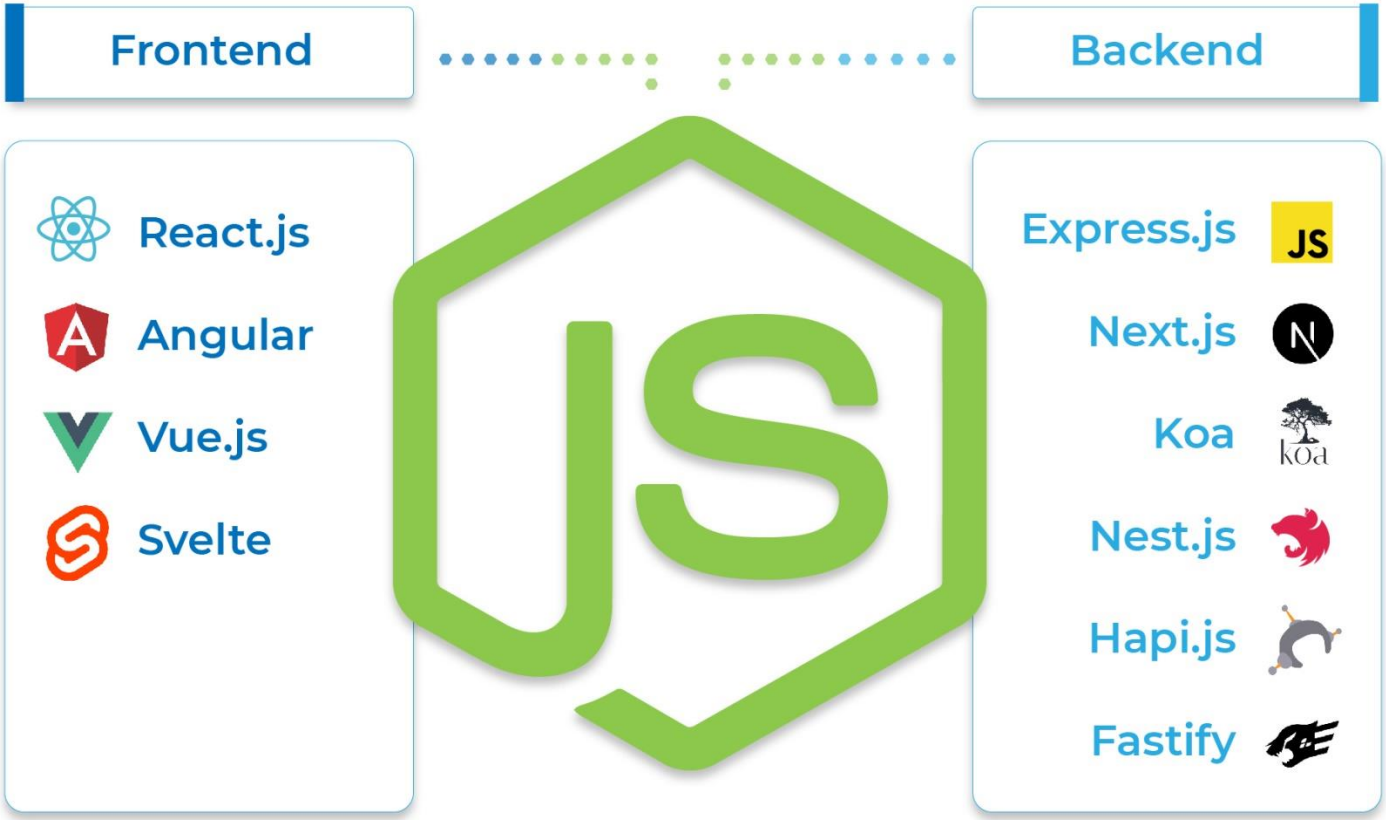
Syllabus:

Express Framework:

- 3.5 Introduction to Express
- 3.6 Installation of Express
- 3.7 Create first Express application
- 3.8 application, request, and response objects
- 3.9 Configuring an Express application
- 3.10 Rendering views
- 3.11 Authentication
- 3.12 Authorization

