How Is an Application Developed?

Developer Jim writes a method called `getStudentMarks()` that fetches all the marks of students from the database.

Developer Sam uses the output of `getStudentMarks()` to calculate the total of all the students and writes a method `calculateTotalMarks()`.

Developer Tom uses the output of `calculateTotalMarks()` to calculate the grade of all the students, produce a report, and write a method `calculateGrade()`.

Think and Tell

- Do you think the code is written in a modular manner?
- If Jim, the developer, makes changes in his code, do you think it will affect Sam's and Tom's code?
- If Sam changes the logic to calculate total marks, will Tom also have to change the logic for grading?
- If the logic needs to be changed, will they have to do the testing again for all the modules of the application or only the part that Sam changed?

Think and Tell

- The application is deployed after creation. A new requirement arises where the total marks scored by students are reduced from 100 to 80.
- Sam must now calculate the total marks in his method for 80, not 100.
- Will Tom have to change the grading logic of his method?
- How will the application get redeployed?
- Will the entire application be stopped and redeployed?

Think and Tell

- Tom wishes to store the data as it is in the database. Will he be able to do that, if the application uses a relational database?
- How will the marks in the subject be stored?
- If the database for the application changes from MySQL to MongoDB, will the developers have to change the code while performing database applications?

```
{
  "studentName":"John",
  "studentRollNo": 101,
  "studentMarksInSubjects":[80,90,80,90,80],
  "studentTotalMarks": 420,
  "grade":"A"
}
```

Slide Note

If it's a large application, only one functionality of shopping alone cannot be scaled up.

The other modules, even though they are not required to be scaled up, will also have to be scaled up if it is one large application.

This involves cost to the business.

Testing effort

Deployment effort

Downtime – Software as a service model, can systems be down?

All modules access the same large database. Integrity of the data is affected if changes are made by multiple modules. Rollback and testing will need to be done.

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?

Slide Note

A Java web application consists of a single WAR file that runs on a web container such as Tomcat.

Let's imagine that you are building a ride share application that helps commuters book cabs at anytime of the day or night.

The application consists of several components such as Passenger Management, Billing , Driver Management, Payment, Trip Management, Notifications, UI Management, etc.

The application is deployed as a single application.

What is the drawback of this approach of deployment?

Ride-share Application

Let us imagine that you are building a ride-sharing application that helps commuters book cabs at any time 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?

Menu

00:06:00:00:43

06/ 43

Slide Note

Create Microservices by Using Spring Boot

Slide Note

Learning Objectives

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

Slide Note

Monolithic Applications

Slide Note

Monolithic software is designed to be self-contained; components of the program are interconnected and interdependent. This is Ok for small sized applications where we are aware of the application size and functionality it provides. Monoliths are OK when we know there would be very little, or no changes in functionality of the application.

This monolith architecture has several benefits:

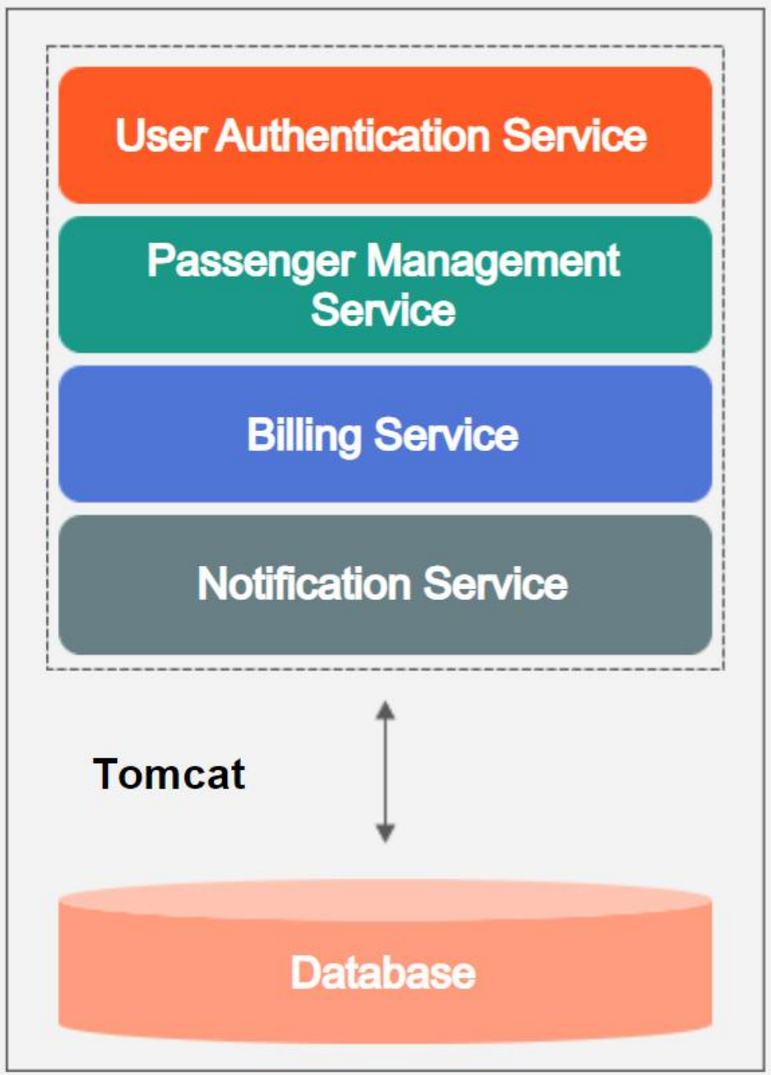
• First it is simple to develop – If you look around current development tools and IDEs, they are inherently designed to support the development of monolithic applications

• Second – It is simple to deploy - you simply need to deploy the WAR file on the appropriate runtime server.

• Also, it is simple to scale - you can scale the application by running multiple copies of the application behind a load balancer.

• So, if you have a small size application, where you expect little or no changes in functionality, monolith is the way to go.

But modern days application have far more dynamics requirements. Applications such as Facebook, Amazon, and Linkedin introduces new features every now and then.



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:
  - Self-contained, and independent from other computing applications.
  - Deployed as a single application. In case, it is a Java web application, it will have a single WAR file that runs on a web container like Tomcat.

Slide Note

A severe complexity is related to Change impact.

§ Small changes in an existing component can have rippling affect, and you might end up making big changes in the overall application.

§ This happens because, as I mentioned, in monolithic applications, components are interconnected and interdependent.

§ For example, let us for a minute consider the ride share service as a monolith application.

§ If we change the User Authentication Module to incorporate a new security features – OAuth2 by replacing current JWT authentication, all the interconnected Passenger Management, Notification etc. will require change – some amount or other.

§ Even the database schemas would change.

§ This meant as a rippling effect. Such a rippling effect is common in Monolithic architecture even for very minor updates.

§ At this point, two decisions can be made:

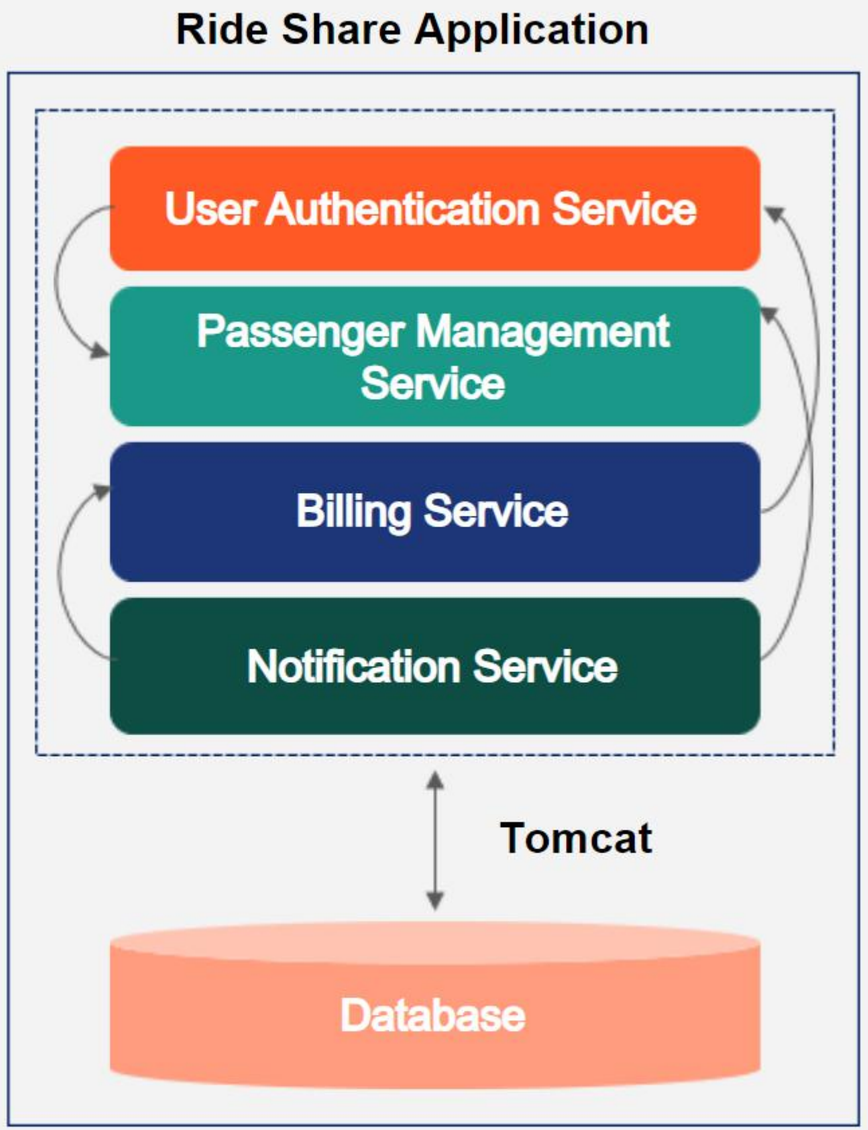
§ One, don't go for the change at all. You will find several legacy applications running with same old functionalities for years.

§ But you can't afford to do so in modern consumer-based applications.

§ New features must be continuously rolled out based on current business demands.

Monolithic Applications - Problems

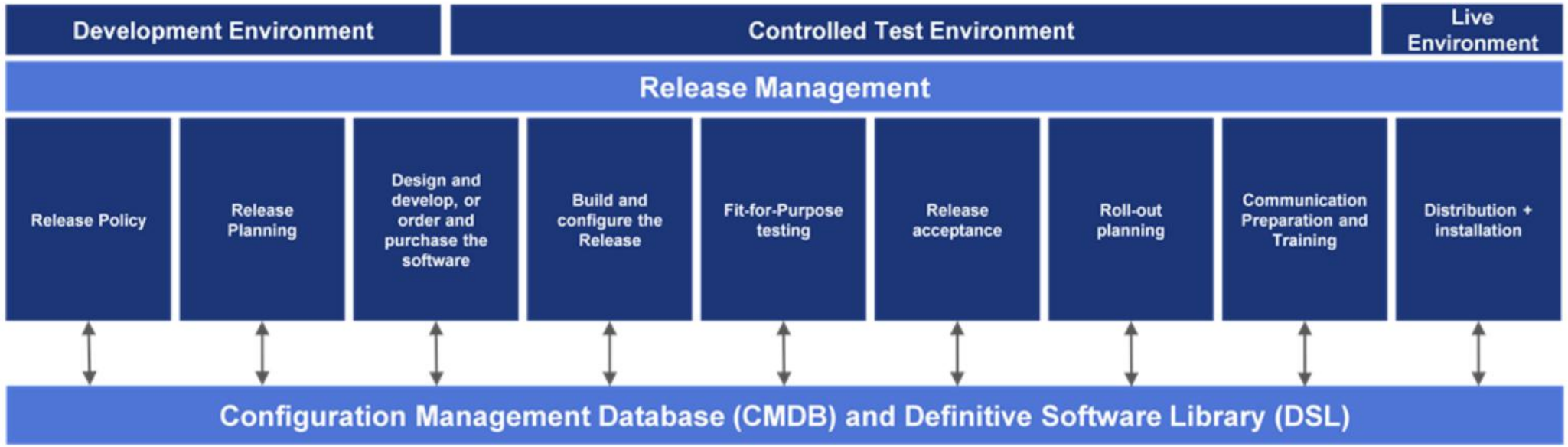
- The Ride Share application currently has four services.
- Imagine the complexity involved if we need to extend the number of services and add a Driver Management Service, Trip Management Service, etc., in the application.
- The actual problem is not the number of services present in the monolithic application, but the interactions that happen between them.
- A severe complexity is related to the change impact.
- Small changes in an existing component can have a ripple effect because the components are interconnected and dependent on one another.



Slide Note

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.



Slide Note

The Large monolithic code base intimidates developers, especially ones who are new to the team. The application can be difficult to understand and modify. As a result, development typically slows down. Also, because there are not hard module boundaries, modularity breaks down over time. Moreover, because it can be difficult to understand how to correctly implement a change, the quality of the code declines over time. It's a downwards spiral.

Overloaded IDE - The larger the code base, the slower the IDE and the less productive developers are.

Overloaded web container - The larger the application the longer it takes to start up. This has a huge impact on developer productivity because of the time wasted waiting for the container to start. It also impacts deployment.

Continuous deployment is difficult - A large monolithic application is also an obstacle to frequent deployments. In order to update one component, you must redeploy the entire application. This will interrupt background tasks (e.g., Quartz jobs in a Java application), regardless of whether they are impacted by the change, and possibly cause problems. There is also the chance that components that haven't been updated will fail to start correctly. As a result, the risk associated with redeployment increases, which discourages frequent updates. This is especially a problem for user interface developers, since they usually need to iterate rapidly and redeploy frequently.

Scaling the application can be difficult - A monolithic architecture is that it can only scale in one dimension. On the one hand, it can scale with an increasing transaction volume by running more copies of the application. Some clouds can even adjust the number of instances dynamically based on load. But on the other hand, this architecture can't scale with an increasing data volume. Each copy of application instance will access all the data, which makes caching less effective and increases memory consumption and I/O traffic. Also, different application components have different resource requirements - one might be CPU intensive while another might memory intensive. With a monolithic architecture we cannot scale each component independently

Obstacle to scaling development - A monolithic application is also an obstacle to scaling development. Once the application gets to a certain size, it's

Disadvantages of a Monolithic Approach

- Tight coupling between components
- Overloaded web container
- Large code base
- Less Scalable
- Obstacles in continuous deployment
- More deployment and restart times
- New technology barriers
- Long term commitment to a tech stack

Slide Note

a problem for user interface developers, since they usually need to iterate rapidly and redeploy frequently.

Scaling the application can be difficult - A monolithic architecture is that it can only scale in one dimension. On the one hand, it can scale with an increasing transaction volume by running more copies of the application. Some clouds can even adjust the number of instances dynamically based on load. But on the other hand, this architecture can't scale with an increasing data volume. Each copy of application instance will access all the data, which makes caching less effective and increases memory consumption and I/O traffic. Also, different application components have different resource requirements - one might be CPU intensive while another might memory intensive. With a monolithic architecture we cannot scale each component independently

Obstacle to scaling development - A monolithic application is also an obstacle to scaling development. Once the application gets to a certain size, it's useful to divide up the engineering organization into teams that focus on specific functional areas. For example, we might want to have the UI team, accounting team, inventory team, etc. The trouble with a monolithic application is that it prevents the teams from working independently. The teams must coordinate their development efforts and redeployments. It is much more difficult for a team to make a change and update production.

Requires a long-term commitment to a technology stack - A monolithic architecture forces you to be married to the technology stack (and in some cases, to a particular version of that technology) you chose at the start of development . With a monolithic application, it can be difficult to incrementally adopt a newer technology. For example, let's imagine that you chose the JVM. You have some language choices since as well as Java you can use other JVM languages that inter-operate nicely with Java, such as Groovy and Scala. But components written in non-JVM languages do not have a place within your monolithic architecture. Also, if your application uses a platform framework that subsequently becomes obsolete, then it can be challenging to incrementally migrate the application to a newer and better framework. It's possible that in order to adopt a newer platform framework you will have to rewrite the entire application, which is a risky undertaking.

