

Spring Boot Micro Services

Learning Objectives

- Explain Monolithic applications
- Define Microservices
- Explore Microservices architecture
- Develop Microservices using Spring Boot

E-commerce Application

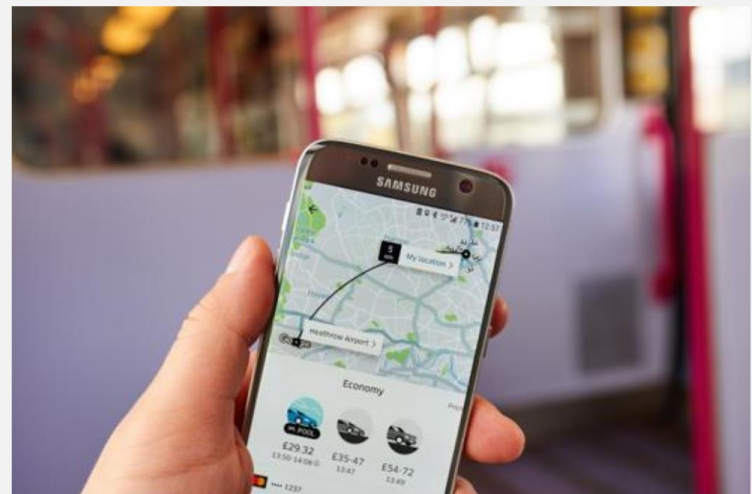
- Do you think this e-commerce application is created as one large application?
- Are all functionalities deployed as one large application to the web server?
- If there is an ongoing sale or some new features are to be added, how can the changes be launched on the application?
- Do we need to stop the application, add changes and then redeploy the application?
- Is this an efficient way?

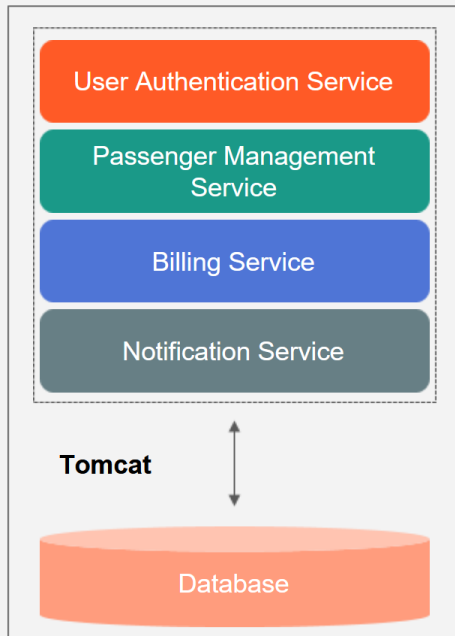


Ride-share Application

Let us imagine that you are building a ride sharing application that helps commuters book cabs at anytime of the day or night by accessing their live location.

- If we add a new feature to the application, what do you think will happen to the live location sharing feature?
- Will the feature go down at the time of redeployment?
- Will this impact the business?

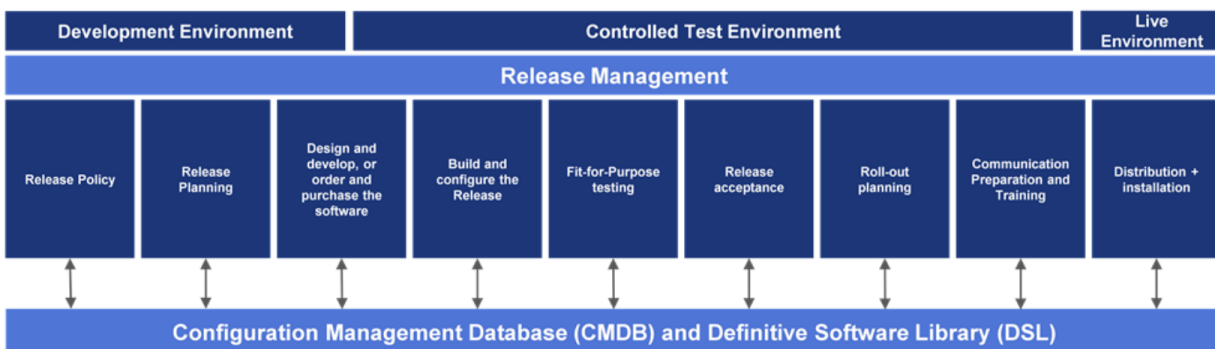


Ride Share Application**Monolithic Applications**

- In software engineering, this application describes a single-tiered software application. Here, the user interface and the data access code are combined into a single program from a single platform.
- A monolithic application is:
 - A self-contained, and independent from other computing applications.
 - Deployed as a single monolithic application. In case it is a Java web application, then it will have a single WAR file that runs on a web container such as Tomcat.

Change Impact

- If changes are to be incorporated in a monolithic application, the whole development, release, and deployment management cycle will have to be implemented all over again.
- The changes made in the application will also go live. If this is the case, then we need to shut down the entire application, make it offline, perform the changes, test the application, and then make it live again.
- To deploy a bug-fix in one of the components in the application, the entire application should be redeployed.



Disadvantages of a Monolithic Approach

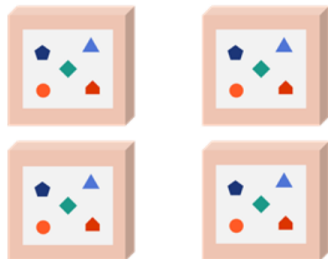
- Tight coupling between components
- Overloaded web container
- Large code base; tough for developers and QA to understand the code and business knowledge
- Less Scalable
- Obstacles in continuous deployment
- More deployment and restart times
- New technology barriers.
- Long term commitment to a tech stack

Shifting From Monolithic to Microservices, cont.

A monolithic application puts all its functionality into a single process...



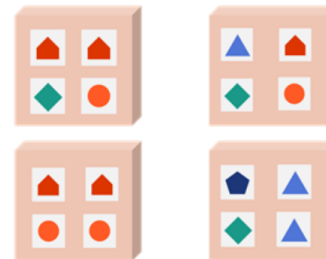
...and scales by replicating the monolith on multiple servers.



A microservices architecture puts each element of functionality into a separate service...



...and scales by distributing these services across servers, replicating as needed.