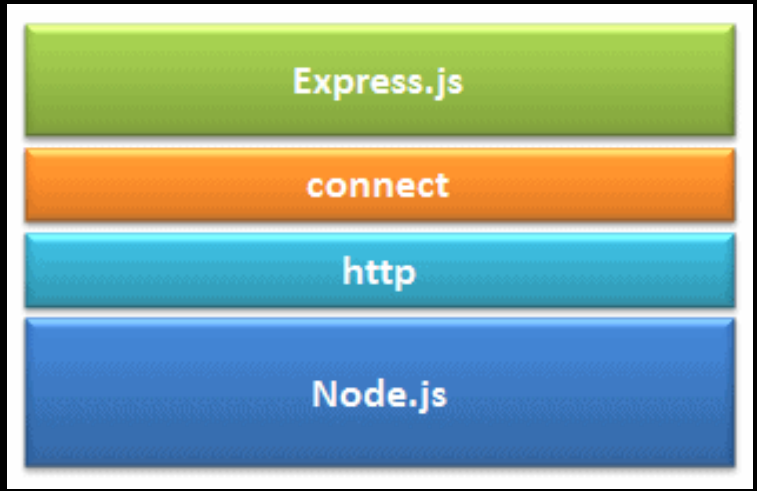
Node.js Frameworks



3.5 Introduction to Express

- **Express.js, or simply Express, is a back end web application framework for Node.js.**
- It is **designed for building web applications, server-side applications and APIs**.
- **Definition: "***Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications***".**
- Express is the **most popular Node web framework**, and is the underlying library for a number of other popular Node web frameworks.
- **Can use other third party middleware** along with express to extend the functionality.

- Express.js is based on the Node.js middleware module called ***connect*** which in turn uses ***http*** module. So, any middleware which is based on connect will also work with Express.js.



Express Features:

- **Asynchronous and single threaded** -> Similar to Node
- Uses **MVC (Model-View-Controller) Architecture**.
- It allows to setup middlewares to respond to HTTP Requests.
- It defines a routing table which is used to perform different actions based on HTTP method and URL.
- Includes various middleware modules (morgan, body-parser, express.static) which you can use to perform additional tasks on request and response.
- Allows you to create **REST API server.**
- Express apps can use many different databases (PostgreSQL, MySQL, Redis, SQLite, and MongoDB), and can perform **C**reate, **R**ead, **U**pdate and **D**elete (CRUD) operations.
- Easy to serve static files and resources of your application.