Disadvantages of a Monolithic Approach

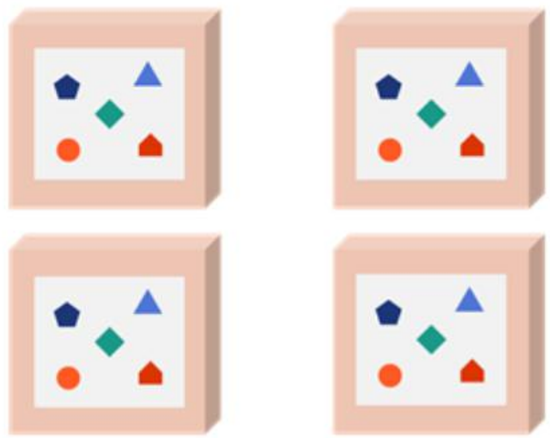
- Tight coupling between components
- Overloaded web container
- Large code base
- Less Scalable
- Obstacles in continuous deployment
- More deployment and restart times
- New technology barriers
- Long term commitment to a tech stack

Slide Note

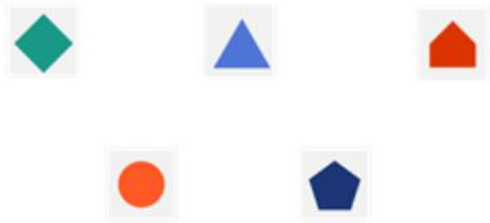
Shifting From Monolithic to Microservices

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

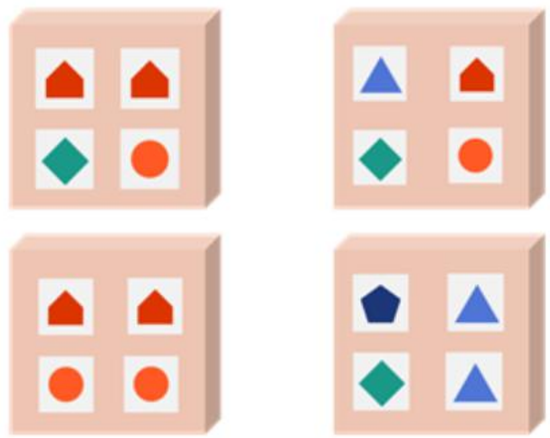
…and scales by replicating the monolith on multiple servers.



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



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



Slide Note

Microservices

Slide Note

"Microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API."

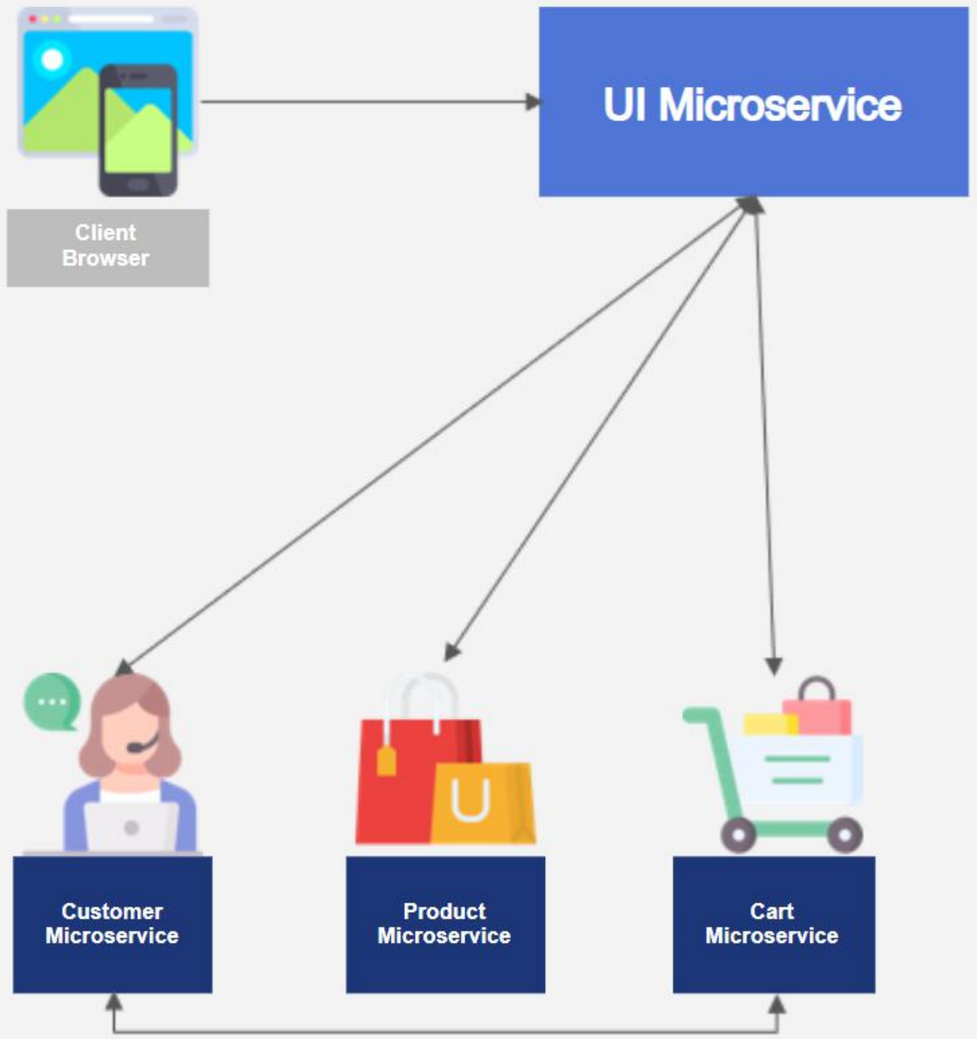
-Martin Fowler and James Lewis

A complex application can be built as a suite of independently deployable microservices.

Its architecture involves decomposing the complex business functionality into modular and granular microservices that can be quickly and continuously developed, deployed, and maintained.

Microservices

- Microservices are modular, autonomous, and logical units of functionality that are independently deployable and scalable.
- It is an architectural style, in which large complex software applications are composed of one or more services.
- Microservices can be deployed on a server independent of each other
- Each microservice focuses on completing only one task and does that one task well.
- A task represents a small business capability.



Slide Note

Modularity: Each microservice represents a logically cohesive, lightweight, and independent business functionality with well-defined boundaries. By design, microservices are highly granular, and independently built and deployed.

Single functionality principle: Each microservice encapsulates a single business functionality or use case.

Asynchronous invocation: Microservices are stateless by default, helping us to asynchronously invoke them to deliver high performance and high scalability.

Independent deployment and independent scalability: Each microservice can be independently deployable, and therefore, they are independently scaled to respond to varying workloads and user demands.

Self-containment: The microservice deployment unit is self-contained as it includes all dependent libraries, storage units, databases, and such. Microservices uses decentralized data management wherein a service-specific database is part of a microservice deployment unit/container.

Resiliency: Microservices architecture eliminates single point of failure through distribution of coherent functionality to various microservices. Even if a single service goes down, the application will continue to work. By leveraging the circuit breaker pattern, the fault tolerance can also be enhanced.

Advantages of Implementing Microservices

- **Modularity** - Each microservice demonstrates a logically cohesive, lightweight, and independent business functionality with well-defined boundaries.
- **Single functionality principle** - The principle of single functionality states that 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 a single point of failure through the distribution of coherent functionality among other services.

Slide Note

Loose coupling: Microservices are designed to be loosely coupled with minimal dependency on other services and libraries.

Smart endpoint and dumb pipes: Microservices communicate with each other with well-defined APIs (smart endpoints) over simple protocols such as REST over HTTP (dumb pipes).

Extensibility: Microservices 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 and still coexist in the application.