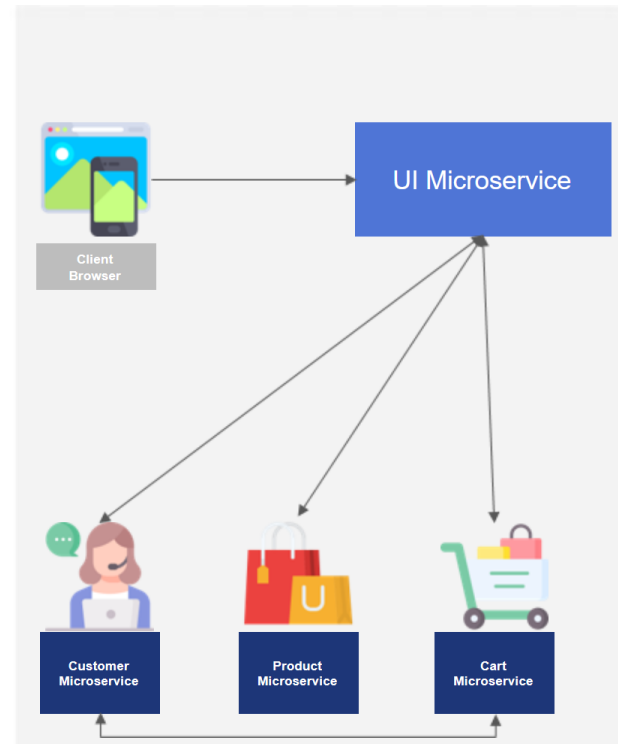
Microservices

Microservices are modular, autonomous, and logical units of functionality that are independently deployable and scalable, and:

- Are an architectural style, in which large complex software applications are composed of one or more services.
- Can be deployed independently of each other and are loosely coupled.

Each microservice focuses on completing only one task and does that one task well.

- A task represents a small business capability.



Advantages of Implementing Microservices

- **Modularity** - Each microservice demonstrates a logically cohesive, lightweight, and independent business functionality with well-defined boundaries.
- **Single functionality principle** - Each service encapsulates a single business functionality.
- **Asynchronous invocation** – The services are stateless by default, thereby helping us invoke them asynchronously.
- **Independent deployment and scalability** - Each of them can be independently deployable, and so, they are independently scaled to respond to varying workloads and user demands.
- **Self-containment** – Its deployment unit is self-contained as it includes all dependent libraries.
- **Fault tolerant** - Its architecture eliminates single point of failure through distribution of coherent functionality among other services.

Advantages of Implementing Microservices, cont.

- **Loose coupling** - They are loosely coupled with minimal dependencies on one another.
- **Well-defined communication between services** - These services communicate with each other with well-defined rest end points.
- **Extensible** - These can be leveraged to create an extensible solution by quickly onboarding newer ones.
- **Multiple technology support** - Each microservice can use any technology independent of one another.
- **Faster release cycles** - Since microservices can be developed in parallel by multiple developers the release time is less.

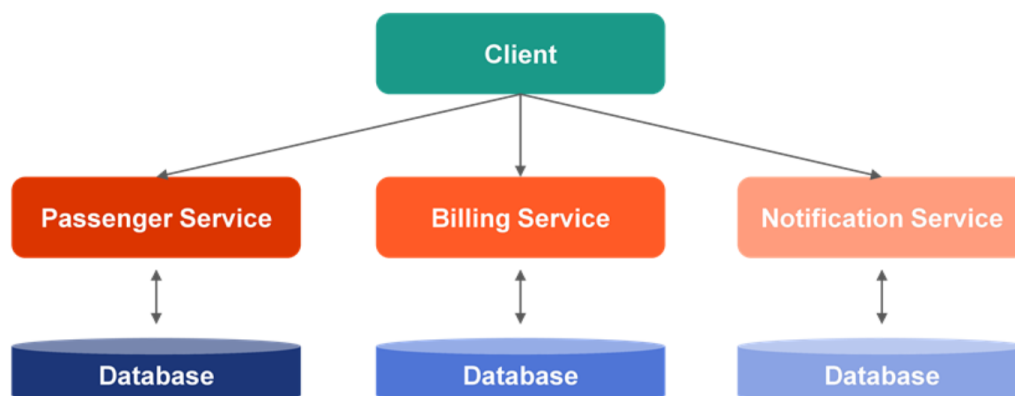
Challenges with Microservices

- **Visibility** – The new microservices added to the application must be automatically visible. This can become a challenge if the number of services added are multiple.
- **Bounded context** – It becomes challenging to set the functional boundaries of each microservice.
- **Configuration management** – Additional complexity of creating a distributed system.
- **Dynamic scale-up and scale-down** – The scaling up and scaling down of microservices based on the demand can be challenging.

Microservices Architecture

The Microservice Architecture, is an architectural style that structures an application as a collection of small autonomous services modelled around a business domain.

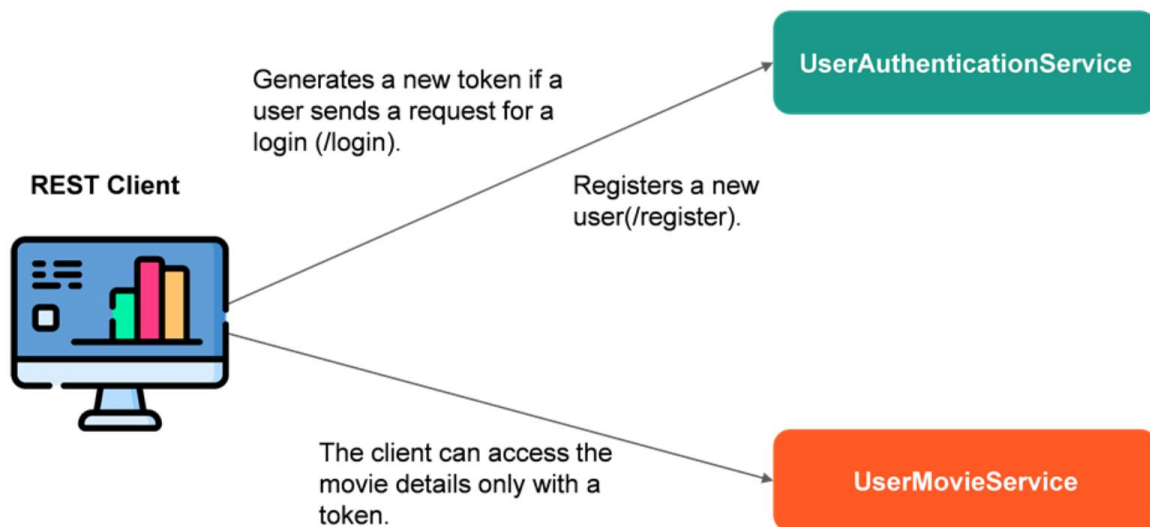
- We can decompose the Monolith for ride share as shown below:



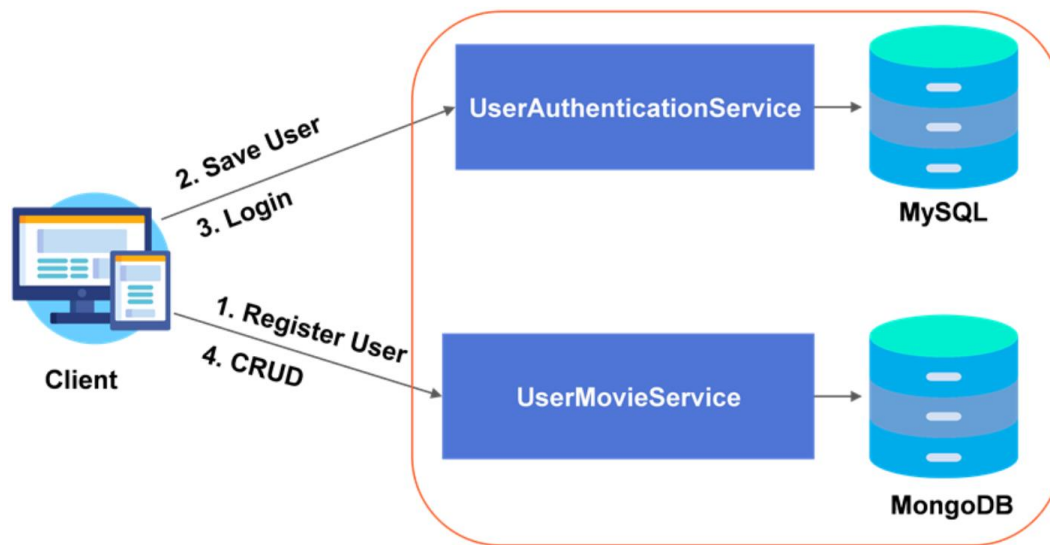
Steps to Create Microservices

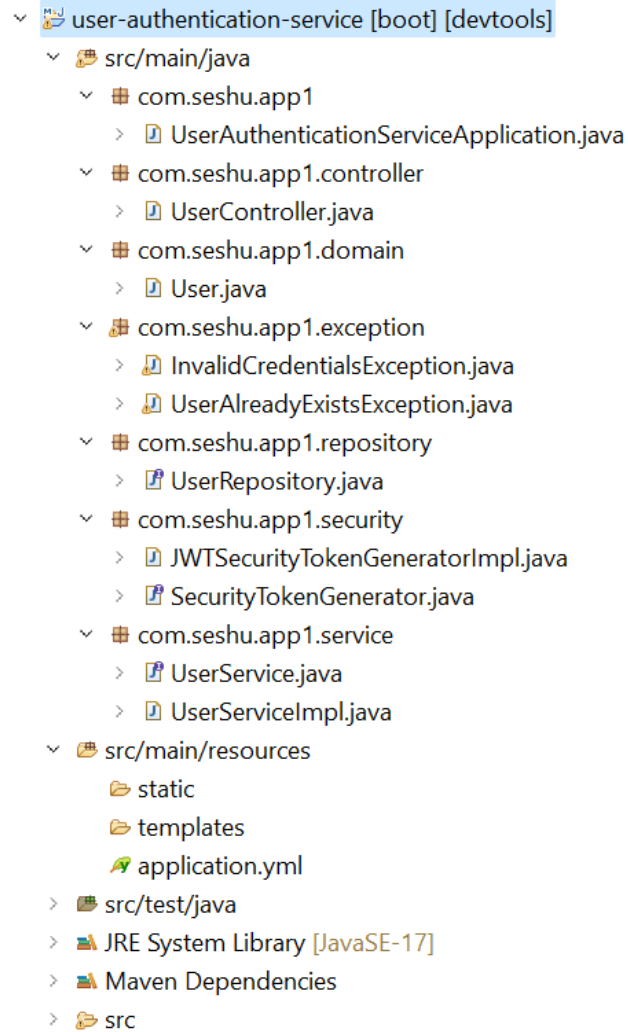
- The Spring Boot applications created during the earlier sessions such as CustomerService, ProductService, UserService are all individual applications that can be stitched together to form microservices.
- Consider a streaming application that enables users to watch movies on any smart device. The application provides multiple features to all its registered users. The user needs to register with the application in order to access some of its features.
 - Create a UserService Spring Boot application that works with the user details
 - Create a MovieService Spring Boot application that works with movie details and links which user has a favorite list or a watch later list.
 - Not all users can add movies to their favorite list or watch later list, thus we need to ensure that only registered users with valid login credentials are able to access this feature.
 - JWT can be used for authentication purposes.

Microservices – An example



How Does the Application Work?



**Example:
Micro Service 1:**

Develop first micro service project as **user-authentication-service** with following dependencies;

- X Spring Boot DevTools
- X Spring Data JPA
- X MySQL Driver
- X Spring Web

Add the following dependency in pom.xml

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.9.1</version>
</dependency>

<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.4.0-b180830.0359</version>
</dependency>
```

application.yml

```
server:
  port: 8085
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/adidb
    username: root
    password: seshu
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL57Dialect

  application:
    name: user-authentication-service
```

User.java

```
package com.seshu.appl.domain;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class User {
    @Id
    private String email;
    private String password;

    public User() {
    }

    public User(String email, String password) {
        this.email = email;
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    @Override
    public String toString() {
        return "User{" +
            "email='" + email + '\'' +
            ", password='" + password + '\'' +
            '}';
    }
}
```

UserRepository.java

```
package com.seshu.appl.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.seshu.appl.domain.User;

@Repository
public interface UserRepository extends JpaRepository<User, String> {
    User findByEmailAndPassword(String email, String password);
}
```

InvalidCredentialsException.java

```
package com.seshu.appl.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.UNAUTHORIZED, reason = "Invalid
credentials")
public class InvalidCredentialsException extends Exception{
}
```

UserAlreadyExistsException.java

```
package com.seshu.appl.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.CONFLICT, reason = "Invalid
credentials")
public class UserAlreadyExistsException extends Throwable {
}
```

UserService.java

```
package com.seshu.appl.service;

import com.seshu.appl.domain.User;
import com.seshu.appl.exception.InvalidCredentialsException;
import com.seshu.appl.exception.UserAlreadyExistsException;

public interface UserService {
    User saveUser(User user) throws UserAlreadyExistsException;
    //user name and pwd is in db or not, if not save
    User findByEmailAndPassword(String email,String password) throws
InvalidCredentialsException;
}
```

UserServiceImpl.java

```
package com.seshu.appl.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.seshu.appl.domain.User;
import com.seshu.appl.exception.InvalidCredentialsException;
import com.seshu.appl.exception.UserAlreadyExistsException;
import com.seshu.appl.repository.UserRepository;

@Service
public class UserServiceImpl implements UserService {

    private UserRepository userRepository;

    @Autowired
    public UserServiceImpl(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public User saveUser(User user) throws UserAlreadyExistsException {
        if (userRepository.findById(user.getEmail()).isPresent()) {
            throw new UserAlreadyExistsException();
        }
        System.out.println(user);
        return userRepository.save(user);
    }

    @Override
    public User findByEmailAndPassword(String email, String password)
throws InvalidCredentialsException {
        System.out.println("email" + email);
    }
}
```