3.6 Installing Express

- npm install express
- npm install –g express
- npm install express --save

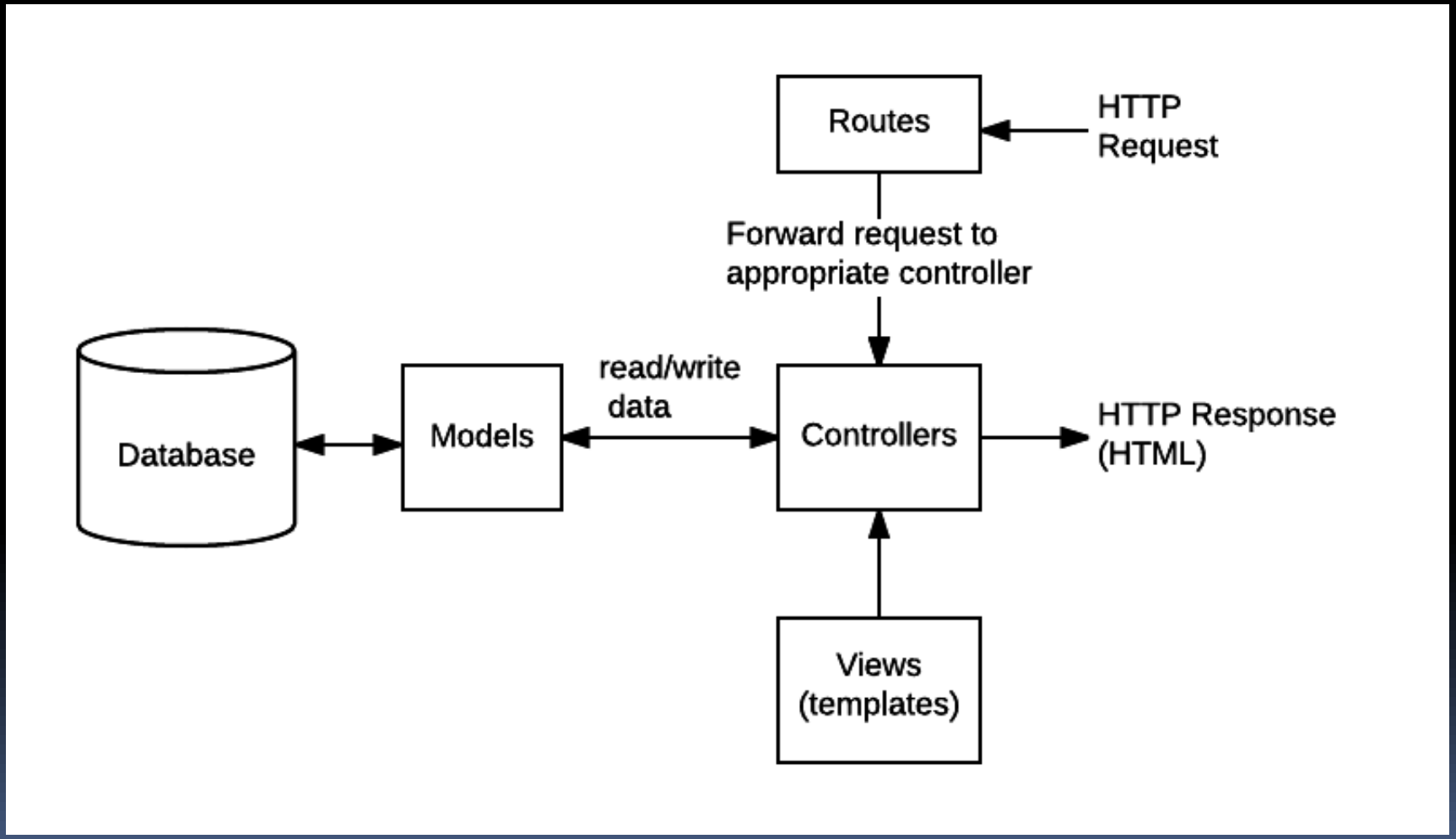
3.7 Creating first express application

- Create a new folder for the express application.
- Create package.json file
  - npm init --yes
- Install express.js and other necessary dependencies using NPM
  - npm install express --save
- Create a file called "Server.js"
- Import express
  - var express = require('express')
- Create an express application object
  - var app = express()

- Implement different middleware functions to handle different HTTP requests
- Test the application
  - node Server.js (or) nodemon Server.js

- **An Express application is essentially a series of middleware function calls.**
- Express middleware are **functions that execute during the lifecycle of a request to the Express server**.
- Middleware functions are functions that **have access to the request object (req), the response object (res), and the next middleware function** in the application's request-response cycle. The next middleware function is commonly denoted by a variable named next.
- Middleware functions can perform the following tasks:
  - **Execute any code.**
  - **Make changes to the request and the response objects.**
  - **End the request-response cycle.**
  - **Call the next middleware function in the stack.**

Handling an HTTP request / response (MVC Architecture)



- **"Routes"** to forward the supported requests (and any information encoded in request URLs) to the appropriate controller functions.
- **"Model"** module that connects to the database and exports some functions that let us operate on the data.
- **"Controller"** functions to get the requested data from the models, create an HTML page displaying the data, and return it to the user to view in the browser.
- **"Views"** (templates) used by the controllers to render the data. It is the template engine that will render the final view, usually in HTML

3.8 The application, request and response objects

- Express presents three major objects that are frequently used:
- 1. application object: It is an instance of an express application and is usually used to configure your application.
- 2. request object: It is a wrapper of Node's HTTP request object and is used to extract information about the currently handled HTTP request.
- 3. response object: It is a wrapper of Node's HTTP response object and is used to set the response data and headers.

application object

- The core part of an Express app is the Application object. The app object conventionally denotes the Express application.
- Create it by calling the top-level express() function exported by the Express module.
  - **var express = require('express')**
  - **var app = express()**
- The app object has methods for
  - **Routing HTTP requests**
  - **Configuring middleware**
  - **Rendering HTML views**

- **app.locals property**:
  - defines the properties that are local variables inside an application.
  - the value of **app.locals** properties persist throughout the life of the application

Methods:

- **app.use([path,] callback [, callback...]) or app.all([path, ,] callback [, callback...]):**
  - Used to create an express middleware to handle HTTP requests sent to the server.
- **app.METHOD(path, callback [, callback ...])**
  - Routes an HTTP request, where METHOD is the HTTP method of the request, such as GET, PUT, POST, and so on, in lowercase.
  - methods are app.get(), app.post(), app.put(), app.delete() and so on.
- **app.listen([port[, host]][, callback]):**
  - Binds and listens for connections on the specified host and port.
  - this method is identical to Node's http.Server.listen().

- **app.set(name, value):**
  - used to assigns the setting name to value.
  - You may store any value that you want, but certain names can be used to configure the behavior of the server.
- **app.get(name):**
  - Returns the value of name previously set
- **app.route(path)**
  - ***Routing*** refers to how an application's endpoints (URIs) respond to client requests.
  - Returns an instance of a single route, which you can then use to handle HTTP verbs/methods with optional middleware.
- **app.engine (ext,callback):**
  - Used to define a given template engine to render certain file types
    - Example: can us EJS template engine to use HTML files as templates
    - **app.engine ('html', require('ejs').renderFile)**

- **app.render(view, [locals], callback)**
  - Returns the rendered HTML of a view via the callback function.

- Differences bewteen app.render() and res.render():
  - app.render() is like res.render(), except it cannot send the rendered view to the client on its own.
  - Internally res.render() uses app.render() to render views.

response object

- The res object represents the HTTP response that an Express app sends when it gets an HTTP request.
- **Methods:**
- **res.send([body])**
  - Sends the HTTP response.
  - Can be a Buffer object, a String, an object, Boolean, or an Array
- **res.end([data])**
  - Ends the response process.
- **res.status(code)**
  - Sets the HTTP status for the response.

