

Full Stack Java Developer Capstone

MyMovie Plan



Software Development

Phase 5 - Testing In a Dev-Ops Lifecycle

Developed by Boigantso Mabena

Sprint Planning:

1st Sprint:

- Creating the flow of the application.
- Initialising the git repository to track changes as development progresses.
- Creating a Spring boot application to fulfil user requirements.
- Configure the Database to maintain the data used for the application.
- Adding the Required dependencies used for the application.

2nd Sprint:

- Implemented the Model Layer and Service Layer for the project.
 - Implemented the Business Logic part.
 - Creating API methods to fetch the response.
 - Configure the API to check the Response of the requested object.
- Testing in the Postman Tool to check whether the API methods are working or not by Response.

3rd Sprint:

- Creating an Angular application to fulfil user requirements.
- Adding the Required Packages used for the application.
- Creating Components and Services, CSS.
- Creating Reactive Forms and Validations to the Forms.
- Creating Service methods to fetch the response.
- Passing the Response from the Service to the component class.
- Implemented the Query params using routing.
- Integrated the Payment Gateway using Stripe API for payments.

Core Concepts Used In The Project:

- **Object-Oriented**: used to create and model objects for users and their credentials.
- **Reactive Forms**: Reactive forms provide a model-driven approach to handling form inputs whose values change over time along with Validations.
- **Component**: Angular components are a subset of directives, always associated with a template.
- **Lifecycle hooks**: ngOnInit()
- **Data Binding**: Data binding deals with how to bind your data from components to HTML DOM elements (Templates). We can easily interact with applications without worrying about how to insert our data. We can make connections in two different ways: “1-way” and “2-way binding”.
- **Directives**: In Angular, Directives are defined as classes that can add new behaviour to the elements in the template or modify existing behaviour.
- **Routing**: The process of defining the navigation element and its associated view is called routing in Angular. Angular provides a separate module, the Router module, for setting up navigation in an Angular application.
- **Services**: Angular Services is a piece of reusable code with a focused purpose. A code that you will use across multiple components in your application.
- **Databases**: used to store and retrieve data.
- **Data Sources**: used to define a set of properties required to identify and access the database.
- **Collections**: used some collections such as array-list to store collection of data.
- **Collections**: used Java8 Streams to filter and fetch a collection of data.
- Custom Exception Handling: used to catch problems that arise in the code, especially in I/O blocks.
- **MVC**: Micro Service is an architecture that allows the developers to develop and deploy services independently.
- **Micro Services (Spring Boot)**: Micro Service is an architecture that allows developers to develop and deploy services independently.
- **RXJS Concepts**: RxJS provides an implementation of the Observable type, which is needed until the type becomes part of the language. The library also provides utility functions for creating and working with observables.
- **API**: API stands for application programming interface, which is a set of definitions and protocols for building and integrating application software.

Technologies Used in the Product:

1. **Angular**: To create a Quiz Application.
2. **Angular Material**: Angular Material is a User Interface (UI) component library that developers can use in their Angular projects to speed up the development of elegant and consistent user interfaces.
3. **Visual Studio Code**: Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running, and version control. It aims to provide just the tools a developer needs for a quick code-build-debug cycle.
4. **Spring Boot**: To create an E-commerce project for Sporty Shoes.
5. **Postman**: To test the API request and response.
6. **MySQL**: To create and manage the database.
7. **JPA**: To manage the operations for the application.
8. **Lombok**: It is the tool of the java library that was used to generate code for minimising the unused code.
9. **Rest-API**: To create the API methods and to check the response of the object entities.
10. **Eclipse**: To write and execute the code.
11. **Tomcat**: To deploy application

Admin Portal:

It deals with all the backend data generation and product information. The admin user can manage the following functions:

1. Add or remove different genres to or from the application to build a rich product line
2. Edit movie details like name, ticket price, language, description, and show timings to keep it aligned to the current prices
3. Enable or disable the movie shows from the application

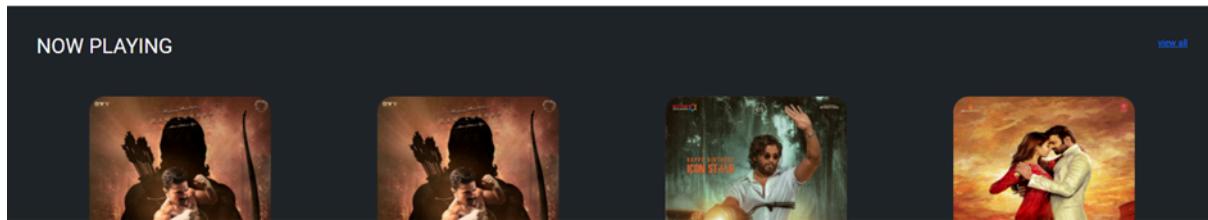
User Portal:

The end-user can do:

1. Sign-in to the application to maintain a record of activities
2. Search for movie tickets based on the search keyword
3. Apply filters and sort results based on different genres
4. Add all the selected movie tickets to a cart and customise the purchase at the end
5. Experience a seamless payment process
6. Receive a booking summary page once the payment is complete

Screenshots (Product Experience):

1. Home Page



2. Login Page

NMS Cinemas

Home [Login](#) Register Forgot Password

Invalid username or password

Login Page

Enter your username
i.riyazu@gmail.com

Enter your password

Login

3. Registration Page:

NMS Cinemas

Home Login [Register](#) Forgot Password

Register Here

Enter your Email

Enter your name Enter your Mobile +91

Enter your password Select Gender Male

Agree to terms & conditions

Register

4. User Home Page

NMS Cinemas Home Movies About Us Contact Us Admin Panel ▾

Cinema Halls	Shows	Movies												
INOX PVR GOPALAN GRAND MALL	<table border="1"><tbody><tr><td>Morning Show</td><td> </td><td>Time: 11:00 AM</td></tr><tr><td>Matinee</td><td> </td><td>Time: 02:00 PM</td></tr><tr><td>First Show</td><td> </td><td>Time: 06:00 PM</td></tr><tr><td>Second Show</td><td> </td><td>Time: 09:00 PM edit remove</td></tr></tbody></table>	Morning Show		Time: 11:00 AM	Matinee		Time: 02:00 PM	First Show		Time: 06:00 PM	Second Show		Time: 09:00 PM edit remove	 Rrr lang: Telugu start: June 2, 2021 end: June 10, 2021
Morning Show		Time: 11:00 AM												
Matinee		Time: 02:00 PM												
First Show		Time: 06:00 PM												
Second Show		Time: 09:00 PM edit remove												
Add Cinema Hall	Add Show	Add New Movie												

