Summer Training Report

On

Backend Development

Submitted to the Department of Electrical Engineering in Partial

Fulfilment for the

Requirements for the Degree of

Bachelor of Technology

(Electrical)

by

Anuj Chhiroliya

(U19EE061)

(B. TECH. IV(EE), VII*th* Semester)

External Guide

Saddam Hussain

Mentor, E-Rental

Internal Guide

C.P. Gor

Assistant Professor / Associate Professor, DoEE

DEPARTMENT OF ELECTRICAL ENGINEERING SARDAR

VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY

AUG-2022

Sardar Vallabhbhai National Institute Of Technology

Surat - 395 007, Gujarat, India

DEPARTMENT OF ELECTRICAL ENGINEERING

CERTIFICATE

This is to certify that the Summer Training Report entitled “**Backend Developer**” is presented & submitted by Candidate Anuj Chhiroliya, bearing Roll No. U19EE061, of B.Tech. IV, 7th Semesterin the partial fulfillment of the requirement for the award of B.Tech. Degree in Electrical Engineering for academic year 2022 - 23.

He has successfully and satisfactorily completed his/her Summer Training Presentation Exam in all respects. We certify that the work is comprehensive, complete andfit for evaluation.

CP Gor

Assistant Professor / Associate Professor & Summer Training Guide

Name of Examiners	**Signature with Date**
1. ____________________	____________________
2. ____________________	____________________

Dr. A.K. Panchal

Head Of Department

DoEE, SVNIT

Seal of The Department

(September 2022)

Acknowledgements

I would like to express my profound gratitude and deep regards to my guide C.P Gor for his guidance. I am heartily thankful for suggestion and the clarity of the concepts of the topic that helped me a lot for this work. I would also like to thank Prof. AK Panchal, Head of the Electrical Engineering Department, SVNIT and all the faculties of EED for their co-operation and suggestions. I am very much grateful to all my classmates for their support.

Anuj Chhiroliya
U19EE061
Sardar Vallabhbhai
National Institute of Technology
Surat

September 17, 2022

Abstract

Summer training is an important part of a student's life as it helps the student to take the first step towards his/her career. A student gains first-hand knowledge about how a professional environment works. It develops an awareness of the industrial approach to problem-solving, based on a broad understanding of the process and mode of operation of the organization. Through summer training a student is able to develop the following skills

- Work under pressure
- Take the lead on assigned projects
- Make decisions and complete goals
- Collaborate compromise to facilitate group decisions

I completed my internship at E-Rental Pvt. Ltd., Mumbai. During my training period of eight weeks, I learned about SpringBoot, Maven tool, API's & JWT token. I learned about their various components, working, implementations, etc. The following report presents my learning from my internship.

Table of contents

Table of Figure

1. SpringBoot Framework :

Spring Boot is an open source, microservice-based Java web framework. The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its codebase. The microservice architecture provides developers with a fully enclosed application, including embedded application servers.

Features

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks, and externalized configuration
- Absolutely no code generation and no requirement for XML configuration

2. Maven

Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven provides developers ways to manage the following

- Builds
- Documentation
- Reporting
- Dependencies
- SCMs
- Releases
- Distribution
- Mailing list

3. OOPS (Object-Oriented Programming)

Object-Oriented Programming is considered as a design methodology for building non-rigid software. In OOPS, every logic is written to get our work done but represented in the form of Objects. OOP allows us to break our problems into a small unit of work that is represented via objects and their functions. We build functions around objects.

There are mainly six pillars (features) of OOP. If all of these four features are presented in programming, the programming is called perfect Object-Oriented Programming.

- Object
- Class
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Let's consider an example of explaining each of these pillars so you can better understand Object-Oriented Programming.

Objects

Any real-world entity which can have some characteristics of which can perform some tasks is called Object. This object is also called an instance i.e. a copy of an entity in a programming language. If we consider the above example, a mobile manufacturing company can be an object. Each object can be different based on their characteristics. For example, here are two objects.

Mobile mbl1 = new Mobile ();

Mobile mbl2 = new Mobile ();

Class

A class in OOP is a plan which describes the object. We call it a blueprint of how the object should be represented. Mainly a class would consist of a name, attributes, and operations. Considering the above example, Mobile can be a class, which has some attributes like Profile Type, IMEI Number, Processor, and some more. It can have operations like Dial, Receive and SendMessage.

There is some OOPS principle that needs to be satisfied while creating a class. This principle is called as SOLID where each letter has some specification and we discuss this in next section of report.

Abstraction

Abstraction allows us to expose limited data and functionality of objects publicly and hide the actual implementation. It is the most important pillar in OOPS. In our example of Mobile class and objects like Nokia, Samsung, iPhone.

Some features of mobiles,

1. Dialling a number call some method internally which concatenate the numbers and displays it on screen but what is it doing we don’t know.
2. Clicking on green button actual send signals to calling a person’s mobile but we are unaware of how it is doing.

This is called abstraction. In classes, we can create methods that can be called and used by the users of the class but users will have no idea what these methods do internally.