- **res.sendStatus(code)**
- **res.set(field [, value]):**
  - Sets the response's HTTP header field to value.
- **res.cookie(name, value [, options]):**
  - This is a method used to set a response cookie. Sets cookie name to value.
- **res.redirect([status,] path):**
  - Redirects to the URL derived from the specified path, with specified status, a positive integer that corresponds to an HTTP status code .
- **res.render(view [, locals] [, callback]):**
  - used to render a view and sends the rendered HTML string to the client.

request object

- The req object **represents the HTTP request** and has properties for the request query string, parameters, body, HTTP headers, and so on.
- **Properties:**
- **req.query** –
  - This is a property that contains the query string parameter in the route.
- **req.params -**
  - This property is an object containing properties mapped to the named route "parameters". For example, if you have the route /user/:name, then the "name" property is available as req.params.name.

- **req.body** –
  - Contains key-value pairs of data submitted in the request body.
  - By default, it is undefined, and is populated when you use **body-parsing** middleware such as express.json() or express.urlencoded().
- **req.cookies –**
  - When using cookie-parser middleware, this property is an object that contains cookies sent by the request.
- **req.path / req.hostname / req.ip** -
  - These are used to retrieve the current request path, hostname and remote IP.

- **Methods:**
  - **req.param(name)**
    - returns the value of param name when present. When parameters are passed, then it can be accessed using this function.

3.9 Configuring an express application

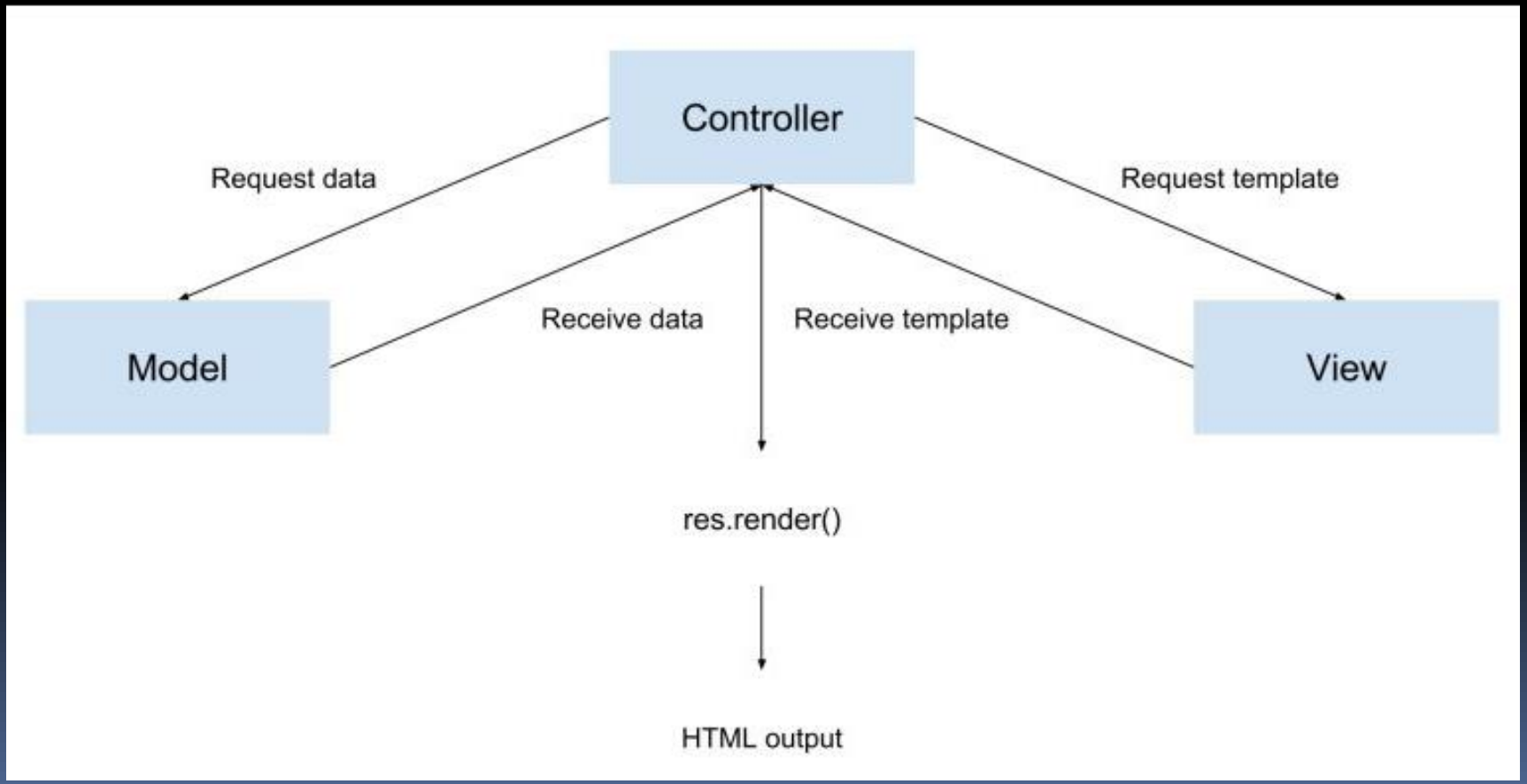
- Express configuration system enables you to add certain functionality to your Express application.
- Although there are predefined configuration options-
  - can change the way it works.
  - can also add your own key/value configuration options for any other usage.
- Another robust feature of Express is the "ability to configure your application based on the environment it's running on" – can give different configurations to different environments
  - **Environments: development, production**