5. Your Profile (MyAccount)

NMS Cinemas Home Movies About Us Contact Us Admin Panel ▾

Cinema Halls	Shows	Movies												
INOX PVR GOPALAN GRAND MALL	<table border="1"><tbody><tr><td>Morning Show</td><td> </td><td>Time: 11:00 AM</td></tr><tr><td>Matinee</td><td> </td><td>Time: 02:00 PM</td></tr><tr><td>First Show</td><td> </td><td>Time: 06:00 PM</td></tr><tr><td>Second Show</td><td> </td><td>Time: 09:00 PM edit remove</td></tr></tbody></table>	Morning Show		Time: 11:00 AM	Matinee		Time: 02:00 PM	First Show		Time: 06:00 PM	Second Show		Time: 09:00 PM edit remove	 Rrr lang: Telugu start: June 2, 2021 end: June 10, 2021
Morning Show		Time: 11:00 AM												
Matinee		Time: 02:00 PM												
First Show		Time: 06:00 PM												
Second Show		Time: 09:00 PM edit remove												
Add Cinema Hall	Add Show	Add New Movie												

6. Display Details

SHOW

Now Playing

Search for movie, genre or language

FILTERS

Show
Now Playing

Language Clear

Genre Clear

Now Playing Now Playing Now Playing

7. Movie Details:

RNG Cinemas

PUSHPA : Pushpa

4D Telugu

2H 30M . Crime, Action, Romance

Book Tickets

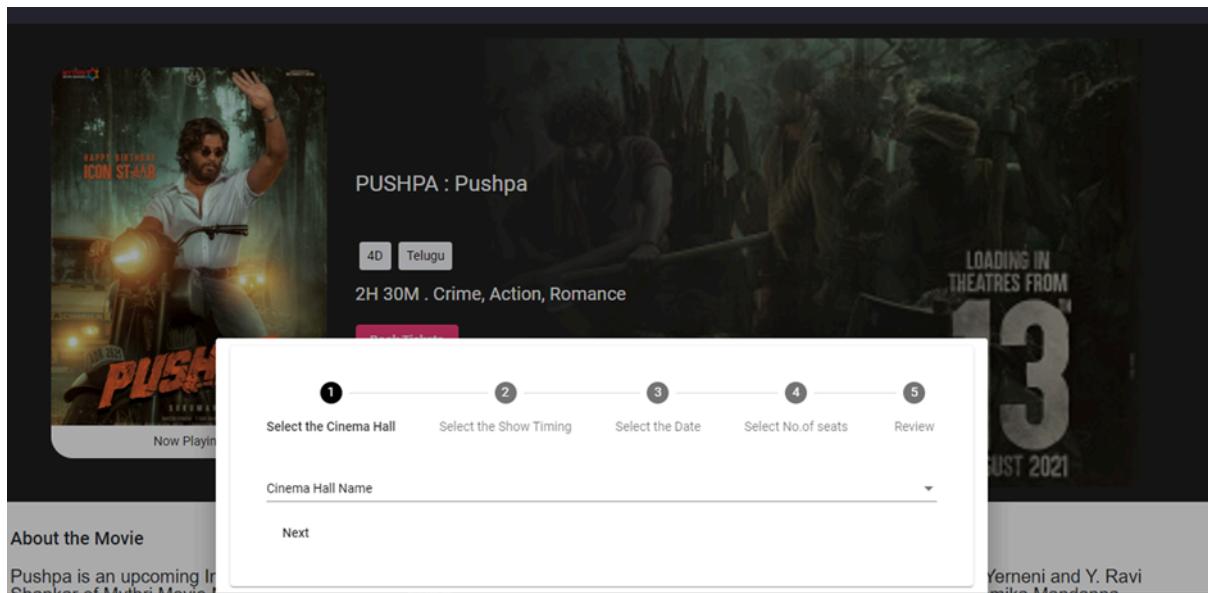
LOADING IN THEATRES FROM

13
AUGUST 2021

About the Movie

Pushpa is an upcoming Indian Telugu-language action thriller film written and directed by Sukumar. Produced by Neveen Venkani and V. Ravi.

8. Booking/Purchasing Process



9. Payment Details

A screenshot of a payment gateway page. At the top, it says "Payment Gateway (do not refresh the page)". Below that is a "Booking Details" section with the following information:

Auditorium	INOX
Show	First Show
Show Timing	06:00 PM
Movie Name	PUSHPA
Movie Lang	Telugu
Date	Jun 15, 2021

To the right of the booking details is a card payment form:

Card Number		
16 digit card number		
Month	Year	CVV
Select card expiry month	select card expiry year	(optional)
Pay ₹2,200.00		