Encapsulation

Encapsulation is defined as the process of enclosing one or more details from the outside world through access right. It says how much access should be given to particular details. Both Abstraction & Encapsulation works hand in hand because Abstraction says what details to be made visible and Encapsulation provides the level of access right to that visible details. i.e. — It implements the desired level of abstraction.

Talking about Bluetooth which we usually have in our mobile. When we switch on a Bluetooth, I am able to connect to another mobile or Bluetooth enabled devices but I’m not able to access the other mobile features like dialling a number, accessing inbox etc. This is because the Bluetooth feature is given some level of abstraction.

Another point is when mobile A is connected with mobile B via Bluetooth whereas mobile B is already connected to mobile C then A is not allowed to connect C via B. This is because of accessibility restriction.

Polymorphism

Polymorphism can be defined as the ability to use the same name for doing different things. More precisely we say it as ‘many forms of single entity’. This play a vital role in the concept of OOPS.

Let’s say Samsung mobile has a 5MP camera available i.e. — it is having a functionality of CameraClick(). Now the same mobile is having Panorama mode available in-camera, so functionality would be the same but with mode. type is said to be Static polymorphism or Compile time polymorphism.

Compile-time polymorphism the compiler knows which overloaded method it is going to call.

The compiler checks the type and number of parameters passed to the method and decides which method to call and it will give an error if there are no methods that match the method signature of the method that is called at compile time.

Another point wherein Send Message was intended to send message to a single person at a time but suppose Nokia had given provision for sending a message to a group at once. i.e. — Overriding the functionality to send message to a group. This type is called Dynamic polymorphism or Runtime polymorphism.

For overriding you need to set the method, which can be overridden to virtual & its new implementation should be decorated with the override keyword.

By runtime polymorphism, we can point to any derived class from the object of the base class at runtime that shows the ability of runtime binding.

Inheritance

Inheritance is the ability to extend the functionality from the base entity in new entity belonging to the same group. This will help us to reuse the functionality which is already defined before and extend into a new entity.

Considering the example, the above figure 1.1 itself shows what is inheritance. Basic Mobile functionality is to send a message, dial and receive a call. So the brands of mobile are using this basic functionality by extending the mobile class functionality and adding their own new features to their respective brand.

There are mainly 4 types of inheritance,

1. Single level inheritance
2. Multi-level inheritance
3. Hierarchical inheritance
4. Hybrid inheritance

Single level inheritance

In Single level inheritance, there is a single base class & a single derived class i.e. - A base mobile features is extended by Samsung brand.

Multilevel inheritance

In Multilevel inheritance, there is more than one single level of derivation. i.e. - After base features are extended by Samsung brand. Now Samsung brand has manufactured its new model with newly added features or advanced OS like Android OS, v4.4.2 (KitKat). From generalization, getting into more specification.

Hierarchal inheritance

In this type of inheritance, multiple derived class would be extended from a base class, it's similar to single level inheritance but this time along with Samsung, Nokia is also taking part in inheritance.

Hybrid inheritance

Single, Multilevel, & hierarchal inheritance all together construct a hybrid inheritance.

Samsung will use the function of multiple Phone (Mobile & Telephone). This would create confusion for complier to understand which function to be called when an event in mobile is triggered like Dial () where Dial is available in both the Phone i.e. — (Mobile & Telephone). To avoid this confusion C# came with the concept of the interface which is different from multiple inheritances actually.

If we take an interface it is similar to a class but without implementation & only declaration of properties, methods, delegates & events. The interface actually enforces the class to have a standard contract to provide all implementation to the interface members. Then what's is the use of interface when they do not have any implementation? The answer is, they are helpful for having readymade contracts, only we need to implement functionality over this contract.

I mean to say, Dial would remain Dial-in case of Mobile or Telephone. It won't be fair if we give a different name when its task is to Call the person.

The interface is defined with the keyword 'interface'.All properties & methods within the interface should be implemented if it is been used. That's the rule of the interface.

4. SOLID principles :-

SOLID principles are object-oriented design concepts relevant to software development.

Background:-

The SOLID principles were first introduced by the famous Computer Scientist Robert J. Martin (a.k.a Uncle Bob) in his paper in 2000. But the SOLID acronym was introduced later by Michael Feathers.

Uncle Bob is also the author of bestselling books *Clean Code* and *Clean* Architecture, and is one of the participants of the "Agile Alliance".

Therefore, it is not a surprise that all these concepts of clean coding, object-oriented architecture, and design patterns are somehow connected and complementary to each other.

1. Single responsibility principle

Every class in Java should have a single job to do. To be precise, there should only be one reason to change a class. Here's an example of a Java class that does not follow the single responsibility principle (SRP):

```java
public class Vehicle {
    public void printDetails() {}
    public double calculateValue() {}
    public void addVehicleToDB() {}
}
```

The `Vehicle` class has three separate responsibilities: reporting, calculation, and database. By applying SRP, we can separate the above class into three classes with separate responsibilities.

2. Open-closed principle

Software entities (e.g., classes, modules, functions) should be open for an extension, but closed for modification.