```
        System.out.println("password" + password);
        User loggedInUser =
userRepository.findByEmailAndPassword(email, password);
        System.out.println(loggedInUser);
        if (loggedInUser == null) {
            throw new InvalidCredentialsException();
        }

        return loggedInUser;
    }
}
```

SecurityTokenGenerator.java

```
package com.seshu.appl.security;

import java.util.Map;

import com.seshu.appl.domain.User;

public interface SecurityTokenGenerator {
    Map<String,String> generateToken(User user); //token and message ->
    the return type can be String also
}
```

JWTSecurityTokenGeneratorImpl.java

```
package com.seshu.appl.security;

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.stereotype.Service;

import com.seshu.appl.domain.User;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;

@Service
public class JWTSecurityTokenGeneratorImpl implements
SecurityTokenGenerator {

    public Map<String, String> generateToken(User user) {
        // multiple claims for a token - 3 types - registered, public, and
        private
        String jwtToken = Jwts.builder().setIssuer("ShopZone")
            .setSubject(user.getEmail())
            .setIssuedAt(new Date())
```

```

        .signWith(SignatureAlgorithm.HS256, "mysecret")
        //mysecret is the key that has to be shared everytime you
do encrypt and decrypt process
        .compact();
        Map<String,String> map = new HashMap<>();
        map.put("token", jwtToken);
        map.put("message", "Authentication Successful");
        return map;
    }
}

```

UserController.java

```

package com.seshu.appl.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.seshu.appl.domain.User;
import com.seshu.appl.exception.InvalidCredentialsException;
import com.seshu.appl.exception.UserAlreadyExistsException;
import com.seshu.appl.security.SecurityTokenGenerator;
import com.seshu.appl.service.UserService;
import java.util.Map;

@RestController
@RequestMapping("/api/v1")
public class UserController {
    @Autowired
    private UserService userService;

    @Autowired
    private SecurityTokenGenerator securityTokenGenerator;

    @PostMapping("/user")
    public ResponseEntity<?> saveUser(@RequestBody User user) throws
UserAlreadyExistsException {
        return new ResponseEntity<>(userService.saveUser(user),
HttpStatus.CREATED);
    }

    @PostMapping("/login")

```

```
    public ResponseEntity<?> loginUser(@RequestBody User user) throws
InvalidCredentialsException {
        User retrievedUser =
userService.findByEmailAndPassword(user.getEmail(), user.getPassword());

        if (retrievedUser == null) {
            throw new InvalidCredentialsException();
        }
        Map<String, String> map =
securityTokenGenerator.generateToken(user);
        return new ResponseEntity<>(map, HttpStatus.OK);
    }
}
```

UserAuthenticationServiceApplication.java

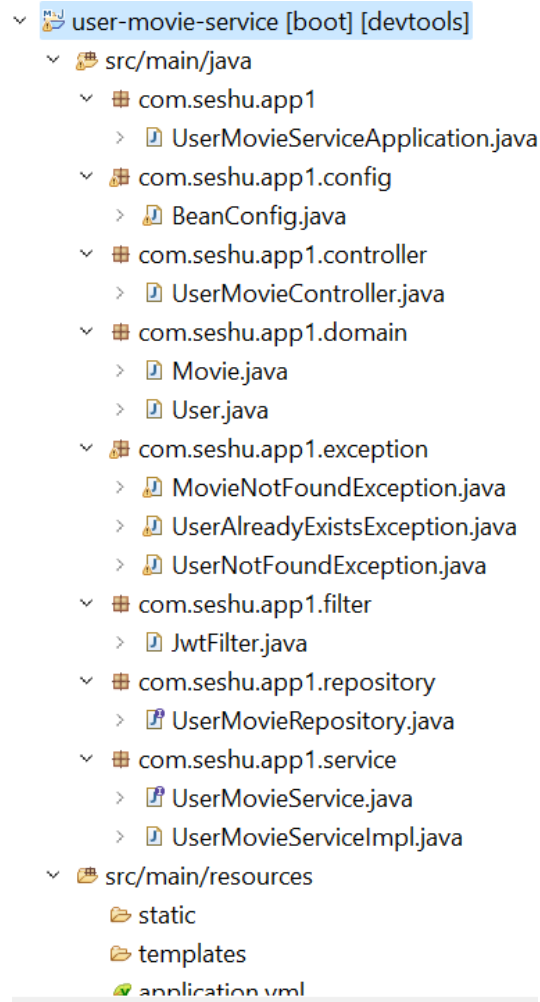
```
package com.seshu.appl;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class UserAuthenticationServiceApplication {
    public static void main(String[] args) {

        SpringApplication.run(UserAuthenticationServiceApplication.class,
args);
    }
}
```


Micro Service 2: user-movie-service application



Generate second micro service as *user-movie-service* with following dependencies.

- X Spring Boot DevTools
- X Spring Data MongoDB
- X Spring Web

Update pom.xml file with following dependency;

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.9.1</version>
</dependency>

<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.4.0-b180830.0359</version>
</dependency>
```

application.yml

```
server:
  port: 8081
spring:
  data:
    mongodb:
      uri: mongodb://localhost:27017/
      database: moviesdb
  application:
    name: user-movie-service
```

Movie.java

```
package com.seshu.appl.domain;

import org.springframework.data.annotation.Id;

import java.util.List;

public class Movie {
    @Id
    private String movieId;
    private String movieName;
    private String genre;
    private List<String> leadActors;
    private String director;
    private int yearOfRelease;
    private int rating;

    public Movie() {
    }

    public Movie(String movieId, String movieName, String genre,
List<String> leadActors, String director, int yearOfRelease, int rating)
{
    this.movieId = movieId;
    this.movieName = movieName;
    this.genre = genre;
    this.leadActors = leadActors;
    this.director = director;
    this.yearOfRelease = yearOfRelease;
    this.rating = rating;
}

    public String getMovieId() {
        return movieId;
    }

    public void setMovieId(String movieId) {
        this.movieId = movieId;
    }

    public String getMovieName() {
        return movieName;
    }

    public void setMovieName(String movieName) {
        this.movieName = movieName;
    }
}
```

```
public String getGenre() {  
    return genre;  
}  
  
public void setGenre(String genre) {  
    this.genre = genre;  
}  
  
public List<String> getLeadActors() {  
    return leadActors;  
}  
  
public void setLeadActors(List<String> leadActors) {  
    this.leadActors = leadActors;  
}  
  
public String getDirector() {  
    return director;  
}  
  
public void setDirector(String director) {  
    this.director = director;  
}  
  
public int getYearOfRelease() {  
    return yearOfRelease;  
}  
  
public void setYearOfRelease(int yearOfRelease) {  
    this.yearOfRelease = yearOfRelease;  
}  
  
public int getRating() {  
    return rating;  
}  
  
public void setRating(int rating) {  
    this.rating = rating;  
}  
}
```