10. Register New Auditorium

REGISTER NEW AUDITORIUM HERE

1 Auditorium Details

Name	Image URL
Seating Capacity 100	Email Id
Customer Care Number	Address

Next

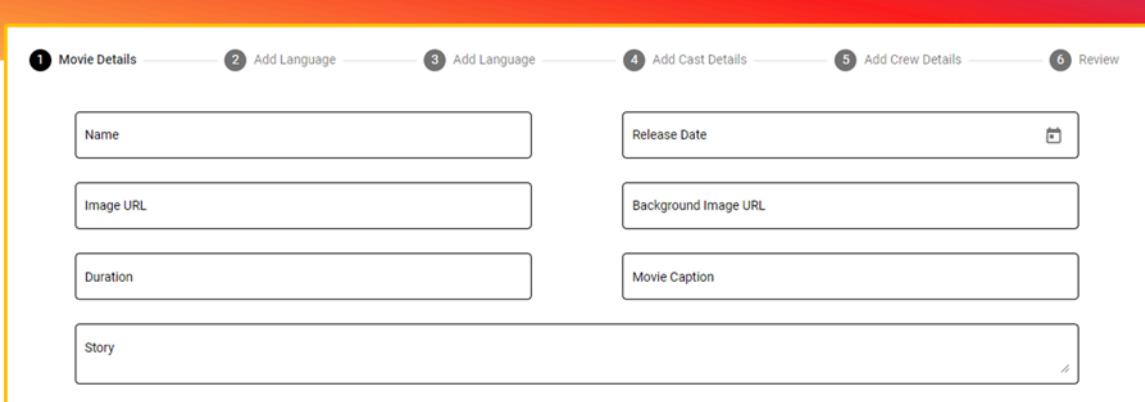


11. Add A New Movie

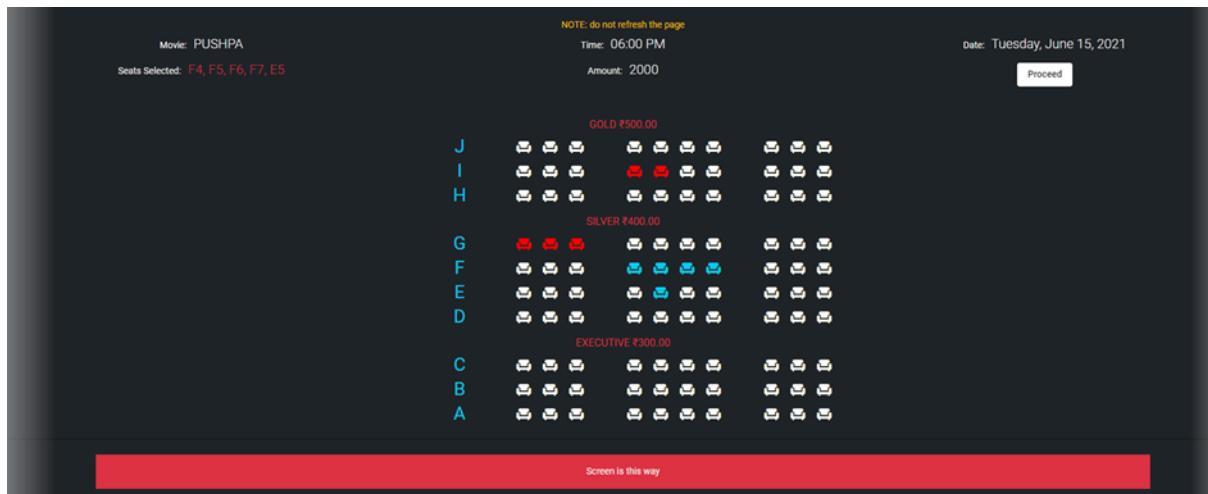
ADD NEW MOVIE

1 Movie Details 2 Add Language 3 Add Language 4 Add Cast Details 5 Add Crew Details 6 Review

Name	Release Date
Image URL	Background Image URL
Duration	Movie Caption
Story	



12. Seat Selection

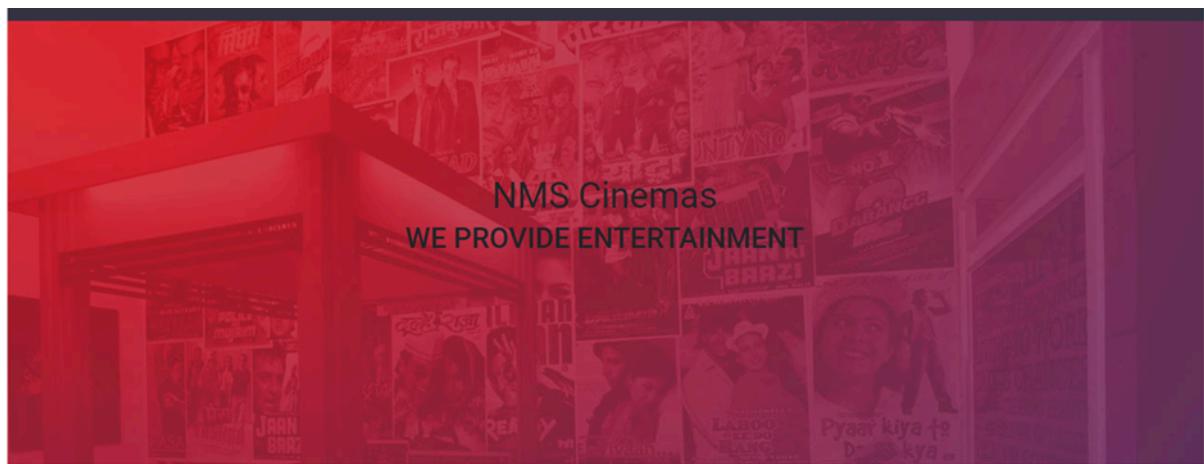


13. Booking Confirmation

The screenshot shows a booking confirmation interface for INOX. It features a header 'Your booking list' and two entries: 'Booking 1' (2021-06-10) and 'Booking 2' (2021-06-10). Below this is a movie poster for 'PUSHPA' featuring Allu Arjun. To the right is a table with booking details:

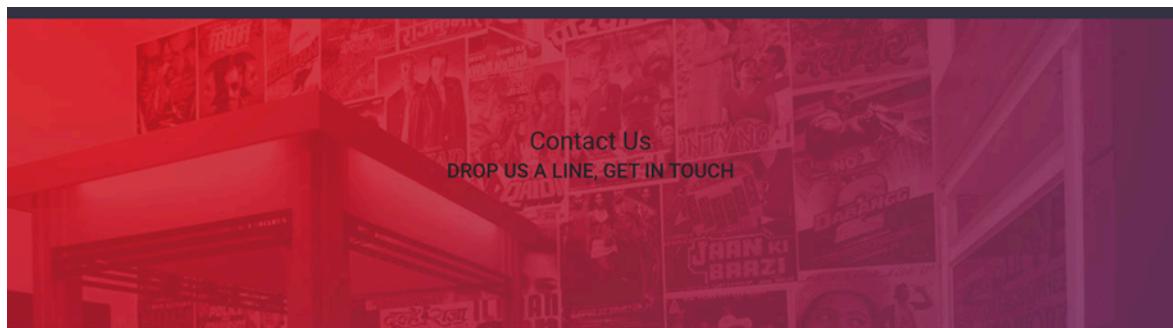
INOX	
Movie Language	Telugu
Total seats	5
Seat No.	F4, F5, F6, F7, E5
Price	₹2,000.00
Date	Jun 15, 2021
Time	06:00 PM
Status	Confirmed

14. About Us Info



About Us

15. Contact Us



Write to us by filling in the form below

Contact Us
DROP US A LINE, GET IN TOUCH

Your Name

Email Id

Message

[Contact Us](#)

Contact Us

Big Tree Entertainment Private Limited Ground Floor, Wajeda House,
Gulmohar Cross Road No. 7, Juhu Scheme, Mumbai, Maharashtra -
400049

nms@cinemas.com

[1800-894-4059](tel:18008944059)

Git Repository:

[Boigantso/CapstoneFullStack \(github.com\)](https://Boigantso/CapstoneFullStack)

AWS Console:

AWS Management Console

AWS services

- Recently visited services
 - EC2
- All services

Build a solution
Get started with simple wizards and automated workflows.