Consider the below method of the class VehicleCalculations:

```java
public class VehicleCalculations {
    public double calculateValue(Vehicle v) {
        if (v instanceof Car) {
            return v.getValue() * 0.8;
        if (v instanceof Bike) {
            return v.getValue() * 0.5;
    }
}
```

Suppose we now want to add another subclass called Truck. We would have to modify the above class by adding another if statement, which goes against the Open-Closed Principle.
A better approach would be for the subclasses Car and Truck to override the calculateValue method:

```
public class Vehicle {
    public double calculateValue() {...}
}
public class Car extends Vehicle {
    public double calculateValue() {
        return this.getValue() * 0.8;
}
public class Truck extends Vehicle{
    public double calculateValue() {
        return this.getValue() * 0.9;
}
```

Adding another Vehicle type is as simple as making another subclass and extending from the Vehicle class.

3. Liskov substitution principle

The **Liskov Substitution Principle (LSP)** applies to inheritance hierarchies such that derived classes must be completely substitutable for their base classes.

Consider a typical example of a Square derived class and Rectangle base class:

```
public class Rectangle {
    private double height;
    private double width;
    public void setHeight(double h) { height = h; }
    public void setWidht(double w) { width = w; }
    ...
}
public class Square extends Rectangle {
    public void setHeight(double h) {
        super.setHeight(h);
        super.setWidth(h);
    }
    public void setWidth(double w) {
        super.setHeight(w);
        super.setWidth(w);
    }
}
```

The above classes do not obey LSP because you cannot replace the Rectangle base class with its derived class Square. The Square class has extra constraints, i.e., the height and width must be the same. Therefore,
substituting Rectangle with Square class may result in unexpected behavior.

4. Interface segregation principle

The Interface Segregation Principle (ISP) states that clients should not be forced to depend upon interface members they do not use. In other words, do not force any client to implement an interface that is irrelevant to them.

Suppose there's an interface for vehicle and a Bike class:

```
public interface Vehicle {
    public void drive();
    public void stop();
    public void refuel();
    public void openDoors();
}
public class Bike implements Vehicle {

    // Can be implemented
    public void drive() {...}
    public void stop() {...}
    public void refuel() {...}

    // Can not be implemented
    public void openDoors() {...}
}
```

As you can see, it does not make sense for a Bike class to implement
the openDoors() method as a bike does not have any doors! To fix this, ISP proposes that the interfaces be broken down into multiple, small cohesive interfaces so that no class is forced to implement any interface, and therefore methods, that it does not need.

5. Dependency inversion principle

The Dependency Inversion Principle (DIP) states that we should depend on abstractions (interfaces and abstract classes) instead of concrete implementations (classes). The abstractions should not depend on details; instead, the details should depend on abstractions.

Consider the example below. We have a Car class that depends on the
concrete Engine class; therefore, it is not obeying DIP.

```
public class Car {
    private Engine engine;
    public Car(Engine e) {
        engine = e;
    }
    public void start() {
        engine.start();
    }
}
public class Engine {
   public void start() {...}
}
```

The code will work, for now, but what if we wanted to add another engine type, let's say a diesel engine? This will require refactoring the Car class.
However, we can solve this by introducing a layer of abstraction. Instead of Car depending directly on Engine, let's add an interface:

```
public interface Engine {
    public void start();
}
```

Now we can connect any type of Engine that implements the Engine interface to the Car class:

```
public class Car {
    private Engine engine;
    public Car(Engine e) {
        engine = e;
    }
    public void start() {
        engine.start();
    }
}
public class PetrolEngine implements Engine {
   public void start() {...}
}
public class DieselEngine implements Engine {
   public void start() {...}
}
```

5. Application Programming Interface (API)

APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols. For example, the weather bureau's software system contains daily weather data. The weather app on your phone "talks" to this system via APIs and shows you daily weather updates on your phone.

How an API works

An API is a set of defined rules that explain how computers or applications communicate with one another. APIs sit between an application and the web server, acting as an intermediary layer that processes data transfer between systems.

Here's how an API works:

1. A client application initiates an API call to retrieve information—also known as a request. This request is processed from an application to the web server via the API's Uniform Resource Identifier (URI) and includes a request verb, headers, and sometimes, a request body.
2. After receiving a valid request, the API makes a call to the external program or web server.
3. The server sends a response to the API with the requested information.
4. The API transfers the data to the initial requesting application.

While the data transfer will differ depending on the web service being used, this process of requests and response all happens through an API. Whereas a user interface is designed for use by humans, APIs are designed for use by a computer or application.

APIs offer security by design because their position as middleman facilitates the abstraction of functionality between two systems—the API endpoint decouples the consuming application from the infrastructure providing the service. API calls usually include authorization credentials to reduce the risk of attacks on the server, and an API gateway can limit access to minimize security threats. Also, during the exchange, HTTP headers, cookies, or query string parameters provide additional security layers to the data.

For example, consider an API offered by a payment processing service. Customers can enter their card details on the frontend of an application for an ecommerce store. The payment processor doesn't require access to the user's bank account; the API creates a unique token for this transaction and includes it in the API call to the server. This ensures a higher level of security against potential hacking threats.