• **Faster release cycles** : Since microservices can be developed in parallel by multiple developers, the release time is less during product development.

Advantages of Implementing Microservices (contd.)

- 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.

Slide Note

• **Visibility** - Individual services may be developed by different groups of app developers and may be introduced into the application at any time. Developers need a way to discover services and their dependencies automatically as soon as they brought live. Application visibility tools should automatically display new services as they come up.

• Developers must deal with the additional complexity of creating a distributed system:

§ Developers must implement the inter-service communication mechanism and deal with partial failure

§ Implementing requests that span multiple services is more difficult

§ Testing the interactions between services is more difficult

§ Implementing requests that span multiple services requires careful coordination between the teams

§ Developer tools/IDEs are oriented on building monolithic applications and don't provide explicit support for developing distributed applications.

• **Deployment complexity** - In production, there is also the operational complexity of deploying and managing a system comprised of many different services.

• **Increased memory consumption** - The microservice architecture replaces N monolithic application instances with NxM services instances. If each service runs in its own JVM (or equivalent), which is usually necessary to isolate the instances, then there is the overhead of M times as many JVM runtimes. Moreover, if each service runs on its own VM (e.g., EC2 instance), as is the case at Netflix, the overhead is even higher.

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 is challenging to set the functional boundaries of each microservice.
- **Configuration management** – There is 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.

Slide Note

The client can be a mobile device or a desktop browser client.

The services can effortlessly communicate with one another.

If we want to change the Authentication type from JWT to Oauth, that can be done without changing other microservices in the application.

Change in DB schema of Billing service will not affect other services.

Microservice architecture is a method of enterprise software development, as a suite of independently deployable, small, modular services in which each service runs a unique process and communicates through a well-defined, lightweight mechanism to serve a business goal.

Microservices have been gaining ground in software development since the term came into existence. Microservice architecture is the variant of service-oriented architecture (SOA) for developing large applications where services are grained into chunks as per the business domain. It provides continuous delivery/deployment of complex applications and makes the application easier to understand, develop, test, and is more resilient to architecture erosion.

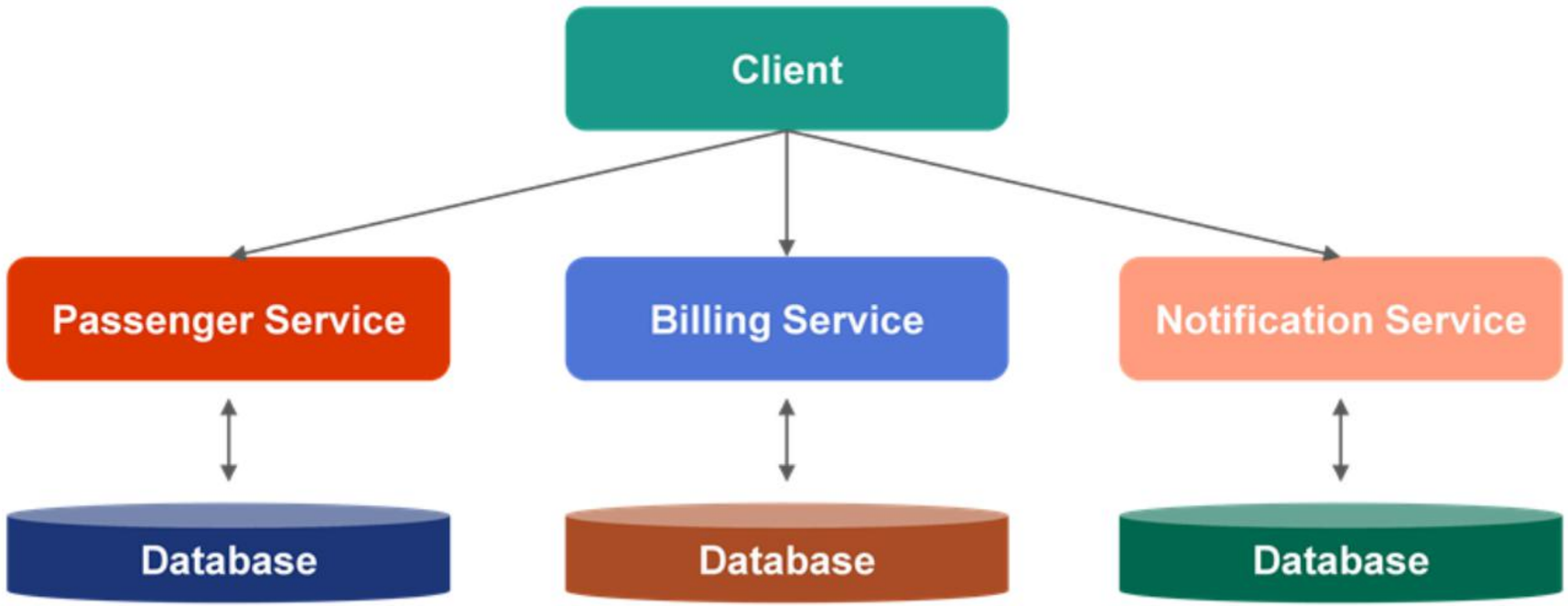
The Microservice architecture provides a new way to weave the existing system in novel ways in order to deliver software solutions quickly. Being one of the hottest topics in the software industry due to its ability to provide modularity, scalability, availability; many enterprise software development companies are keen to adopt it.

Also, Microservices architecture is suitable when it is required to support different types of clients, like desktop browsers, mobile browsers, and native mobile applications, and it is required to integrate with other applications via web services or message broker.

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 the ride share application as shown below:**



Slide Note

Quick Check

Microservices are _____ by default.

1. stateful
2. singleton
3. stateless
4. secure

Slide Note

Quick Check: Solution

Microservices are _____ by default.

1. stateful
2. singleton
3. **stateless**
4. secure

Slide Note

Developing Microservices

Slide Note

Steps to Create Microservices

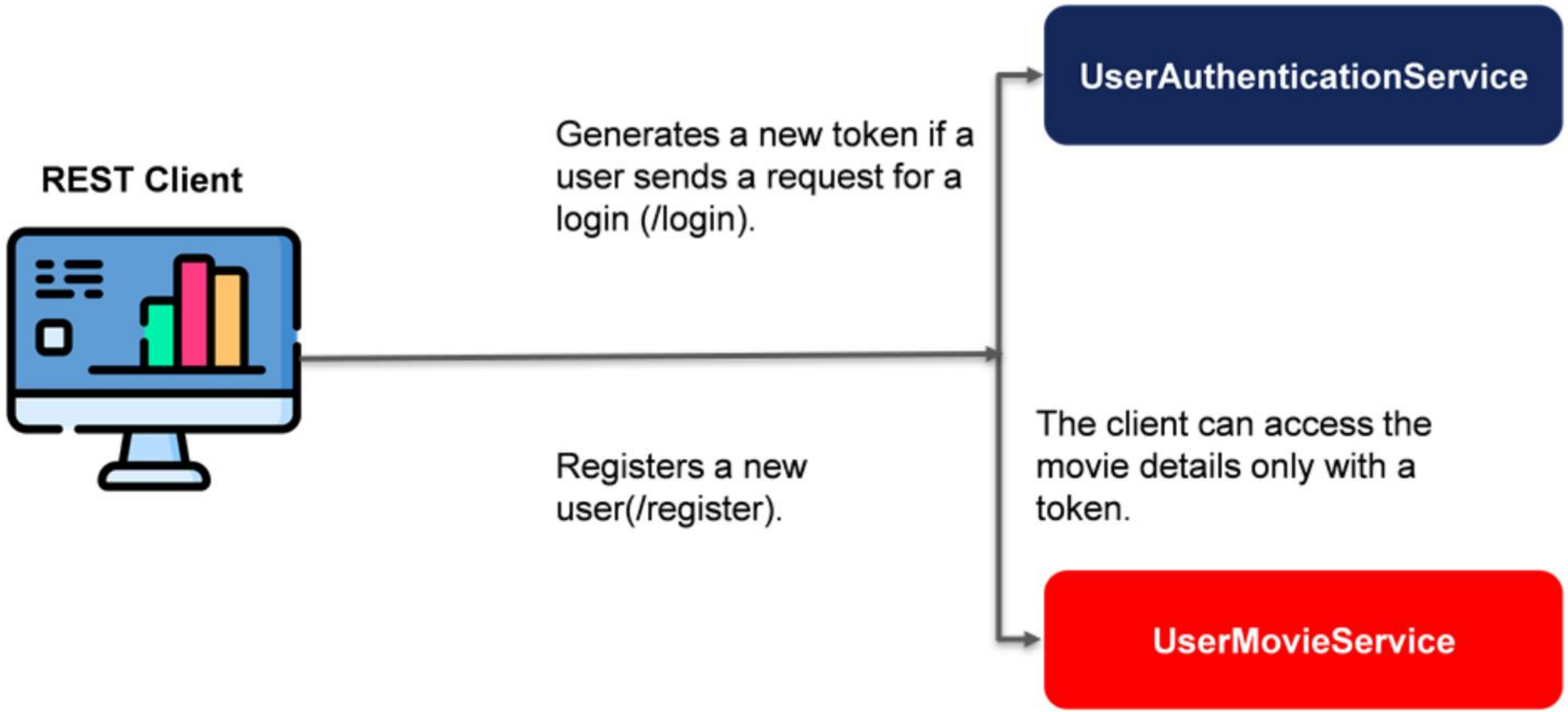
- The Spring Boot applications created during the earlier sessions such as `CustomerService`, `ProductService`, `UserService` are all individual applications that can be stitched together to form microservices.
- Consider a streaming application that enables users to watch movies on any smart device. The application provides multiple features to all its registered users.
  - 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 the user to the list of the user's favorite movie 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 is used for authentication purposes.