Launch a virtual machine With EC2 2-3 minutes	Build a web app With Elastic Beanstalk 6 minutes	Build using virtual servers With Lightsail 1-2 minutes	Register a domain With Route 53 3 minutes

Connect an IoT device
With AWS IoT
5 minutes

Start migrating to AWS
With AWS MGN
1-2 minutes

Start a development project
With CodeStar
5 minutes

Deploy a serverless microservice
With Lambda, API Gateway
2 minutes

Stay connected to your AWS resources on-the-go
AWS Console Mobile App now supports four additional regions. Download the AWS Console Mobile App to your iOS or Android mobile device. Learn more [\[link\]](#)

Explore AWS

Modernize Your APIs with GraphQL
AWS AppSync is a fully-managed GraphQL service that improves app performance and developer productivity. Learn more [\[link\]](#)

AWS Backup
Centrally manage and automate backups across AWS services. Learn more [\[link\]](#)

AWS Certification
Discover the top 7 reasons to get AWS Certified. Learn more [\[link\]](#)

Announcing Incident Manager from AWS Systems Manager
Respond faster to application issues using new incident response and analysis capabilities. Learn more [\[link\]](#)

Have feedback?

AWS Management Console

AWS services

- Recently visited services
 - EC2
- All services

Build a solution
Get started with simple wizards and automated workflows.

Launch a virtual machine With EC2 2-3 minutes	Build a web app With Elastic Beanstalk 6 minutes	Build using virtual servers With Lightsail 1-2 minutes	Register a domain With Route 53 3 minutes

Connect an IoT device
With AWS IoT
5 minutes

Start migrating to AWS
With AWS MGN
1-2 minutes

Start a development project
With CodeStar
5 minutes

Deploy a serverless microservice
With Lambda, API Gateway
2 minutes

Stay connected to your AWS resources on-the-go
AWS Console Mobile App now supports four additional regions. Download the AWS Console Mobile App to your iOS or Android mobile device. Learn more [\[link\]](#)

Explore AWS

Modernize Your APIs with GraphQL
AWS AppSync is a fully-managed GraphQL service that improves app performance and developer productivity. Learn more [\[link\]](#)

AWS Backup
Centrally manage and automate backups across AWS services. Learn more [\[link\]](#)

AWS Certification
Discover the top 7 reasons to get AWS Certified. Learn more [\[link\]](#)

Announcing Incident Manager from AWS Systems Manager
Respond faster to application issues using new incident response and analysis capabilities. Learn more [\[link\]](#)

Have feedback?

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. Learn more about Amazon EC2 security groups.

Assign a security group: Create a new security group Select an existing security group

Security group name: launch-wizard-2

Description: launch-wizard-2 created 2021-06-07T11:30:07.135+05:30

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
All traffic	All	0 - 65535	Anywhere 0.0.0.0/0	e.g. SSH for Admin Desktop

Warning
Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Step 7: Review Instance Launch

AMI Details

Amazon Linux 2 AMI (HVM), SSD Volume Type - ami-0d5ef0f6fb40b4de

Instance Type

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage
t2.micro	-	1	1	SSD

Security Groups

Security group name	Description
launch-wizard-2	launch-wizard-2 created 2021-06-07T11:30:07.135+05:00

Instance Details

Storage

Tags

Select an existing key pair or create a new key pair

A key pair consists of a public key that AWS stores, and a private key file that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about removing existing key pairs from a public AMI.

Create a new key pair
Key pair: my-key-pair
my-movie-plan.pem
Download Key Pair

You have to download the private key file (*.pem file) before you can continue. Store it in a secure and accessible location. You will not be able to download the file again after it's created.

Cancel Launch Instance

Cancel Previous Launch

New EC2 Experience

EC2 Dashboard

Events

Tags

Limits

Instances

Launch Templates

Spot Requests

Savings Plans

Reserved Instances

Dedicated Hosts

Scheduled Instances

Capacity Reservations

Images

AMIs

Elastic Block Store

Volumes

Snapshots

Lifecycle Manager

Network & Security

Instances (1/1) Info

Filter instances

Instance state: running

Clear filters

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
i-0b1bd5979f151bb1f	i-0b1bd5979f151bb1f	Running	t2.micro	Initializing	No alarms	+ us-east-1d	ec2-3-87-23-8.compute...	3.87.23.8	-

Connect Instance state Actions Launch Instance

EC2 > Instances > i-0e19201d262ba77e5 > Connect to instance

Connect to instance Info

Connect to your instance i-0e19201d262ba77e5 using any of these options

EC2 Instance Connect Session Manager **SSH client** EC2 Serial Console

Instance ID: i-0e19201d262ba77e5

- Open an SSH client.
- Locate your private key file. The key used to launch this instance is my-movie-plan.pem
- Run this command, if necessary, to ensure your key is not publicly viewable.
`chmod 400 my-movie-plan.pem`
- Connect to your instance using its Public DNS:
`ec2-52-7-134-66.compute-1.amazonaws.com`

Example:
`ssh -i "my-movie-plan.pem" ec2-user@ec2-52-7-134-66.compute-1.amazonaws.com`

Note: In most cases, the guessed user name is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI user name.

Cancel

Jenkins:

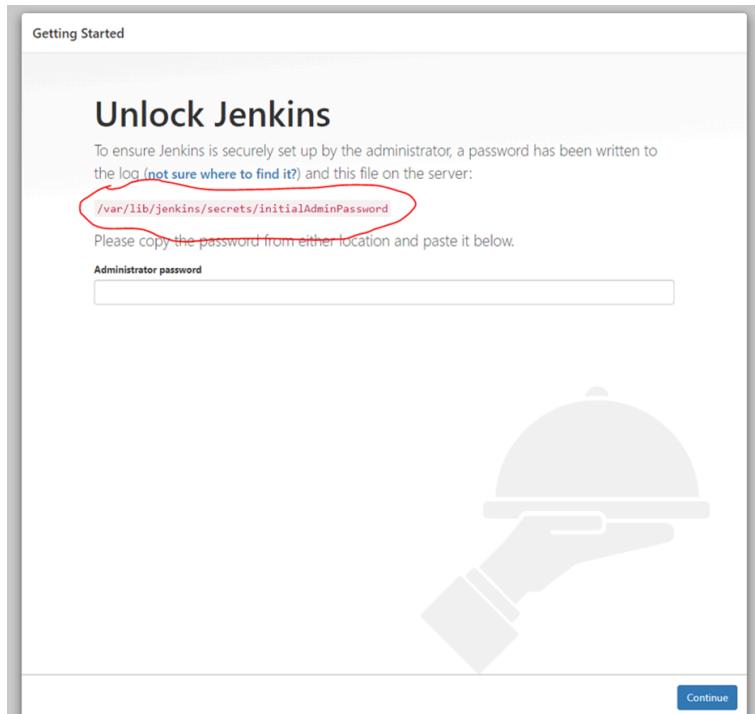
```
ssh -i "my-movie-plan.pem" ec2-user@ec2-52-7-134-66.compute-1.amazonaws.com
The authenticity of host 'ec2-52-7-134-66.compute-1.amazonaws.com (52.7.134.66)' can't be established.
ECDSA key fingerprint is SHA256:W2ndOKDsigx6WpnWePv9ya8umIsUNYw/ntPzLFvTxE.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-52-7-134-66.compute-1.amazonaws.com,52.7.134.66' (ECDSA) to the list of known hosts.

  _|_ _|_
  |(_ / Amazon Linux 2 AMI
  ---\_\_|\__|_>

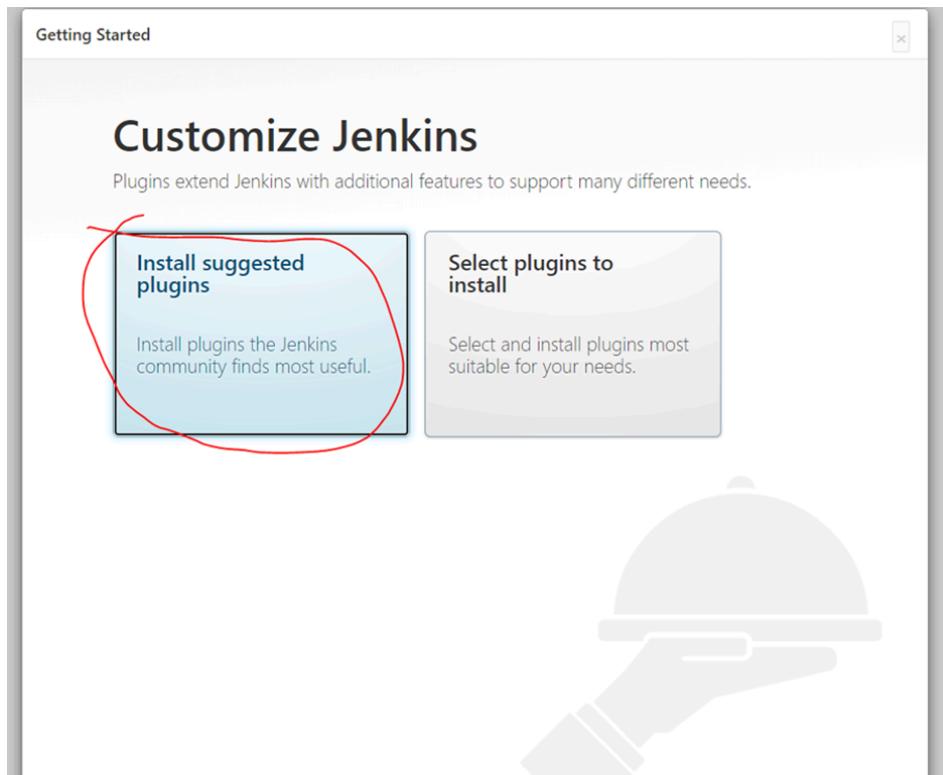
https://aws.amazon.com/amazon-linux-2/
1 package(s) needed for security, out of 17 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-86-0 ~]$ sudo yum update -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
--> Running transaction check
--> Package chrony.x86_64 0:3.5.1-1.amzn2.0.1 will be updated
--> Package chrony.x86_64 0:4.0-3.amzn2.0.1 will be an update
--> Package ec2-utils.noarch 0:1.2-43.amzn2 will be updated
--> Package ec2-utils.noarch 0:1.2-44.amzn2 will be an update
--> Package glibc.x86_64 0:2.26-44.amzn2 will be updated
--> Package glibc.x86_64 0:2.26-45.amzn2 will be an update
--> Package glibc-all-langpacks.x86_64 0:2.26-44.amzn2 will be updated
--> Package glibc-all-langpacks.x86_64 0:2.26-45.amzn2 will be an update
--> Package glibc-common.x86_64 0:2.26-44.amzn2 will be updated
--> Package glibc-common.x86_64 0:2.26-45.amzn2 will be an update
--> Package glibc-locale-source.x86_64 0:2.26-44.amzn2 will be updated
--> Package glibc-locale-source.x86_64 0:2.26-45.amzn2 will be an update
--> Package glibc-minimal-langpack.x86_64 0:2.26-44.amzn2 will be updated
--> Package glibc-minimal-langpack.x86_64 0:2.26-45.amzn2 will be an update
```

```
[root@ip-172-31-29-148 ~]# sudo systemctl start jenkins
[root@ip-172-31-29-148 ~]# sudo cat /var/lib/jenkins/secrets/initialAdminPassword
0811c483fdc84df7906bf37a2906754c
[root@ip-172-31-29-148 ~]# |
```

Unlock Jenkins:



Customise Jenkins:



Create:

The screenshot shows the Jenkins 'Create Item' page. At the top left, it says 'Jenkins'. Below that is the 'Dashboard > All >'. In the center, there is a form with a red circle around the input field where 'mysql-my-movie-plan' is typed. The input field has a placeholder 'Enter an item name' and a note 'Required field'. Below the input field, there is a list of project types: 'Freestyle project' (with a note about combining SCM and build systems), 'Pipeline' (which is circled in red), 'Multi-configuration project' (for testing across environments), 'Folder' (for grouping items), 'GitHub Organization' (for scanning GitHub organizations), 'GitHub Organization' (another entry for GitHub organizations), and 'Multibranch Pipeline' (for detecting branches in one repository). At the bottom of the form, there is a blue 'OK' button with a red circle around it.