User.java

```
package com.seshu.appl.domain;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import java.util.List;

@Document
public class User {
    @Id
    private String email;
    private String userName;
    private String password;
    private String phoneNumber;
    private List<Movie> movieList;

    public User() {
    }

    public User(String email, String userName, String password, String
phoneNumber, List<Movie> movieList) {
        this.email = email;
        this.userName = userName;
        this.password = password;
        this.phoneNumber = phoneNumber;
        this.movieList = movieList;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getPassword() {
        return password;
    }
}
```

```
public void setPassword(String password) {
    this.password = password;
}

public String getPhoneNumber() {
    return phoneNumber;
}

public void setPhoneNumber(String phoneNumber) {
    this.phoneNumber = phoneNumber;
}

public List<Movie> getMovieList() {
    return movieList;
}

public void setMovieList(List<Movie> movieList) {
    this.movieList = movieList;
}

@Override
public String toString() {
    return "User{" +
        "email='" + email + '\'' +
        ", userName='" + userName + '\'' +
        ", password='" + password + '\'' +
        ", phoneNumber='" + phoneNumber + '\'' +
        ", movieList=" + movieList +
        '}';
}
```

UserMovieRepository.java

```
package com.seshu.app1.repository;

import org.springframework.data.mongodb.repository.MongoRepository;
import com.seshu.app1.domain.User;

public interface UserMovieRepository extends MongoRepository<User, String>
{
    User findByEmail(String email);
}
```

MovieNotFoundException.java

```
package com.seshu.appl.exception;

public class MovieNotFoundException extends Throwable {
}
```

UserAlreadyExistsException.java

```
package com.seshu.appl.exception;

public class UserAlreadyExistsException extends Exception {
}
```

UserNotFoundException.java

```
package com.seshu.appl.exception;

public class UserNotFoundException extends Throwable {
}
```

UserMovieService.java

```
package com.seshu.appl.service;

import java.util.List;

import com.seshu.appl.domain.Movie;
import com.seshu.appl.domain.User;
import com.seshu.appl.exception.MovieNotFoundException;
import com.seshu.appl.exception.UserAlreadyExistsException;
import com.seshu.appl.exception.UserNotFoundException;

public interface UserMovieService {

    User registerUser(User user) throws UserAlreadyExistsException;

    User saveUserMovieToList(Movie movie, String email) throws
UserNotFoundException;

    User deleteUserMovieFromList(String email, String movieId) throws
UserNotFoundException, MovieNotFoundException;

    List<Movie> getAllUserMovies(String email) throws
UserNotFoundException;
}
```

UserMovieServiceImpl.java

```
package com.seshu.appl.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.seshu.appl.domain.Movie;
import com.seshu.appl.domain.User;
import com.seshu.appl.exception.MovieNotFoundException;
import com.seshu.appl.exception.UserAlreadyExistsException;
import com.seshu.appl.exception.UserNotFoundException;
import com.seshu.appl.repository.UserMovieRepository;

import java.util.Arrays;
import java.util.List;

@Service
public class UserMovieServiceImpl implements UserMovieService{
    private UserMovieRepository userMovieRepository;
    @Autowired
    public UserMovieServiceImpl(UserMovieRepository userMovieRepository)
    {
        this.userMovieRepository = userMovieRepository;
    }

    @Override
    public User registerUser(User user) throws UserAlreadyExistsException
    {
        if(userMovieRepository.findById(user.getEmail()).isPresent())
        {
            throw new UserAlreadyExistsException();
        }
        return userMovieRepository.save(user);
    }

    @Override
    public User saveUserMovieToList(Movie movie, String email) throws
UserNotFoundException {
        if(userMovieRepository.findById(email).isEmpty())
        {
            throw new UserNotFoundException();
        }
        User user = userMovieRepository.findByEmail(email);
        if(user.getMovieList() == null)
        {
            user.setMovieList(Arrays.asList(movie));
        }
        else {
```



```
        List<Movie> movies = user.getMovieList();
        movies.add(movie);
        user.setMovieList(movies);
    }
    return userMovieRepository.save(user);
}

@Override
public User deleteUserMovieFromList(String email, String movieId)
throws UserNotFoundException, MovieNotFoundException {
    boolean movieIdIsPresent = false;
    if(userMovieRepository.findById(email).isEmpty())
    {
        throw new UserNotFoundException();
    }
    User user = userMovieRepository.findById(email).get();
    List<Movie> movies = user.getMovieList();
    movieIdIsPresent = movies.removeIf(x-
>x.getMovieId().equals(movieId));
    if(!movieIdIsPresent)
    {
        throw new MovieNotFoundException();
    }
    user.setMovieList(movies);
    return userMovieRepository.save(user);
}

@Override
public List<Movie> getAllUserMovies(String email) throws
UserNotFoundException {
    if(userMovieRepository.findById(email).isEmpty())
    {
        throw new UserNotFoundException();
    }
    return userMovieRepository.findById(email).get().getMovieList();
}
}
```

BeanConfig.java

```
package com.seshu.app1.config;

import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.seshu.app1.filter.JwtFilter;

@Configuration
public class BeanConfig {
    @Bean
    public FilterRegistrationBean jwtFilterBean() {
        FilterRegistrationBean filterRegistrationBean = new
FilterRegistrationBean();
        filterRegistrationBean.setFilter(new JwtFilter());
        filterRegistrationBean.addUrlPatterns("/api/v1/user/*");
        return filterRegistrationBean;
    }
}
```

JwtFilter.java

```
package com.seshu.appl.filter;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import org.springframework.web.filter.GenericFilterBean;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class JwtFilter extends GenericFilterBean {
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException,
ServletException {
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        HttpServletResponse response = (HttpServletResponse)
servletResponse;

        //expects the token to come from the header
        final String authHeader = request.getHeader("Authorization");
        if(request.getMethod().equals("OPTIONS")) {
            //if the method is options the request can pass through not
validation of token is required
            response.setStatus(HttpServletResponse.SC_OK);
            filterChain.doFilter(request, response);
        }
        else if(authHeader == null || !authHeader.startsWith("Bearer "))
        {
            throw new ServletException("Missing or Invalid Token");
        }
        //extract token from the header
        String token = authHeader.substring(7); //Bearer => 6+1 = 7, since
token begins with Bearer

        //token validation
        Claims claims =
Jwts.parser().setSigningKey("mysecret").parseClaimsJws(token).getBody();
        request.setAttribute("claims", claims);

        //pass the claims in the request, anyone wanting to

        filterChain.doFilter(request, response);
    }
}
```

```
}  
}
```