- Node.js assumes it's always running in a **development environment**. We can **change the environment by using NODE_ENV.**
  - In the CLI use the command to change value of NODE_ENV
    - **SET NODE_ENV='production'**
- To modify other environment variables, you will need:
  - Install **"dotenv"** - a lightweight npm package that automatically *imports environment variables from a '**.env** 'file into the **process.env** object*.
  - Create **".env"** file and
  - Use "***process.env***" property to access them.
- The **process.env** is a global variable that allows you to access environment variables
- Commonly used: **NODE_ENV, PORT**

3.10 Rendering Views

- A very common feature of web frameworks is the ability to render views.
- Basic concept:
  - Passing your data to a template engine that will render the final view, usually in HTML.
- Template engine:
  - A ***template engine*** enables you to use static template files in your application.
  - At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client.
  - Some popular template engines that work with Express are Pug, Mustache, and **EJS (Embedded Java Script)**.

- In the MVC pattern:
  - controller uses the model to retrieve the data portion and
  - view template to render the HTML output, as described in the next diagram.



Using EJS template engine to render views

- **Install the ejs package**
  - npm install ejs
- **Set the 'view engine' to 'ejs'**
  - app.set('view engine', 'ejs');
- **Inside views folder create ejs template files**
  - EJS views basically consists of HTML code mixed with EJS tags.
  - EJS templates will reside in the **app/views** folder and will have **.ejs** extension
- Use **res.render()** method to render the view and sends the rendered HTML document to the client.

- **Serving static files in Express:**
  - To serve static files **such as images, CSS files, and JavaScript files**, use the **express.static** built-in middleware function in Express.
  - For example, use the following code to serve images, CSS files, and JavaScript files in a directory named public:
    - **app.use(express.static('public'))**