Slide Note

Only a registered user can access the data. Many applications use this token access pattern.

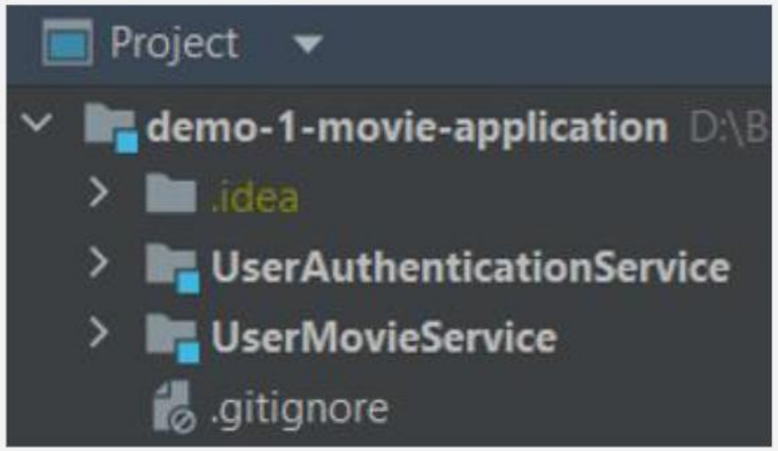
§ For example, during online shopping, only if you are a registered user, the application will allow you to make payment or place order.

Microservices – An Example



Slide Note

Multi-Module Project - Maven



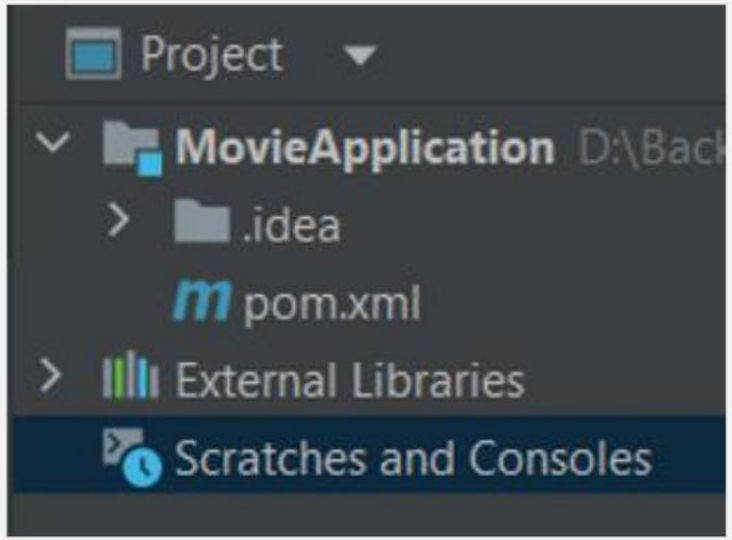
- The microservices together form a single application, thus it is logical to build a multi-module project using Maven.
- It is wise to use Maven's multi-module feature to logically separate services from each other because the project contains multiple services.
- This creates an explicit boundary between the modules or services to avoid using components that belong to other services.
- The movie application has multiple services like `UserAuthenticationService` and `UserMovieService` etc.
- Each of the services is a module of the movie application.

Slide Note

Creating a Multi-Module Application

- A multi-module application is like a directory which holds the different microservices as modules.
- To create a multi-module project, follow the steps given below:
  - Create a Maven quick start project.
  - Remove the `src` folder.
  - Move all the microservices created into the folder under the created project.
  - Declare the microservices under the modules tag in the parent pom.

Slide Note



Quick Start Project

- Create a Maven quick start project which is the parent project.
- Delete the src folder.

Slide Note

Parent Pom

- The project `pom.xml` will be called as the parent pom that will hold common dependencies of all the microservices.
- The packaging of the parent `pom.xml` must be a `pom`.
- The Spring Boot starter parent dependency must be added.

```xml
<modelVersion>4.0.0</modelVersion>
<groupId>com.bej</groupId>
<artifactId>movieapplication</artifactId>
<version>1.0-SNAPSHOT</version>
<name>demo-1-movie-application</name>
<packaging>pom</packaging>
<description>A movie application using microservices</description>
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.7.3</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Slide Note

```
  <modules>
    <module>UserAuthenticationService</module>
    <module>UserMovieService</module>
  </modules>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <configuration>
        <useSystemClassLoader>false</useSystemClassLoader>
      </configuration>
    </plugin>
  </plugins>
  </build>
</project>
```

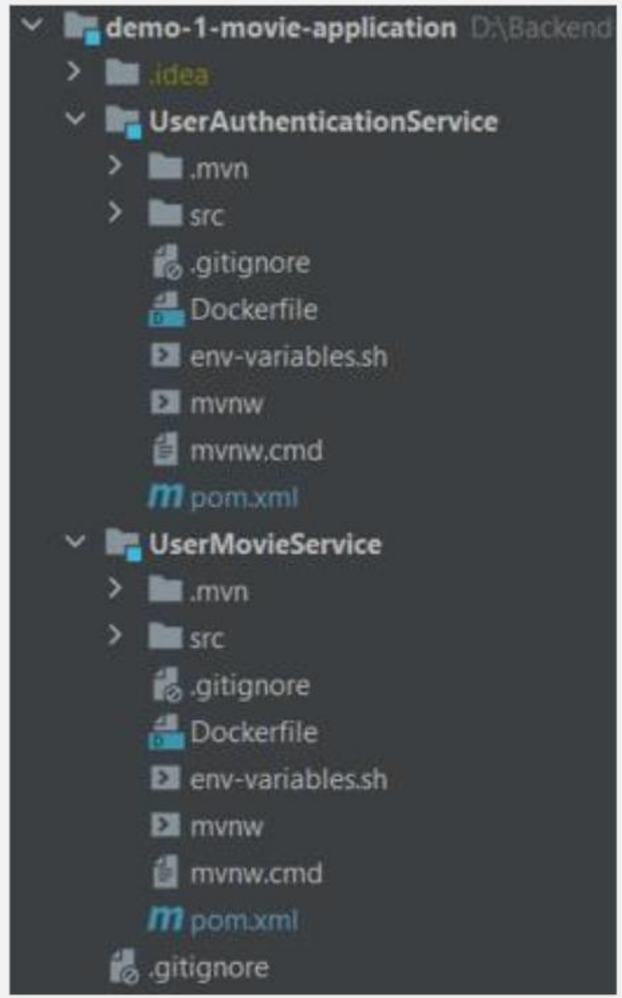
Parent Pom Modules

- In the modules tag, all the names of the modules in the application should be added.
- The surefire plugin must also be added.
- A parent project allows you to define the inheritance relationship between the parent and child POMs.

Slide Note

Modules

- **Create the Spring Boot projects for** `UserAuthenticationService` **and** `UserMovieService` **using the initializer and extract the project to the parent project.**
- **There are two modules created under the parent** `demo-1-movie-application` **project.**



Slide Note