Why we need APIs

Whether you're managing existing tools or designing new ones, you can use an application programming interface to simplify the process. Some of the main benefits of APIs include the following:

- Improved collaboration: The average enterprise uses almost 1,200 cloud applications (link resides outside of IBM), many of which are disconnected. APIs enable integration so that these platforms and apps can seamlessly communicate with one another. Through this integration, companies can automate workflows and improve workplace collaboration. Without APIs, many enterprises would lack connectivity and would suffer from informational silos that compromise productivity and performance.
- Easier innovation: APIs offer flexibility, allowing companies to make connections with new business partners, offer new services to their existing market, and, ultimately, access new markets that can generate massive returns and drive digital transformation. For example, the company Stripe began as an API with just seven lines of code. The company has since partnered with many of the biggest enterprises in the world, diversified to offer loans and corporate cards, and was recently valued at USD 36 billion (link resides outside of IBM).
- Data monetization: Many companies choose to offer APIs for free, at least initially, so that they can build an audience of developers around their brand and forge relationships with potential business partners. However, if the API grants access to valuable digital assets, you can monetize it by selling access (this is referred to as the API economy). When AccuWeather (link resides outside of IBM) launched its self-service developer portal to sell a wide range of API packages, it took just 10 months to attract 24,000 developers, selling 11,000 API keys and building a thriving community in the process.
- Added security: As noted above, APIs create an added layer of protection between your data and a server. Developers can further strengthen API security by using tokens, signatures, and Transport Layer Security (TLS) encryption; by implementing API gateways to manage and authenticate traffic; and by practicing effective API management.

Common API examples

Because APIs allow companies to open up access to their resources while maintaining security and control, they have become a valuable aspect of modern business. Here are some popular examples of application programming interfaces you may encounter:

- Universal logins: A popular API example is the function that enables people to log in to websites by using their Facebook, Twitter, or Google profile login details. This convenient feature allows any website to leverage an API from one of the more popular services to quickly authenticate the user, saving them the time and hassle of setting up a new profile for every website service or new membership.
- Third-party payment processing: For example, the now-ubiquitous "Pay with PayPal" function you see on ecommerce websites works through an API. This allows people to pay for products online without exposing any sensitive data or granting access to unauthorized individuals.
- Travel booking comparisons: Travel booking sites aggregate thousands of flights, showcasing the cheapest options for every date and destination. This service is made possible through APIs that provide application users with access to the latest information about availability from hotels and airlines. With an autonomous

exchange of data and requests, APIs dramatically reduce the time and effort involved in checking for available flights or accommodation.

- Google Maps: One of the most common examples of a good API is the Google Maps service. In addition to the core APIs that display static or interactive maps, the app utilizes other APIs and features to provide users with directions or points of interest. Through geolocation and multiple data layers, you can communicate with the Maps API when plotting travel routes or tracking items on the move, such as a delivery vehicle.
- Twitter: Each Tweet contains descriptive core attributes, including an author, a unique ID, a message, a timestamp when it was posted, and geolocation metadata. Twitter makes public Tweets and replies available to developers and allows developers to post Tweets via the company's API.

Types of API protocols

- As the use of web APIs has increased, certain protocols have been developed to provide users with a set of defined rules that specifies the accepted data types and commands. In effect, these API protocols facilitate standardized information exchange:
- SOAP (Simple Object Access Protocol) is an API protocol built with XML, enabling users to send and receive data through SMTP and HTTP. With SOAP APIs, it is easier to share information between apps or software components that are running in different environments or written in different languages.
- REST (Representational State Transfer) is a set of web API architecture principles, which means there are no official standards (unlike those with a protocol). To be a REST API (also known as a RESTful API), the interface must adhere to certain architectural constraints. It’s possible to build RESTful APIs with SOAP protocols, but the two standards are usually viewed as competing specifications.
- XML-RPC is a protocol that relies on a specific format of XML to transfer data, whereas SOAP uses a proprietary XML format. XML-RPC is older than SOAP, but much simpler, and relatively lightweight in that it uses minimum bandwidth.
- JSON-RPC is a protocol similar to XML-RPC, as they are both remote procedure calls (RPCs), but this one uses JSON instead of XML format to transfer data. Both protocols are simple. While calls may contain multiple parameters, they only expect one result.

REST: Representational State Transfer

REST is a set of architectural principles attuned to the needs of lightweight web services and mobile applications. Because it's a set of guidelines, it leaves the implementation of these recommendations to developers.

When a request for data is sent to a REST API, it’s usually done through hypertext transfer protocol (commonly referred to as HTTP). Once a request is received, APIs designed for REST (called RESTful APIs or RESTful web services) can return messages in a variety of formats: HTML, XML, plain text, and JSON. JSON (JavaScript object notation) is favored as a message format because it can be read by any programming language (despite the name), is human- and machine-readable, and is lightweight. In this way, RESTful APIs are more flexible and can be easier to set up.