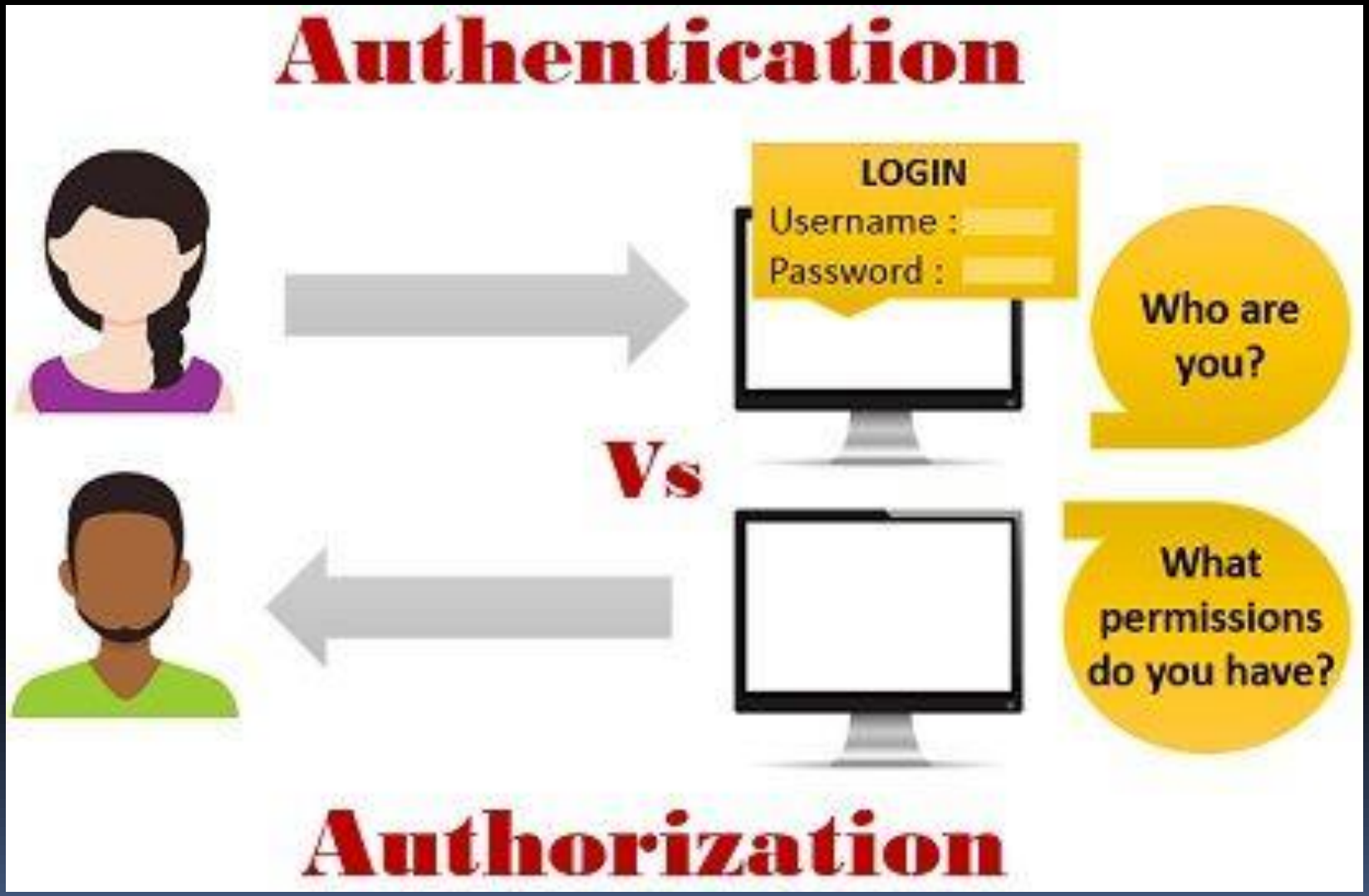
3.11 Authentication

- **"process of verifying who a user is"**
- The identity of users are checked
- Authentication is the process of **verifying a user's identification through** the acquisition of **credentials** and using those credentials to confirm the user's identity.
- Example:
  - Login form – credentials(username and password) entered by the user are compared to those on file in a database of authorized users' information
  - If the credentials match, the authentication process is completed and the user is granted authorization for access.

3.12 Authorization

- "process of verifying what they have access to"
- Usually done after successful authentication.
- Authorization is the process of allowing authenticated users access to resources by determining whether they have permission to access a resource.
- **verifying what specific applications, information, files, and databases a user has access to.**
- access to a resource is protected by both authentication and authorization.
  - If you can't prove your identity, you won't be allowed into a resource.
  - And even if you can prove your identity, if you are not authorized for that resource, you will still be denied access.

Authentication Vs Authorization

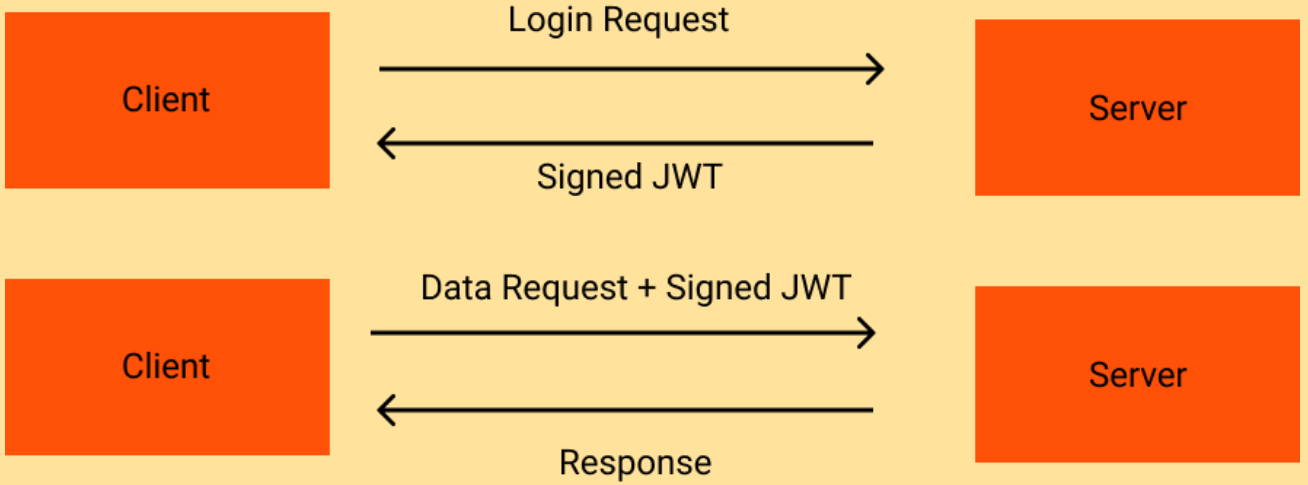


Authentication	Authorization
Determines whether users are who they claim to be Verifies user's credentials	Determines what users can and cannot access Validate user's permissions
Challenges the user to validate credentials (for example, through passwords, answers to security questions, or facial recognition)	Verifies whether access is allowed to a resource through policies and rules
Usually done before authorization	Usually done after successful authentication
Generally, transmits info through an ID Token	Generally, transmits info through an Access Token
Example: Employees in a company are required to authenticate through the network before accessing their company email	**Example:** After an employee successfully authenticates, the system determines what information the employees are allowed to access