```xml
<modelVersion>4.0.0</modelVersion>
<parent>
    <groupId>com.bej</groupId>
    <artifactId>movieapplication</artifactId>
    <version>1.0-SNAPSHOT</version>
</parent>
<artifactId>UserMovieService</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>UserMovieService</name>
<description>The movie service </description>
<properties>
    <java.version>11</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>
```

Child Pom

- Add the parent pom's `artifactId` and `groupId` in the parent tag.
- This provides an inheritance relationship between the modules and the parent project.
- All the common dependencies like the starter web and starter test can be added in the parent pom and can be shared by the child modules.
- The starter MongoDB dependency is needed for the `UserMovieService` so that is only specified in `pom.xml` file.

Slide Note

Quick Check

A multi-module project is built from a/an _____ that manages a group of submodules.

1. Child pom
2. Aggregator pom
3. Package pom
4. Pom

Slide Note

Quick Check: Solution

A multi-module project is built from a/an _____ that manages a group of submodules.

1. Child pom
2. Aggregator pom
3. Package pom
4. Pom

Slide Note

1. REST call is made to the UserMovieService to register the new user.

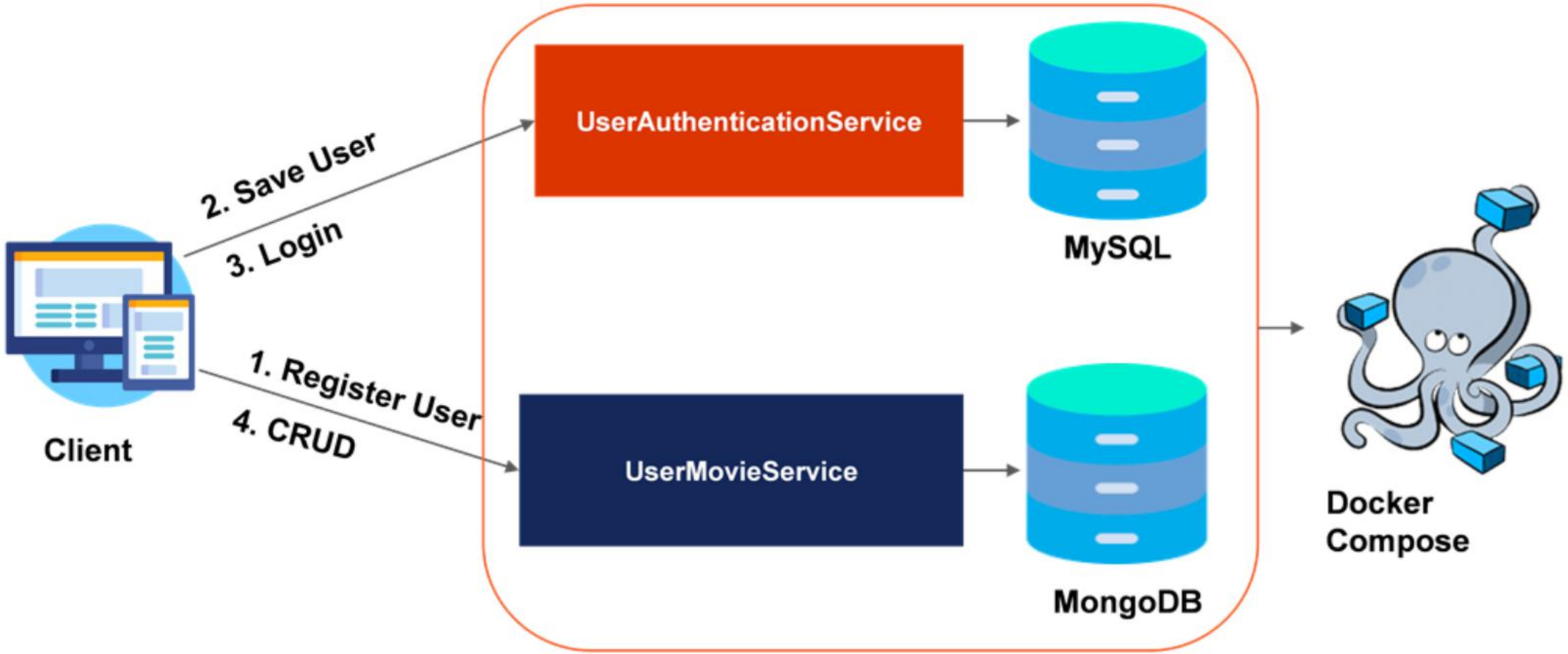
2. Once the user is registered, he/she will be directed login, thus we need to save the user in the UserAuthenticationService.

3. Then the user logs into the system and a JSON web token is generated.

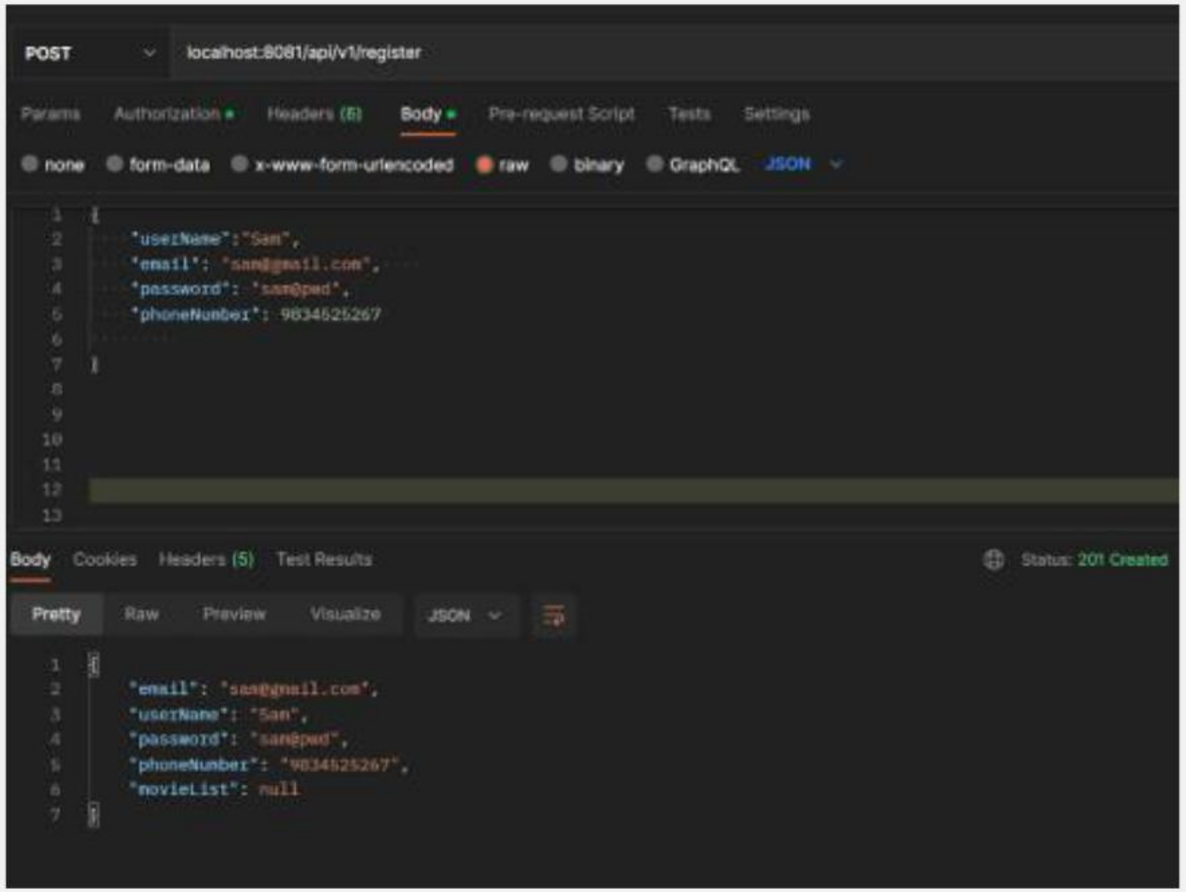
4. Using the JWT, the user can perform all the CRUD operations.

5. Finally, the entire application is dockerized.

How Does the Application Work?



Slide Note



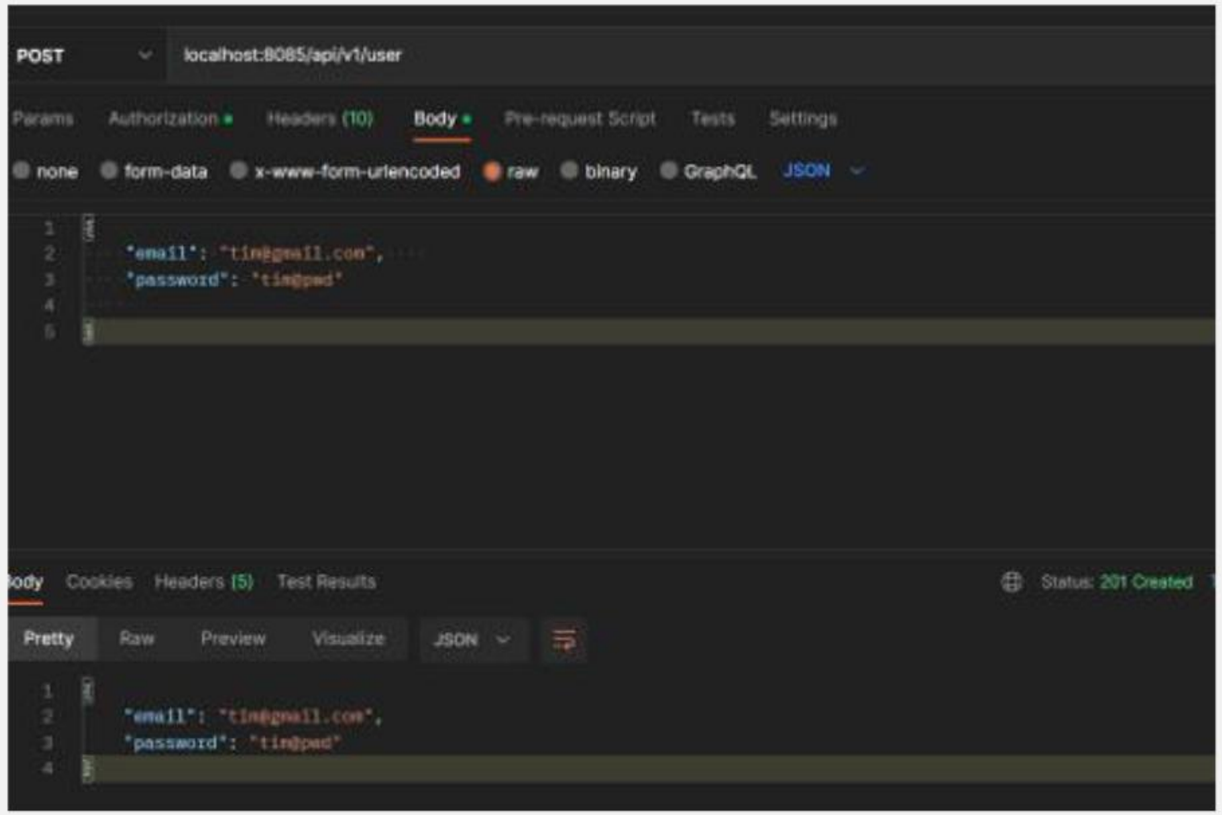
Register the User

- **Register the user on the** `UserMovieService`

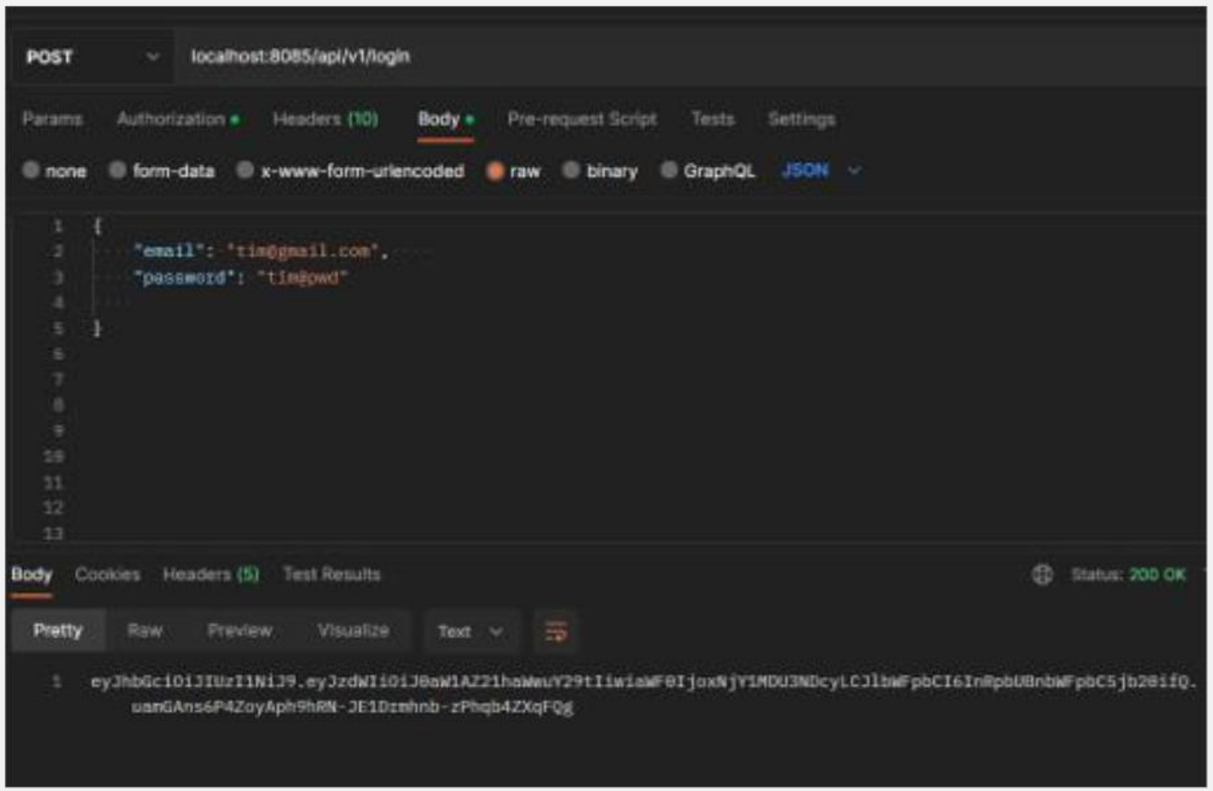
Slide Note

Save the User

- The user credentials must be saved before a user can access the features of the application.
- Hence save the basic credentials like email and password.
- Save the `email` and `password` of the user on `UserAuthenticationService`.



Slide Note



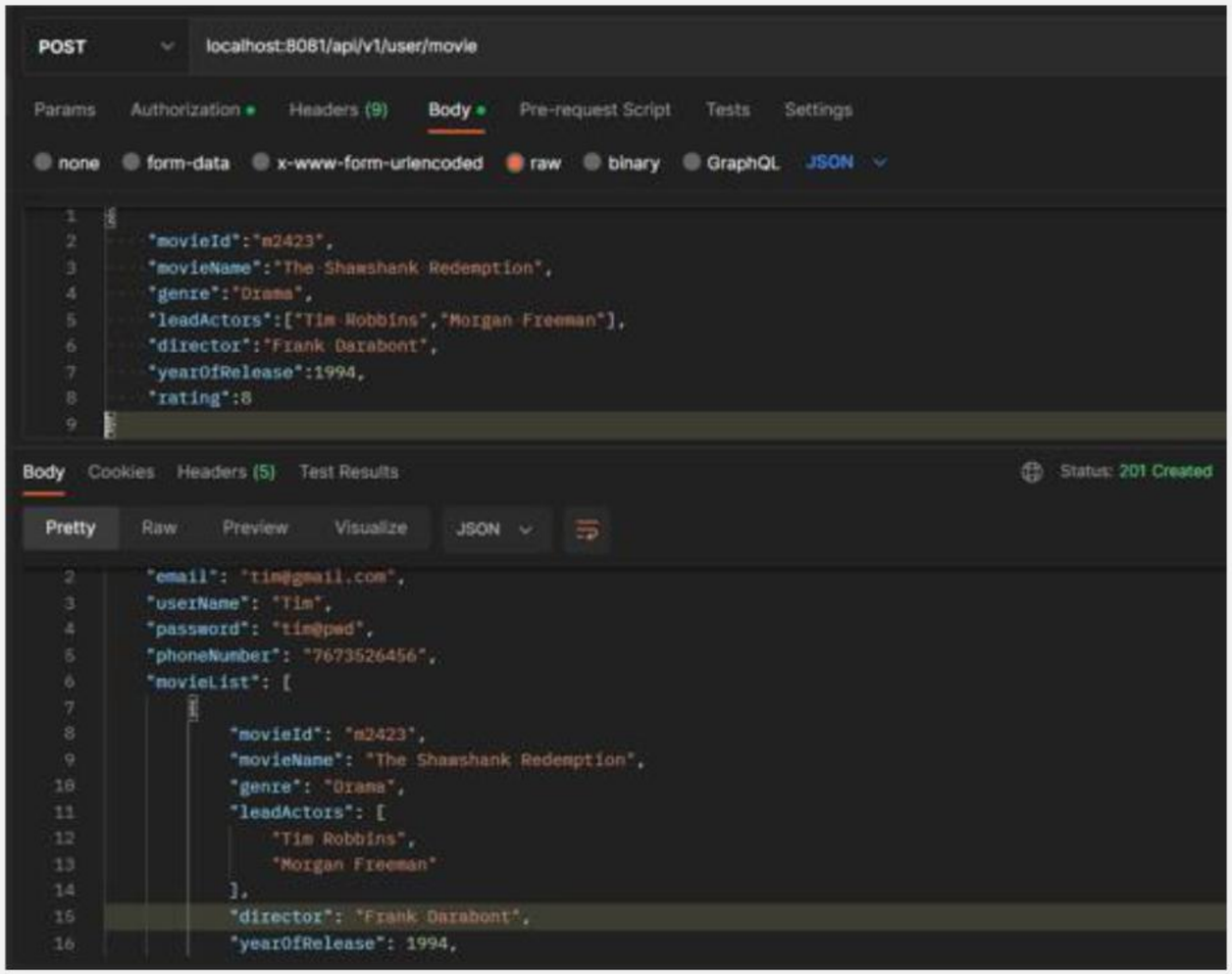
Login

- Login to the `UserMovieService` to add, delete movies.