An application is said to be RESTful if it follows 6 architectural guidelines. A RESTful application must have:

1. A client-server architecture composed of clients, servers, and resources.
2. Stateless client-server communication, meaning no client content is stored on the server between requests. Information about the session’s state is instead held with the client.
3. Cacheable data to eliminate the need for some client-server interactions.
4. A uniform interface between components so that information is transferred in a standardized form instead of specific to an application’s needs. This is described by Roy Fielding, the originator of REST, as “the central feature that distinguishes the REST architectural style from other network-based styles.”
5. A layered system constraint, where client-server interactions can be mediated by hierarchical layers.
6. Code on demand, allowing servers to extend the functionality of a client by transferring executable code (though also reducing visibility, making this an optional guideline).

SOAP: Simple Object Access Protocol

SOAP is a standard protocol that was first designed so that applications built with different languages and on different platforms could communicate. Because it is a protocol, it imposes built-in rules that increase its complexity and overhead, which can lead to longer page load times. However, these standards also offer built-in compliances that can make it preferable for enterprise scenarios. The built-in compliance standards include security, atomicity, consistency, isolation, and durability (ACID), which is a set of properties for ensuring reliable database transactions.

Common web service specifications include:

- Web services security (WS-security): Standardizes how messages are secured and transferred through unique identifiers called tokens.
- WS-ReliableMessaging: Standardizes error handling between messages transferred across unreliable IT infrastructure.
- Web services addressing (WS-addressing): Packages routing information as metadata within SOAP headers, instead of maintaining such information deeper within the network.
- Web services description language (WSDL): Describes what a web service does, and where that service begins and ends.

When a request for data is sent to a SOAP API, it can be handled through any of the application layer protocols: HTTP (for web browsers), SMTP (for email), TCP, and others. However, once a request is received, return SOAP messages must be returned as XML documents—a markup language that is both human- and machine-readable. A completed request to a SOAP API is not cacheable by a browser, so it cannot be accessed later without resending to the API.

SOAP vs REST API

SOAP	REST
SOAP stands for Simple Object Access Protocol	REST stands for Representational State Transfer
SOAP is a protocol. SOAP was designed with a specification. It includes a WSDL file which has the required information on what the web service does in addition to the location of the web service.	REST is an Architectural style in which a web service can only be treated as a RESTful service if it follows the constraints of being 1. Client Server 2. Stateless 3. Cacheable 4. Layered System 5. Uniform Interface
SOAP cannot make use of REST since SOAP is a protocol and REST is an architectural pattern.	REST can make use of SOAP as the underlying protocol for web services, because in the end it is just an architectural pattern.
SOAP uses service interfaces to expose its functionality to client applications. In SOAP, the WSDL file provides the client with the necessary information which can be used to understand what services the web service can offer.	REST use Uniform Service locators to access to the components on the hardware device.
SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it, the amount of data transfer using SOAP is generally a lot.	REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages. Below is an example of a JSON message passed to a web server. You can see that the size of the message is comparatively smaller to SOAP.
SOAP can only work with XML format. As seen from SOAP messages, all data passed is in XML format.	REST permits different data format such as Plain text, HTML, XML, JSON, etc. But the most preferred format for transferring data is JSON.

HTTP Methods for REST API Requests

REST APIs enable you to develop all kinds of web applications having all possible CRUD (create, retrieve, update, delete) operations.

1. HTTP GET

Use *GET* requests to retrieve resource representation/information only – and not modify it in any way. As GET requests do not change the resource's state, these are said to be safe methods**.**

Additionally, GET APIs should be idempotent**.** Making multiple identical requests must produce the same result every time until another API (POST or PUT) has changed the state of the resource on the server.

If the Request-URI refers to a data-producing process, it is the produced data that shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process**.**

1.1. GET API Response Codes

- For any given HTTP GET API, if the resource is found on the server, then it must return HTTP response code 200 (OK) – along with the response body, which is usually either XML or JSON content (due to their platform-independent nature).
- In case the resource is NOT found on the server then API must return HTTP response code 404 (NOT FOUND).
- Similarly, if it is determined that the GET request itself is not correctly formed then the server will return the HTTP response code 400 (BAD REQUEST).

2. HTTP POST

Use POST APIs to create new subordinate resources, e.g., a file is subordinate to a directory containing it or a row is subordinate to a database table.

When talking strictly about REST, POST methods are used to create a new resource into the collection of resources.

Responses to this method are not cacheable unless the response includes appropriate Cache-Control or Expires header fields.

Please note that POST is neither safe nor idempotent, and invoking two identical POST requests will result in two different resources containing the same information (except resource ids).

2.1. POST API Response Codes

- Ideally, if a resource has been created on the origin server, the response SHOULD be HTTP response code 201 (Created) and contain an entity that describes the status of the request and refers to the new resource, and a Location header.

Many times, the action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either HTTP response code 200 (OK) or 204 (No Content) is the appropriate response status.

3. HTTP PUT

Use PUT APIs primarily to update an existing resource (if the resource does not exist, then API may decide to create a new resource or not).