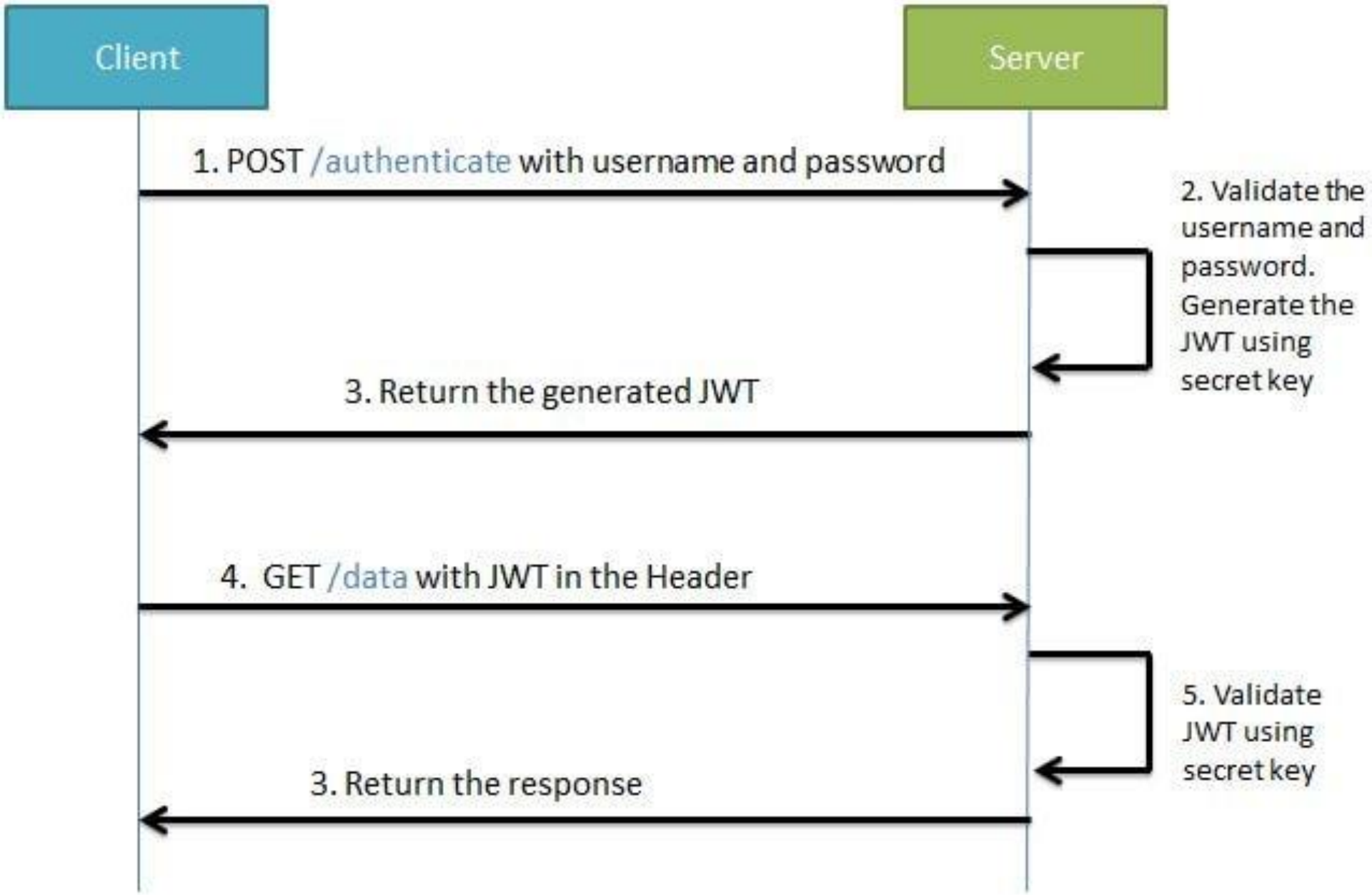
Authentication and Authorization in Express API using JWT