- At login the jwt Token will be generated

Slide Note

Add a Movie to a User

- The movie details are sent in the Request Body.

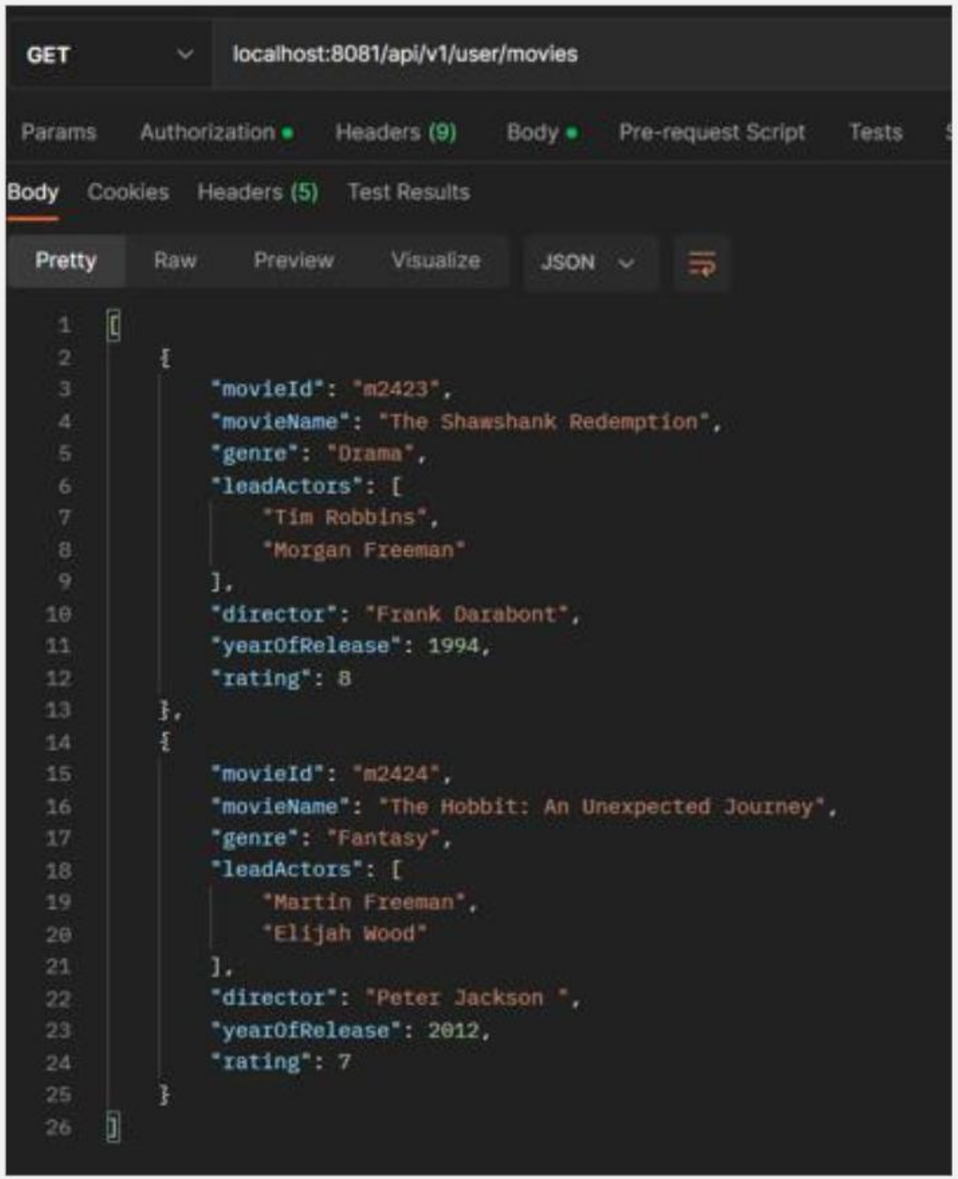


Menu

00:39:00:00:43

39/ 43

Slide Note



Get the Movie List of a User

- Fetch all the movies from the user movie list

Menu

00:40:00:00:43

40/ 43

Slide Note

Streaming Application

Consider a streaming application that enables users to watch movies on any smart device. It provides multiple features to all its registered users. Let's create multiple microservices for the streaming application.

1. The user must first register with the application.
2. Login credentials such as Id, password must be entered.
3. All features provided by the streaming application, like adding favourites, compiling a watch later list, etc., can then be accessed.

DEMO

Slide Note

Streaming Application

Create a parent project called MovieApplication.
This will contain the UserAuthenticationService and UserMovieService as microservices.
Create the UserAuthenticationService
Create the UserMovieService

Check the solution here.

DEMO