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries should be treated as stale. Responses to PUT method are not cacheable.

3.1. PUT API Response Codes

- If a new resource has been created by the PUT API, the origin server MUST inform the user agent via the HTTP response code 201 (Created) response.
- If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request.

POST requests are made on resource collections, whereas PUT requests are made on a single resource.

4. HTTP DELETE

As the name applies, DELETE APIs delete the resources (identified by the Request-URI).

DELETE operations are idempotent. If you DELETE a resource, it's removed from the collection of resources.

Some may argue that it makes the DELETE method non-idempotent. It's a matter of discussion and personal opinion.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are not cacheable.

4.1. DELETE API Response Codes

- A successful response of DELETE requests SHOULD be an HTTP response code 200 (OK) if the response includes an entity describing the status.
- The status should be 202 (Accepted) if the action has been queued.
- The status should be 204 (No Content) if the action has been performed but the response does not include an entity.
- Repeatedly calling DELETE API on that resource will not change the outcome – however, calling DELETE on a resource a second time will return a 404 (NOT FOUND) since it was already removed.

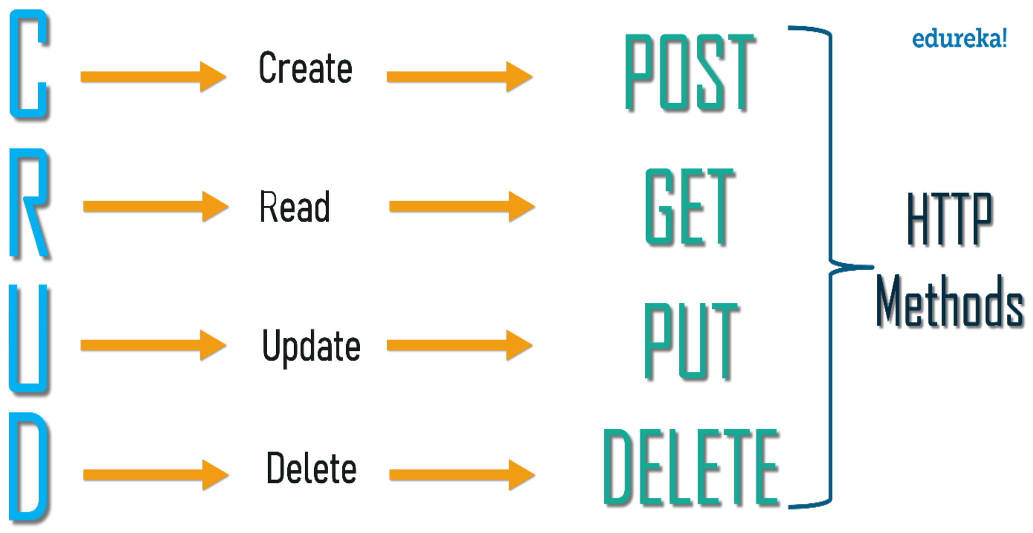


Figure 1: CRUD operation

6. JWT Token

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 HMAC algorithm) or a public/private key pair using RSA.

- Compact: Because of its size, it can be sent through an URL, POST parameter, or inside an HTTP header. Additionally, due to its size its transmission is fast.
- Self-contained: The payload contains all the required information about the user, to avoid querying the database more than once.

WHEN SHOULD YOU USE JSON WEB TOKENS?

These are some scenarios where JSON Web Tokens are useful:

- Authentication: This is the typical scenario for using JWT, once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used among systems of different domains.
- Information Exchange: JWTs are a good way of securely transmitting information between parties, because as they can be signed, for example using a public/private key pair, you can be sure that the sender is who they say they are. Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't changed.

JSON WEB TOKEN STRUCTURE

JWTs consist of three parts separated by dots (.), which are:

- Header
- Payload
- Signature

Therefore, a JWT typically looks like the following.

```
xxxxx.yyyyy.zzzzz
```

Let's break down the different parts.

Header

The header *typically* consists of two parts: the type of the token, which is JWT, and the hashing algorithm such as HMAC SHA256 or RSA.

For example:

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

Then, this JSON is Base64Url encoded to form the first part of the JWT.

Payload

The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional metadata. There are three types of claims: *reserved*, *public*, and *private* claims.

- Reserved claims: These are a set of predefined claims, which are not mandatory but recommended, thought to provide a set of useful, interoperable claims. Some of them are: iss (issuer), exp (expiration time), sub (subject), aud (audience), among others.
- Public claims: These can be defined at will by those using JWTs. But to avoid collisions they should be defined in the IANA JSON Web Token Registry or be defined as a URI that contains a collision resistant namespace.
- Private claims: These are the custom claims created to share information between parties that agree on using them.

An example of payload could be:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

The payload is then Base64Url encoded to form the second part of the JWT.

Signature

To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

For example if you want to use the HMAC SHA256 algorithm, the signature will be created in the following way.

```
HMACSHA256(

  base64UrlEncode(header) + "." +

  base64UrlEncode(payload),

  secret)
```

The signature is used to verify that the sender of the JWT is who it says it is and to ensure that the message was't changed in the way.