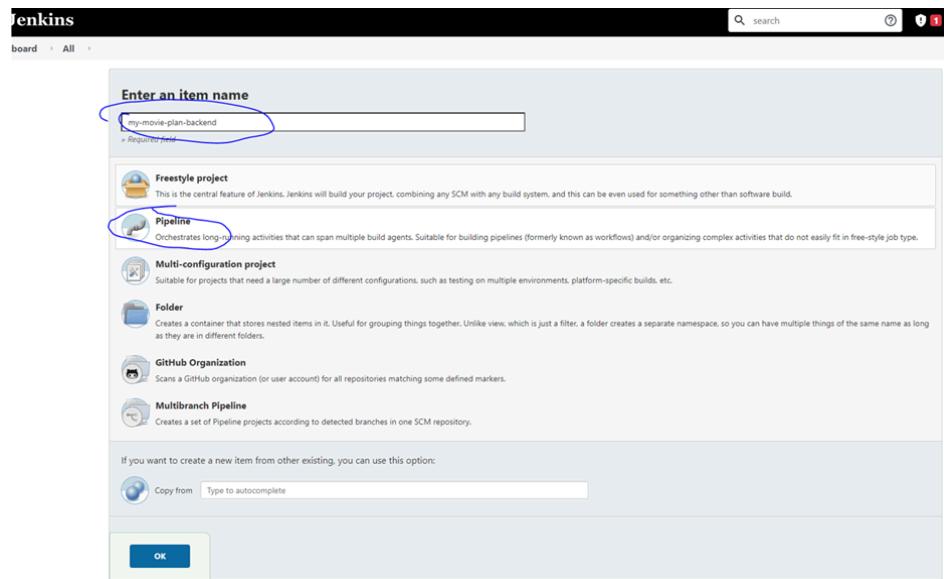
Docker 1:

```
43 napixyz      available  [ =stable ]
46 collectd      available  [ =stable ]
47 aws-nitro-enclaves-cli  available  [ =stable ]
48 R4           available  [ =stable ]
49 kernel-5.4    available  [ =stable ]
50 selinux-ng    available  [ =stable ]
51 php8.0        available  [ =stable ]
52 tomcat9       available  [ =stable ]
53 unbound1.13   available  [ =stable ]
54 mariadb10.5   available  [ =stable ]
55 kernel-5.10   available  [ =stable ]
56 redis6        available  [ =stable ]
ec2-user@ip-172-31-86-0 ~]$ sudo systemctl start docker
ec2-user@ip-172-31-86-0 ~]$
ec2-user@ip-172-31-86-0 ~]$
ec2-user@ip-172-31-86-0 ~]$ sudo usermod -a -G docker ec2-user
ec2-user@ip-172-31-86-0 ~]$
ec2-user@ip-172-31-86-0 ~]$
[ec2-user@ip-172-31-86-0 ~]$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
[ec2-user@ip-172-31-86-0 ~]$
```

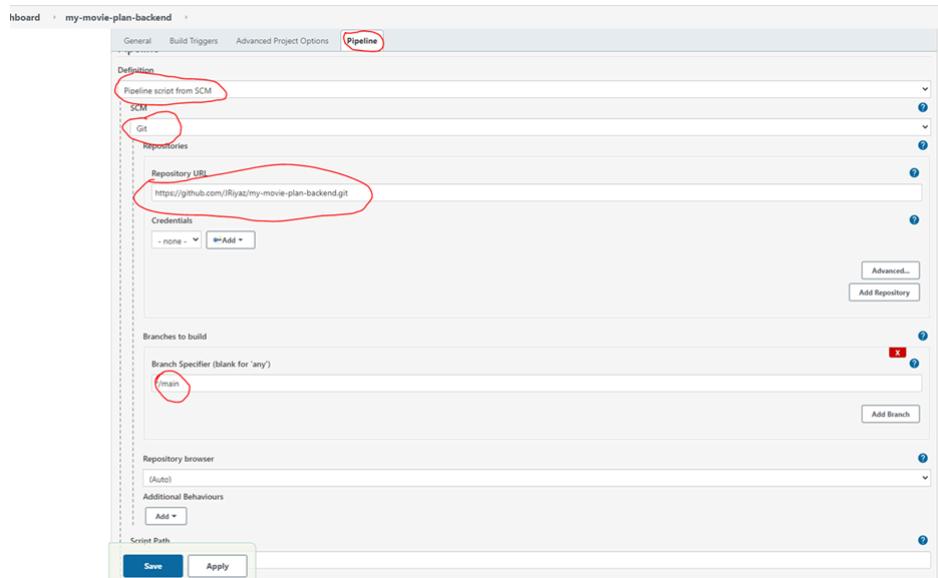
Docker 2:

```
47 aws-nitro-enclaves-cli  available  [ =stable ]
48 R4           available  [ =stable ]
49 kernel-5.4    available  [ =stable ]
50 selinux-ng    available  [ =stable ]
51 php8.0        available  [ =stable ]
52 tomcat9       available  [ =stable ]
53 unbound1.13   available  [ =stable ]
54 mariadb10.5   available  [ =stable ]
55 kernel-5.10   available  [ =stable ]
56 redis6        available  [ =stable ]
ec2-user@ip-172-31-86-0 ~]$ sudo systemctl start docker
ec2-user@ip-172-31-86-0 ~]$
ec2-user@ip-172-31-86-0 ~]$
ec2-user@ip-172-31-86-0 ~]$ sudo usermod -a -G docker ec2-user
ec2-user@ip-172-31-86-0 ~]$
ec2-user@ip-172-31-86-0 ~]$
ec2-user@ip-172-31-86-0 ~]$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
ec2-user@ip-172-31-86-0 ~]$
ec2-user@ip-172-31-86-0 ~]$
ec2-user@ip-172-31-86-0 ~]$
ec2-user@ip-172-31-86-0 ~]$
ec2-user@ip-172-31-86-0 ~]$
ec2-user@ip-172-31-86-0 ~]$
ec2-user@ip-172-31-86-0 ~]$ sudo usermod -a -G docker jenkins
ec2-user@ip-172-31-86-0 ~]$ |
```

Jenkins Details:



Backend:



Global Tool Configuration:

The screenshot shows the Jenkins Global Tool Configuration page. At the top, there are tabs for 'Dashboard' and 'Global Tool Configuration'. The 'Global Tool Configuration' tab is active and highlighted with a blue circle. Below it, there are sections for 'Gradle' and 'Ant'. The 'Maven' section is expanded, showing a table with one row. The row contains the name 'maven', the type 'MAVEN_HOME', the value '/usr/share/maven', and a checkbox labeled 'Install automatically'. There are 'Add Maven' and 'Delete Maven' buttons at the bottom of the table. At the very bottom of the page are 'Save' and 'Apply' buttons.

Pipeline:

The screenshot shows the Jenkins Pipeline creation interface. At the top, there is a search bar and a 'Jenkins' logo. Below it, a 'Dashboard' link and an 'All' link are visible. A large input field labeled 'Enter an item name' contains the text 'my-movie-plan-backend', which is circled in blue. Below this, there is a list of project types: 'Freestyle project', 'Pipeline', 'Multi-configuration project', 'Folder', 'GitHub Organization', and 'Multibranch Pipeline'. The 'Pipeline' option is highlighted with a blue circle. At the bottom of the screen, there is a 'Copy from' dropdown and a large blue 'OK' button.

Triggers:

The screenshot shows the Jenkins Project Configuration page for 'my-movie-plan-backend'. The 'Build Triggers' section is active and highlighted with a blue circle. It contains a checkbox for 'Build after other projects are built' and a dropdown menu where 'myself-my-movie-plan' is selected. Other options in the dropdown include 'Trigger only if build is stable', 'Trigger even if the build is unstable', and 'Trigger even if the build fails'. Below this, there are sections for 'Advanced Project Options' and 'Pipeline'. The 'Pipeline' section includes a 'Definition' dropdown set to 'Pipeline script from SCM' and a 'SCM' tree view with 'Git' and 'Repositories' branches. At the bottom of the page are 'Save' and 'Apply' buttons.