UserMovieController.java

```
package com.seshu.appl.controller;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.*;  
  
import com.seshu.appl.domain.Movie;  
import com.seshu.appl.domain.User;  
import com.seshu.appl.exception.MovieNotFoundException;  
import com.seshu.appl.exception.UserAlreadyExistsException;  
import com.seshu.appl.exception.UserNotFoundException;  
import com.seshu.appl.service.UserMovieService;  
  
@RestController  
@RequestMapping("/api/v2/")  
public class UserMovieController {  
    @Autowired  
    private UserMovieService userMovieService;  
  
    private ResponseEntity<?> responseEntity;  
  
    @PostMapping("/register")  
    public ResponseEntity<?> registerUser(@RequestBody User user) throws  
UserAlreadyExistsException {  
        try {  
            responseEntity = new  
ResponseEntity<>(userMovieService.registerUser(user),  
HttpStatus.CREATED);  
        }  
        catch (UserAlreadyExistsException e)  
        {  
            throw new UserAlreadyExistsException();  
        }  
        return responseEntity;  
    }  
    @PostMapping("/user/{email}/movie")  
    public ResponseEntity<?> saveUserMovieToList(@RequestBody Movie  
movie, @PathVariable String email) throws UserNotFoundException {  
        try {
```

```
        responseEntity = new
ResponseEntity<>(userService.saveUserMovieToList(movie, email),
HttpStatus.CREATED);
    }
    catch (UserNotFoundException e)
    {
        throw new UserNotFoundException();
    }
    return responseEntity;
}
@GetMapping("/user/{email}/movies")
public ResponseEntity<?> getAllUserMoviesFromList(@PathVariable
String email) throws UserNotFoundException {
    try{
        responseEntity = new
ResponseEntity<>(userService.getAllUserMovies(email),
HttpStatus.OK);
    } catch (UserNotFoundException e)
    {
        throw new UserNotFoundException();
    }
    return responseEntity;
}
@DeleteMapping("/user/{email}/{movieId}")
public ResponseEntity<?> deleteUserProductFromList(@PathVariable
String email,@PathVariable String movieId)
    throws UserNotFoundException, MovieNotFoundException
    {
        try {
            responseEntity = new
ResponseEntity<>(userService.deleteUserMovieFromList(email,
movieId), HttpStatus.OK);
        } catch (UserNotFoundException | MovieNotFoundException m) {
            throw new MovieNotFoundException();
        }
        return responseEntity;
    }
}
```

UserMovieServiceApplication.java

```
package com.seshu.app1;

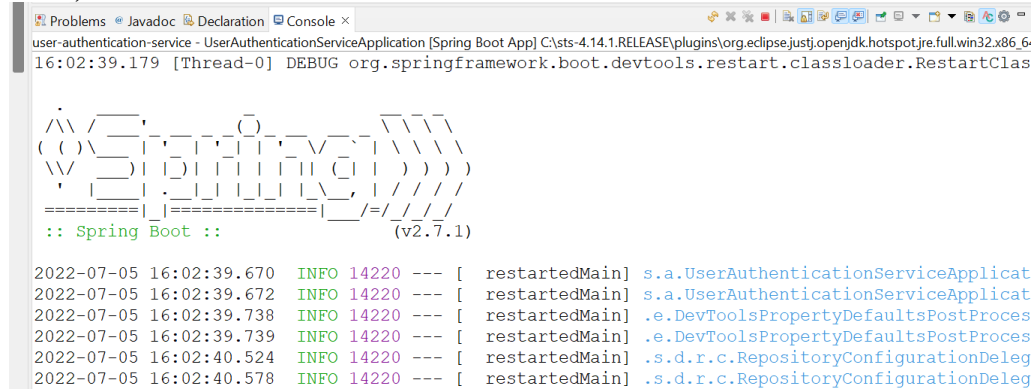
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class UserMovieServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(UserMovieServiceApplication.class, args);
    }
}
```

Execution steps:

First, run user-authentication-service starter



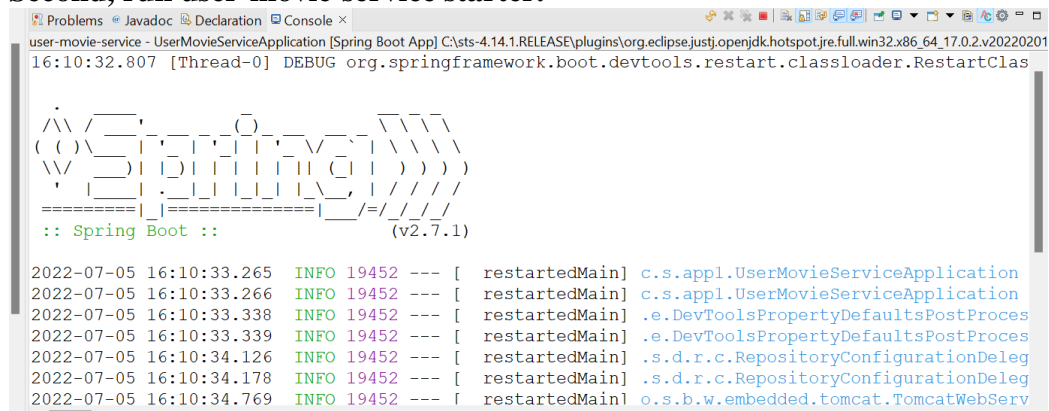
```

Problems Javadoc Declaration Console x
user-authentication-service - UserAuthenticationServiceApplication [Spring Boot App] C:\sts-4.14.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64.jre\bin\java.exe
16:02:39.179 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassloader
:: Spring Boot :: (v2.7.1)

2022-07-05 16:02:39.670 INFO 14220 --- [ restartedMain] s.a.UserAuthenticationServiceApplication
2022-07-05 16:02:39.672 INFO 14220 --- [ restartedMain] s.a.UserAuthenticationServiceApplication
2022-07-05 16:02:39.738 INFO 14220 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor
2022-07-05 16:02:39.739 INFO 14220 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor
2022-07-05 16:02:40.524 INFO 14220 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate
2022-07-05 16:02:40.578 INFO 14220 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate

```

Second, run user-movie-service starter.



```

Problems Javadoc Declaration Console x
user-movie-service - UserMovieServiceApplication [Spring Boot App] C:\sts-4.14.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64.jre\bin\java.exe
16:10:32.807 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassloader
:: Spring Boot :: (v2.7.1)

2022-07-05 16:10:33.265 INFO 19452 --- [ restartedMain] c.s.appl.UserMovieServiceApplication
2022-07-05 16:10:33.266 INFO 19452 --- [ restartedMain] c.s.appl.UserMovieServiceApplication
2022-07-05 16:10:33.338 INFO 19452 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor
2022-07-05 16:10:33.339 INFO 19452 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor
2022-07-05 16:10:34.126 INFO 19452 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate
2022-07-05 16:10:34.178 INFO 19452 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate
2022-07-05 16:10:34.769 INFO 19452 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer

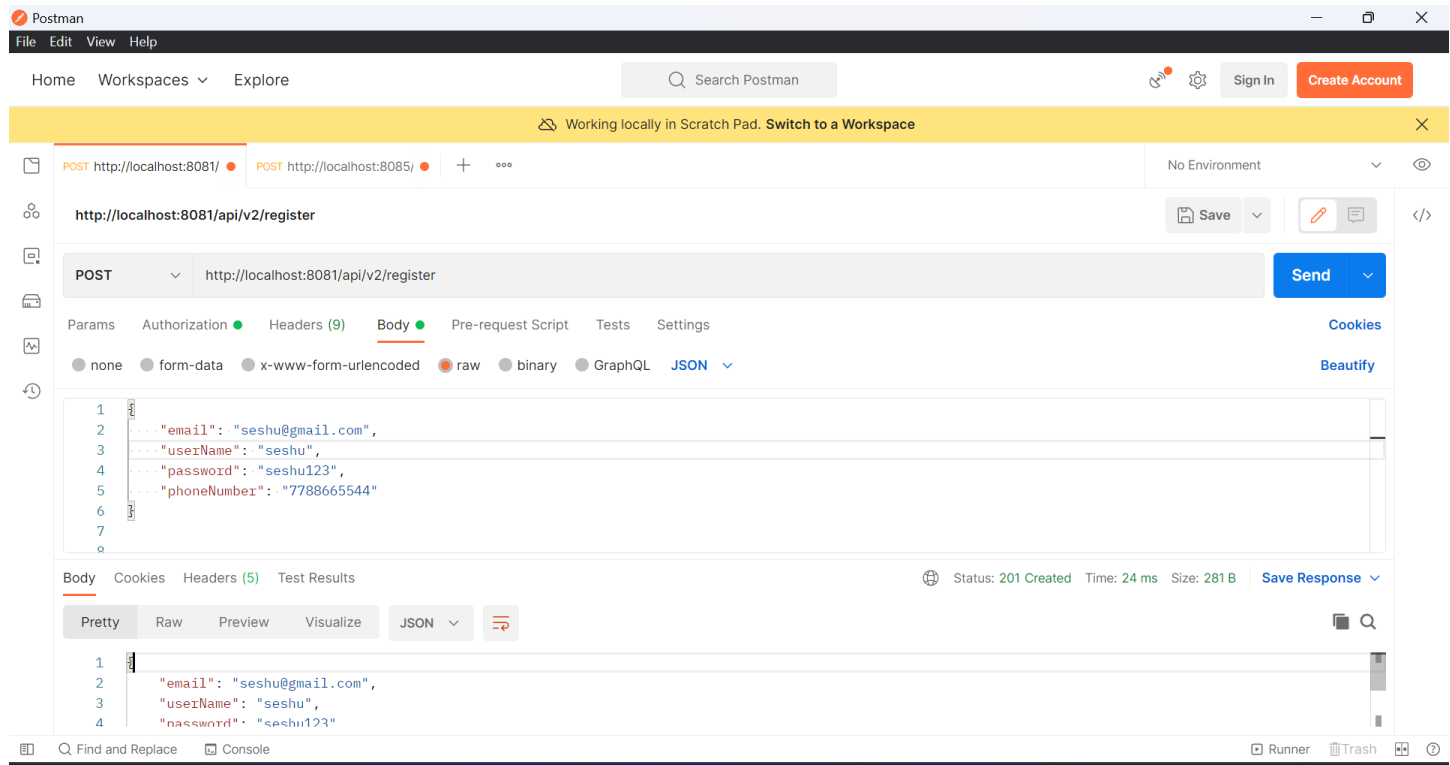
```

Testing application using Postman tool

Register the user

<http://localhost:8081/api/v2/register>

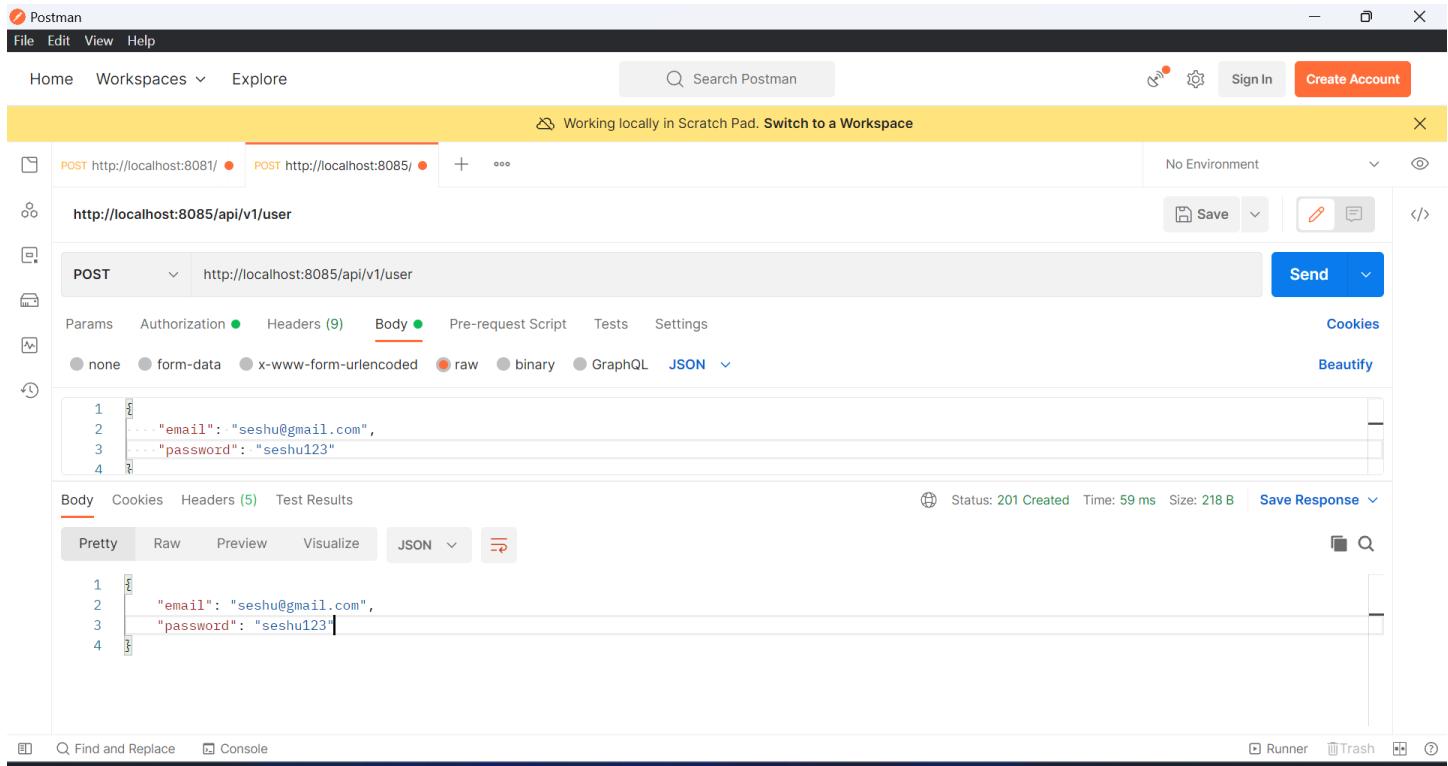
```
{
  "email": "seshu@gmail.com",
  "userName": "seshu",
  "password": "seshu123",
  "phoneNumber": "7788665544"
}
```