Putting all together

The output is three Base64 strings separated by dots that can be easily passed in HTML and HTTP environments, while being more compact compared to XML-based standards such as SAML.

The following shows a JWT that has the previous header and payload encoded and it is signed with a secret.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

JWT WORK

In authentication, when the user successfully logs in using their credentials, a JSON Web Token will be returned. Since tokens are credentials, great care must be taken to prevent security issues. In general, you should not keep tokens longer than required.

You also should not store sensitive session data in browser storage due to lack of security.

Whenever the user wants to access a protected route, it should send the JWT, typically in the Authorization header using the Bearer schema. Therefore the content of the header should look like the following.

Authorization: Bearer <token>

This is a stateless authentication mechanism as the user state is never saved in the server memory. The server's protected routes will check for a valid JWT in the Authorization header, and if there is, the user will be allowed. As JWTs are self-contained, all the necessary information is there, reducing the need of going back and forward to the database.

This allows to fully rely on data APIs that are stateless and even make requests to downstream services. It doesn't matter which domains are serving your APIs, as Cross-Origin Resource Sharing (CORS) won't be an issue as it doesn't use cookies.

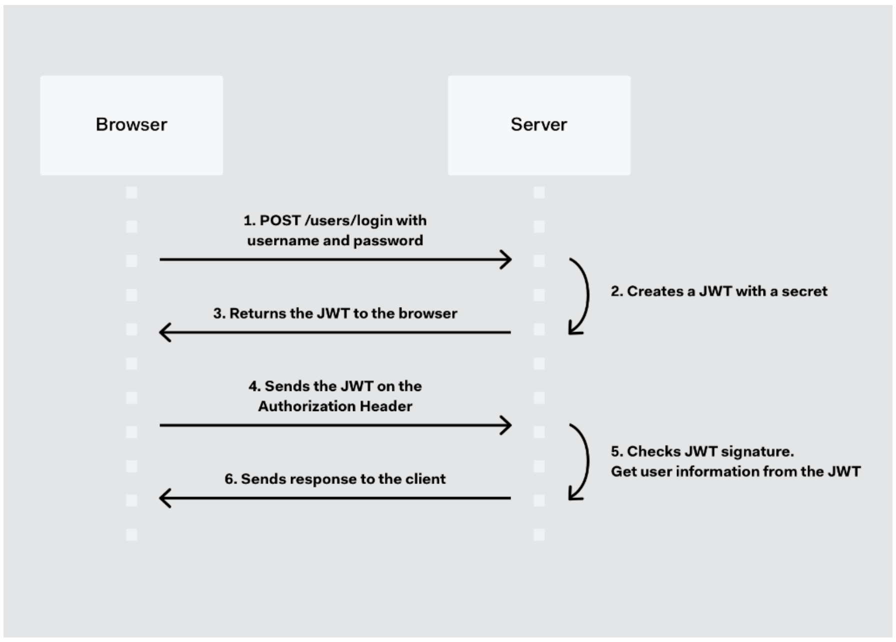
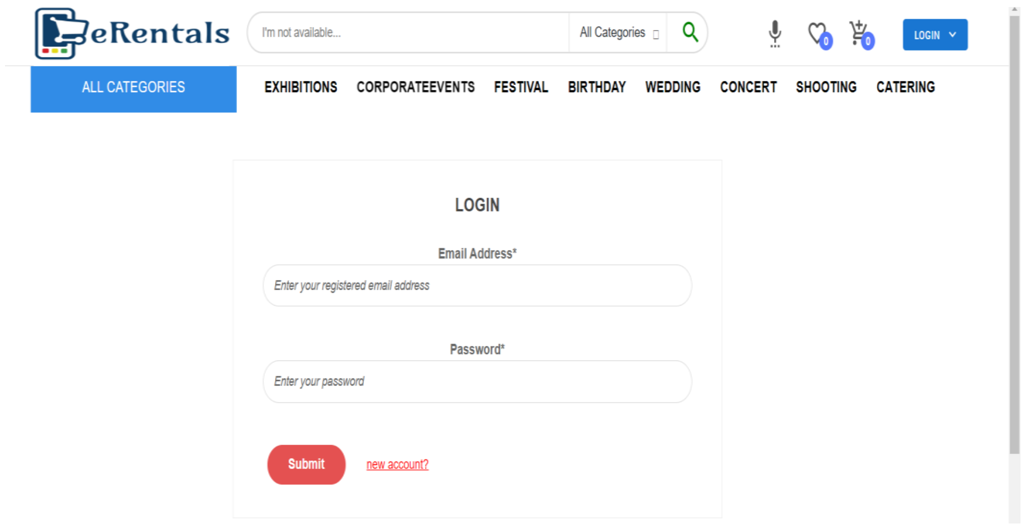

Browser
Server
1. POST /users/login with username and password
2. Creates a JWT with a secret
3. Returns the JWT to the browser
4. Sends the JWT on the Authorization Header
5. Checks JWT signature. Get user information from the JWT
6. Sends response to the client