Source Code:

```
package com.example.MyMoviePlan.config;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

@Component
public class SimpleCORSFilter implements Filter{

    private final Logger log = LoggerFactory.getLogger(SimpleCORSFilter.class);

    public SimpleCORSFilter() {
        log.info("SimpleCORSFilter init");
    }

    @Override
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain
chain)
            throws IOException, ServletException {
```

```
HttpServletRequest request = (HttpServletRequest) req;
HttpServletResponse response = (HttpServletResponse) res;
response.setHeader("Access-Control-Allow-Origin",
request.getHeader("Origin"));
response.setHeader("Access-Control-Allow-Credentials", "true");
response.setHeader("Access-Control-Allow-Methods", "POST, PUT,
GET, OPTIONS, DELETE");
response.setHeader("Access-Control-Max-Age", "3600");
response.setHeader("Access-Control-Allow-Headers", "Content-Type,
Accept, X-Requested-With, remember-me");
chain.doFilter(req, res);
}
@Override
public void init(FilterConfig filterConfig) {
}
@Override
public void destroy() {
}
}
```

1. Cart Controller:

```
package com.example.MyMoviePlan.controller;

import java.util.List;

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

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

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.example.MyMoviePlan.model.Cart;

import com.example.MyMoviePlan.model.MovieTicket;

import com.example.MyMoviePlan.repository.CartRepository;

@RestController

@RequestMapping("/api/v1/cart")

public class CartController {

    @Autowired

    CartRepository cartRepository;

    @GetMapping("/movieTickets/all")

    public List<Cart> getAllMovieTickets() {

        return cartRepository.findAll();
```

```
}
```

```
@PostMapping("/movieTickets/add")
```

```
public Cart addMovieTicketToCart(@RequestBody(required = false) Cart cart)
```

```
{
```

```
    return cartRepository.save(cart);
```

```
}
```

```
@DeleteMapping("/movieTickets/delete/{id}")
```

```
public void deleteMovieTicketFromCart(@PathVariable Long id) {
```

```
    cartRepository.deleteById(id);
```

```
}
```

```
@DeleteMapping("/movieTickets/delete/all")
```

```
public void deleteAllMovieTickets() {
```

```
    cartRepository.deleteAll();
```

```
}
```

```
}
```

2. Movie Ticket Controller:

```
package com.example.MyMoviePlan.controller;

import java.util.List;

import java.util.Optional;

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

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.example.MyMoviePlan.model.MovieTicket;

import com.example.MyMoviePlan.repository.MovieTicketRepository;

@RestController

@RequestMapping("/api/v1/movieTicket")

public class MovieTicketController {

    @Autowired

    MovieTicketRepository movieTicketRepository;

    @GetMapping("/movieTickets/all")

    public List<MovieTicket> getAllMovieTickets() {
```

```
        return movieTicketRepository.findAll();

    }

    @GetMapping("/movieTickets/getAParticularMovie/{movieName}")

    public Optional<MovieTicket> getMovieTicketByMovieName(@PathVariable
String movieName) {

        return
movieTicketRepository.getMovieTicketByMovieName(movieName);

    }

    @GetMapping("/movieTickets/getAParticularMovieById/{id}")

    public MovieTicket getMovieTicketById(@PathVariable Long id) {

        return movieTicketRepository.getById(id);

    }

    @PostMapping("/movieTickets/add")

    public MovieTicket addMovieTicket(@RequestBody(required = false)
MovieTicket movieTicket) {

        return movieTicketRepository.save(movieTicket);

    }

    @PutMapping("/movieTickets/update")

    public MovieTicket updateMovieTicket(@RequestBody(required = false)
MovieTicket movieTicket) {

        return movieTicketRepository.save(movieTicket);

    }
```

```

    @DeleteMapping("/movieTickets/delete/{id}")

    public void deleteMovieTicket(@PathVariable Long id) {

        movieTicketRepository.deleteById(id);

    }

}

```

3. Movie Ticket Search Controller:

```

package com.example.MyMoviePlan.controller;

import java.util.List;

import java.util.Optional;

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

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;

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.example.MyMoviePlan.model.MovieTicketSearchResults;

import com.example.MyMoviePlan.repository.MovieTicketSearchResultsRepository;

@RestController
@RequestMapping("/api/v1/movieTicketSearchResults")

public class MovieTicketSearchResultsController {

```

```
@Autowired

private MovieTicketSearchResultsRepository
movieTicketSearchResultsRepository;

@GetMapping("/movieTicket/search/results/all")

public List<MovieTicketSearchResults> getMovieTicketByMovieName() {

    return movieTicketSearchResultsRepository.findAll();

}

@PostMapping("/movieTicket/search/results/add")

public MovieTicketSearchResults createMovieTicketSearchResult(


    @RequestBody(required = false) String
movieTicketSearchResults) {

    MovieTicketSearchResults movieTicketSearchResults1 = new
MovieTicketSearchResults();

    movieTicketSearchResults1.setMovieName(movieTicketSearchResults);

    return
movieTicketSearchResultsRepository.save(movieTicketSearchResults1);

}

@DeleteMapping("/movieTicket/search/results/delete/")

public void deleteAllMovieTicketSearchResults() {

    movieTicketSearchResultsRepository.deleteAll();

}

}
```

4. User Controller:

```
package com.example.MyMoviePlan.controller;

import java.util.List;

import java.util.Optional;

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

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

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.example.MyMoviePlan.model.Cart;

import com.example.MyMoviePlan.model.User;

import com.example.MyMoviePlan.repository.UserRepository;

@RestController

@RequestMapping("/api/v1/user")

public class UserController {

    @Autowired

    UserRepository userRepository;
```

```
@PostMapping("/users/add")
public User addUser(@RequestBody(required = false) User user) {
    return userRepository.save(user);
}

@GetMapping("/users/getParticularUser/{id}")
public Optional<User> getUserById(@PathVariable Long id) {
    return userRepository.findById(id);
}

@GetMapping("/users/getAParticularUser/details")
public Optional<User> getUserDetails(@RequestBody(required = false) User user) {
    return userRepository.getUserDetails(user.getId(), user.getFirstName(),
        user.getLastName(),
        user.getUsername(), user.getEmail());
}

@GetMapping("/users/all")
public List<User> getAllUsers() {
    return userRepository.findAll();
}
```