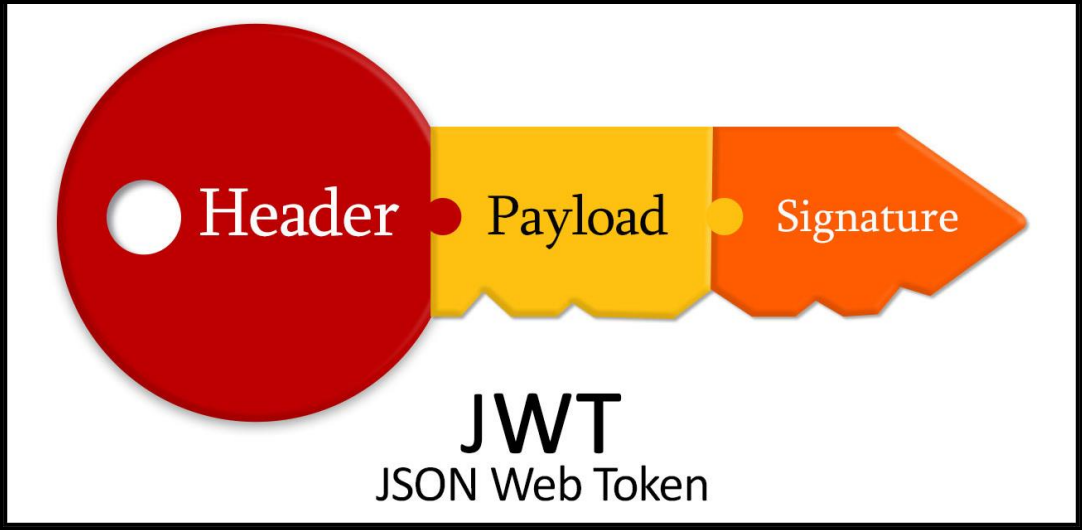
- **JSON Web Tokens** (JWT) have been introduced as a method of secure communication between two parties.
- JWT is very popular for handling authentication and authorization via HTTP.
- "JSON Web Token (JWT) is an **open standard (RFC 7519)** that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. "
- This information can be verified and trusted because it is digitally signed.
- JWTs can be **signed using a secret** (with the **HMAC** algorithm) **or a public/private key pair** (**using RSA or ECDSA**).

how JWT works?



1. Client sends a login request with login credentials (username, password)
2. On the server side the given login credentials are verified (Authentication).
3. When the user is authenticated using his credentials, server generates a signed JWT web token with user info and sends it back to the client.
4. Whenever the user wants to access a protected route or resource, the user agent should send the JWT, **typically in the Authorization header using the Bearer schema.**
5. On the server side we check if the provided JWT is valid, then check if the user is allowed to access the resource that was requested (authorization).If authorized, can access the resource, otherwise we send an error message.





Structure of JWT:

- JSON Web Tokens consist of three parts separated by dots (.), which are:
  - **Header**
  - **Payload**
  - **Signature**
- Therefore, a JWT typically looks like the following.
  - **xxxxx.yyyyy.zzzzz**

1 2

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o 3

1 Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

2 Payload

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

3 Signature

```
HMACSHA256(
BASE64URL(header)
.
BASE64URL(payload),
secret)
```

- **Header :**
  - Consists of two parts:
    1. the type of the token, which is JWT, and
    2. hashing algorithm, which was used to generate the sign such as HMAC SHA256 or RSA.
- **Payload:**
  - Contains "Claims".
  - Claims are statements about an entity (typically, the user) and additional data **- contains the JSON object of user data**
  - Represents the data to be securely exchanged
- **Signature:**
  - It is the hash of the header and payload generated using hash algorithm specified in the header and a secret key.

Methods used:

- **jwt.sign(payload, secretOrPrivateKey, [options, callback])** - Create and returns JWT token
  - Payload - object literal, buffer or string representing valid JSON.
  - secretOrPrivateKey - secret key for HMAC algorithms or the private key for RSA and ECDSA.
  - options:
    - algorithm (default: HS256), expiresIn , subject, ….
- **jwt.verify(token, secretOrPublicKey, [options, callback])** - verifies token and returns payload
  - token - JsonWebToken string

Procedure for creating JWT:

- **Create a directory**
- **Create package.json file**
  - **npm init --yes**
- **Install necessary dependencies**
  - express
  - Jsonwebtoken: used to generate and verify JWT tokens.
  - **npm install express jsonwebtoken --save**
- **Create a file called index.js:**
  - Create a JWT token using jwt.sign() method:
    - **jwt.sign(payload, secretOrPrivateKey, [options, callback])** - returns JWT token
  - Send the JWT token to the client using res.json() method
    - **res.json({JWT Token});**

Verifying JWT Token:

- Whenever the user wants to access a protected route or resource, the user agent(browser) should send the **JWT**, typically in the **Authorization** header.
- Client browser adds JWT Token to the **Authorization** header of the request using the **Bearer**.
- Format of token:
  - **Authorization : Bearer <access-token>**
- Use **jwt.verify()** to verify the token
  - **jwt.verify(token, secretOrPublicKey, [options, callback])**
- Returns the **payload** decoded

Advantages of JWT:

- **JWTs are digitally signed:**
  - Since JWTs are cryptographically signed (HMAC /RSA/ECDSA), they require a cryptographic algorithm to verify. Information is verified and trusted.
- **Self Content :**
  - It contains the details of user, which has authentication information, expire time information, and other user defined claims digitally signed. So no need to query database to get user details.
- **Time Based Tokens:**
  - JWTs expire at specific intervals making them more secure.
- **No Session to Manage (stateless):**
  - JWT tokens can be saved in cookies or local storage, instead of the traditional approach of creating a session in the server.
- **Portable:**
  - A single token can be used with multiple backends.