Figure 2:JWT working

7. Implementation:



LOGIN

Email Address*

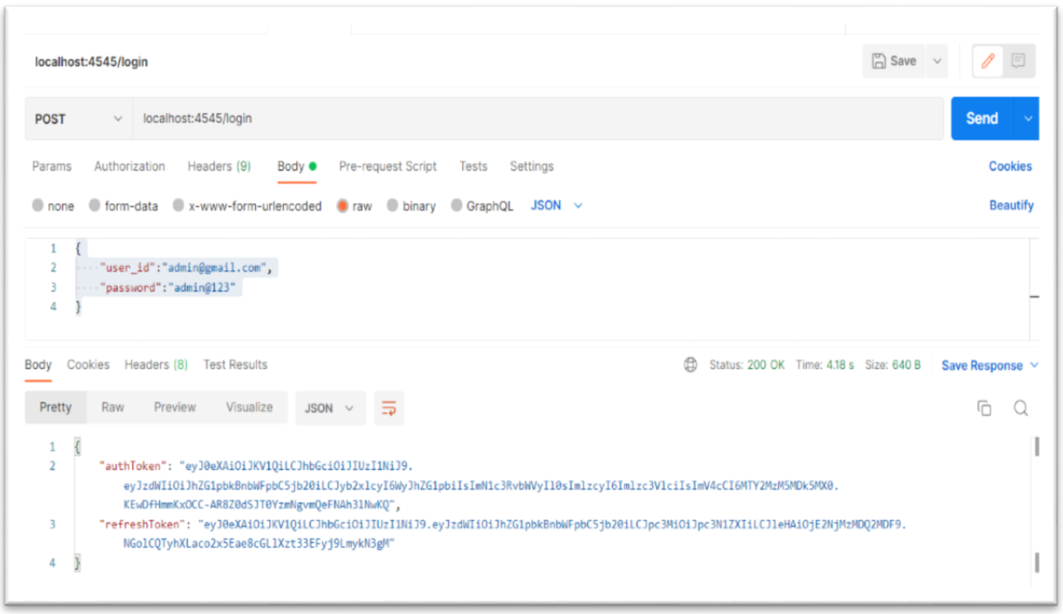
Enter your registered email address

Password*

Enter your password

Submit new account?

Figure 3: Frontend Login page



localhost:4545/login

POST localhost:4545/login

```
{
    "user_id":"admin@gmail.com",
    "password":"admin@123"
}
```

Status: 200 OK Time: 4.18 s Size: 640 B

```
{
    "authToken": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
        eyJzdWIiOiJhZG1pbkBnbWFpbC5jb20iLCJyb2xlcyI6WyJhZG1pbiIsImN1c3RvbWVyIl0sImlzcyI6Imlzc3VlciIsImV4cCI6MTY2MzM5MDk5MX0.
        KEwDfHmmKxOCC-AR8Z0dSJT0YzmNgvmQeFNAh31NwKQ",
    "refreshToken": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbkBnbWFpbC5jb20iLCJpc3MiOiJpc3N1ZXIiLCJleHAiOjE2NjMzMDQ2MDF9.
        NGolCQTyhXLaco2x5Eae8cGL1Xzt33EFyj9LmykN3gM"
}
```

Figure 4: Login page in Backend testing

8. Summary and Future Scope

I can say that this training was a great experience. Thanks to the project. I acquired deeper knowledge concerning my technical skills.

At present every website have frontend design which mostly we interact also called as UI/UX but functionality in website (like Sign-In/Sign-Out) which is done by Backend Developer.

In my summer Internship we designed E-Commerce for renting product and I worked as backend developer worked on Spring boot Framework and Maven Tool using Java also used OOPs and SOLID for efficient and readable code.

Learning Outcome

- Java Programming Language
- SpringBook Framework and Maven Tool
- SOLID Principal
- REST API
- JWT Token

At present website is a common part of web applications, and it is one of the most popular language for web designing used by professionals worldwide. If we surf internet. we can see millions of websites designed with Frontend and Backend. This training taught me how big and interactive websites are developed and what all features make websites more appealing and easier to use to the user. Creating a webpage also gave me hands on learning and made my concepts that I learnt more clear and strong.

There are huge opportunities available for the students who want to work in this field. Many private and public organizations hire web designer for their online work and website development. With the rapid advent of online industry. the demand of web development professionals is increasing. and this has created a huge job opportunity for the aspirants in the upcoming days.

Future Scope

If someone has no experience in this field, finding work can be a real challenge. A successful internship can help an individual turn an experience into a career opportunity. Some future scopes in this field are:

- Software Engineer
- Web Engineer
- Web Developer
- QA Tester
- Freelancer

9. References

- https://www.tutorialspoint.com/maven/maven_overview.htm
- https://spring.io/projects/spring-boot
- https://jwt.io/#encoded-jwt
- https://www.freecodecamp.org/news/solid-principles-explained-in-plain-english/
- https://www.guru99.com/comparison-between-web-services.html
- https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest
- https://www.ibm.com/cloud/learn/api#toc-types-of-a-PrilHB2R