5. Cart Model:

```
package com.example.MyMoviePlan.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "cart")
public class Cart {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private Long id;

    @Column(name = "movie_name")
    private String movieName;

    @Column(name = "show_date")
    private String showDate;

    @Column(name = "show_time")
    private String showTime;
```

```
@Column(name = "showing_location")
private String showingLocation;

@Column(name = "price")
private Long price;

public Cart() {}

public Cart(String movieName, String showDate, String showTime, String
showingLocation, Long price) {
    super();
    this.movieName = movieName;
    this.showDate = showDate;
    this.showTime = showTime;
    this.showingLocation = showingLocation;
    this.price = price;
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getMovieName() {
    return movieName;
```

```
}

public void setMovieName(String movieName) {

    this.movieName = movieName;

}

public String getShowDate() {

    return showDate;

}

public void setShowDate(String showDate) {

    this.showDate = showDate;

}

public String getShowTime() {

    return showTime;

}

public void setShowTime(String showTime) {

    this.showTime = showTime;

}

public String getShowingLocation() {

    return showingLocation;

}

public void setShowingLocation(String showingLocation) {

    this.showingLocation = showingLocation;

}

public Long getPrice() {

    return price;

}
```

```
public void setPrice(Long price) {  
    this.price = price;  
}  
  
}
```

6. Movie Ticket Model:

```
package com.example.MyMoviePlan.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
@Entity  
@Table(name = "movie_tickets")  
public class MovieTicket {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    @Column(name = "id")
```

```
private Long id;

@Column(name = "movie_name")
private String movieName;

@Column(name = "show_date")
private String showDate;

@Column(name = "show_time")
private String showTime;

@Column(name = "showing_location")
private String showingLocation;

@Column(name = "price")
private Long price;

public MovieTicket() {}

public MovieTicket(String movieName, String showDate, String showTime,
String showingLocation,
Long price) {
    super();
    this.movieName = movieName;
    this.showDate = showDate;
    this.showTime = showTime;
```

```
    this.showingLocation = showingLocation;
    this.price = price;
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getMovieName() {
    return movieName;
}

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

public String getShowDate() {
    return showDate;
}

public void setShowDate(String showDate) {
    this.showDate = showDate;
}

public String getShowTime() {
    return showTime;
}

public void setShowTime(String showTime) {
```

```
        this.showTime = showTime;  
    }  
  
    public String getShowingLocation() {  
        return showingLocation;  
    }  
  
    public void setShowingLocation(String showingLocation) {  
        this.showingLocation = showingLocation;  
    }  
  
    public Long getPrice() {  
        return price;  
    }  
  
    public void setPrice(Long price) {  
        this.price = price;  
    }  
  
}
```

7. Movie Ticket Search Result Model:

```
package com.example.MyMoviePlan.model;  
  
import javax.persistence.Column;  
  
import javax.persistence.Entity;  
  
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name = "movie_ticket_search_results")
public class MovieTicketSearchResults {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private long id;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    @Column(name = "movie_name")
    private String movieName;

    public MovieTicketSearchResults() {}

    public MovieTicketSearchResults(String movieName) {
        super();
        this.movieName = movieName;
    }
}
```

```
    }

    public String getMovieName() {
        return movieName;
    }

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

8. User Model:

```
package com.example.MyMoviePlan.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "users")

public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
@Column(name = "id")
```

```
private Long id;
```

```
@Column(name = "first_name")
```

```
private String firstName;
```

```
@Column(name = "last_name")
```

```
private String lastName;
```

```
@Column(name = "username")
```

```
private String username;
```

```
@Column(name = "email")
```

```
private String email;
```

```
@Column(name = "password")
```

```
private String password;
```

```
public User() {}
```

```
public User(String firstName, String lastName, String username, String email,  
String password) {
```

```
    super();
```

```
    this.firstName = firstName;
```

```
    this.lastName = lastName;
```

```
    this.username = username;
```

```
        this.email = email;  
  
        this.password = password;  
    }  
  
    public Long getId() {  
        return id;  
    }  
  
    public void setId(Long id) {  
        this.id = id;  
    }  
  
    public String getFirstName() {  
        return firstName;  
    }  
  
    public void setFirstName(String firstName) {  
        this.firstName = firstName;  
    }  
  
    public String getLastName() {  
        return lastName;  
    }  
  
    public void setLastName(String lastName) {  
        this.lastName = lastName;  
    }  
  
    public String getUsername() {  
        return username;  
    }  
  
    public void setUsername(String username) {
```

```
        this.username = username;  
    }  
  
    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;  
    }  
  
}
```

9. Cart Repository:

```
package com.example.MyMoviePlan.repository;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import org.springframework.stereotype.Repository;  
  
import com.example.MyMoviePlan.model.Cart;
```

```
@Repository  
public interface CartRepository extends JpaRepository<Cart, Long> {  
  
}
```

10. Movie Ticket Repository:

```
package com.example.MyMoviePlan.repository;  
  
import java.util.Optional;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import org.springframework.data.jpa.repository.Query;  
  
import org.springframework.data.repository.query.Param;  
  
import org.springframework.stereotype.Repository;  
  
import com.example.MyMoviePlan.model.MovieTicket;  
  
@Repository  
public interface MovieTicketRepository extends JpaRepository<MovieTicket, Long> {  
  
    @Query(value = "SELECT * FROM movie_tickets WHERE movie_name = :movieName", nativeQuery = true)  
    public Optional<MovieTicket>  
    getMovieTicketByMovieName(@Param("movieName") String movieName);  
  
}
```

11. Movie Ticket Search Result Repository:

```
package com.example.MyMoviePlan.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import com.example.MyMoviePlan.model.MovieTicketSearchResults;

@Repository
public interface MovieTicketSearchResultsRepository extends
JpaRepository<MovieTicketSearchResults, Long> {

    @Query(value = "SELECT * FROM movie_ticket_search_results WHERE
movieName = :movieName", nativeQuery = true)

    public Optional<MovieTicketSearchResults>
getMovieTicketByMovieName(@Param("movieName") String movieName);

}
```

12. End-User Repository:

```
package com.example.MyMoviePlan.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
```

```

import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import com.example.MyMoviePlan.model.User;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    @Query(value = "SELECT * FROM users WHERE id = :id AND first_name = :firstName AND "
            + "last_name = :lastName AND username = :username AND email = :email", nativeQuery = true)
    public Optional<User> getUserDetails(@Param("id") Long id,
                                         @Param("firstName") String firstName,
                                         @Param("lastName") String lastName,
                                         @Param("username") String username,
                                         @Param("email") String email);
}

```

13. Spring Boot Backend MyMovieApplication:

```

package com.example.MyMoviePlan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class SpringbootBackendMyMoviePlanApplication {
    public static void main(String[] args) {

```

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