Save the User;

<http://localhost:8085/api/v1/user>

```
{
  "email": "seshu@gmail.com",
  "password": "seshu123"
}
```



Login

<http://localhost:8085/api/v1/login>

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Home', 'Workspaces', and 'Explore' tabs. Below this is a search bar and buttons for 'Sign In' and 'Create Account'. A yellow banner indicates 'Working locally in Scratch Pad. Switch to a Workspace'. The main workspace shows a collection of requests, with the selected request being a POST to 'http://localhost:8085/api/v1/login'. The request body is in JSON format, containing 'email': 'seshu@gmail.com' and 'password': 'seshu123'. The response is also in JSON format, showing a successful authentication with a message and a token. The status is 200 OK, time is 29 ms, and size is 358 B.

Postman

File Edit View Help

Home Workspaces Explore

Search Postman

Sign In Create Account

Working locally in Scratch Pad. Switch to a Workspace

POST http://localhost:8081/ POST http://localhost:8085/ POST http://localhost:8085/ + ...

No Environment

http://localhost:8085/api/v1/login

Save

Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1 2 3 4 5

1 2 3 4

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 29 ms Size: 358 B Save Response

Pretty Raw Preview Visualize JSON

1 2 3 4

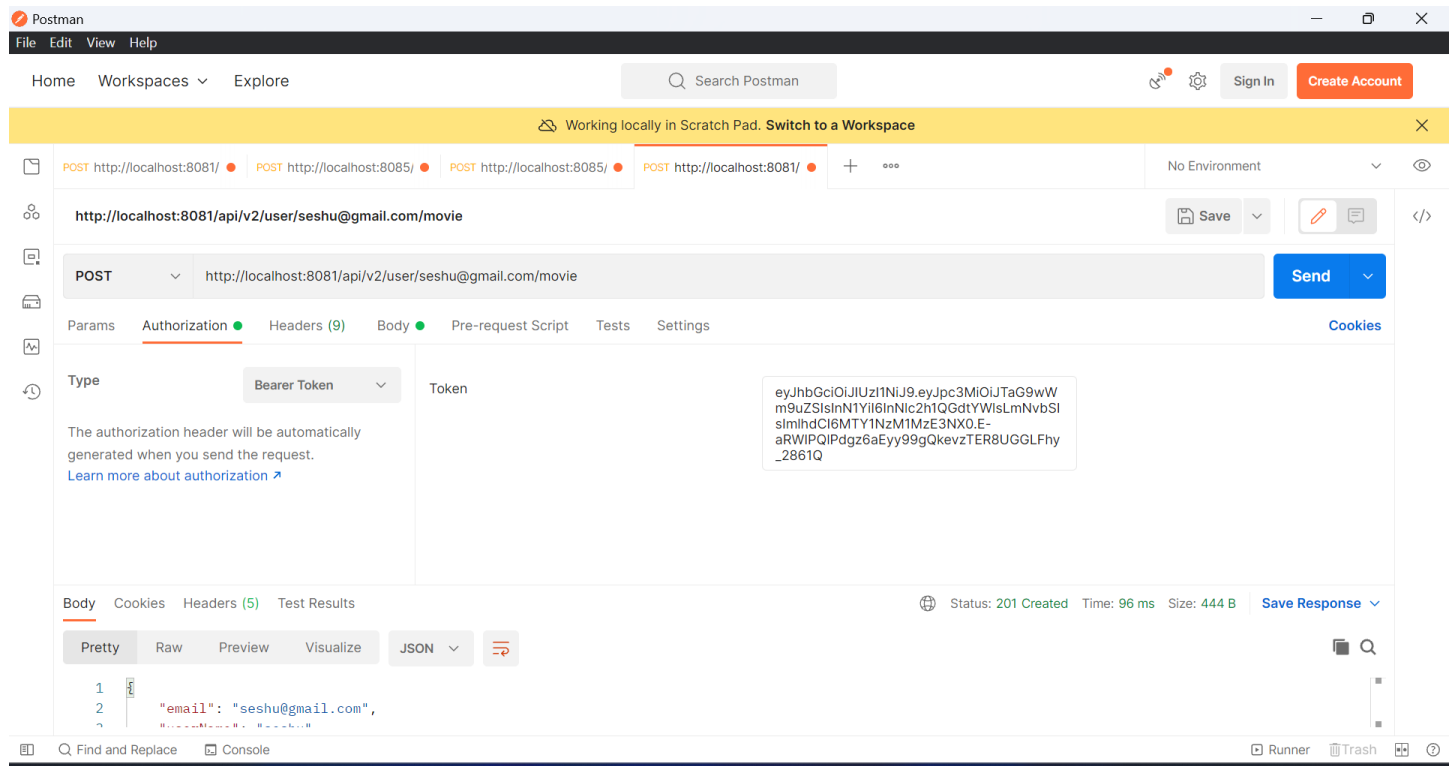
Find and Replace Console

Runner Trash

Add a favourite movie to a user;

<http://localhost:8081/api/v2/user/seshu@gmail.com/movie>

Add token



Add in request body

```
{
  "movieId": "M001",
  "movieName": "Titanic",
  "genre": "Drama",
  "leadActors": ["Kate winslet", "Leonardo DiCaprio"],
  "director": "James Cameron",
  "yearOfRelease": 1997,
  "rating": 8
}
```

Postman interface showing a POST request to `http://localhost:8081/api/v2/user/seshu@gmail.com/movie`. The request body is a JSON object with the following data:

```
1 {
2   "movieId": "M001",
3   "movieName": "Titanic",
4   "genre": "Drama",
5   "leadActors": ["Kate winslet", "Leonardo DiCaprio"],
6   "director": "James Cameron",
7   "yearOfRelease": 1997,
8   "rating": 8
9 }
10
```

The response status is 201 Created, Time: 96 ms, Size: 444 B. The response is saved and beautified.

Display all favourite movies list;

<http://localhost:8081/api/v2/user/seshu@gmail.com/movies>

Postman interface showing a GET request to `http://localhost:8081/api/v2/user/seshu@gmail.com/movies`. The response status is 200 OK, Time: 27 ms, Size: 331 B. The response body is a JSON array containing movie details:

```
1 [
2   {
3     "movieId": "M001",
4     "movieName": "Titanic",
5     "genre": "Drama",
6     "leadActors": [
7       "Kate winslet",
8       "Leonardo DiCaprio"
9     ],
10    "director": "James Cameron",
11    "yearOfRelease": 1997
12  }
13 ]
```