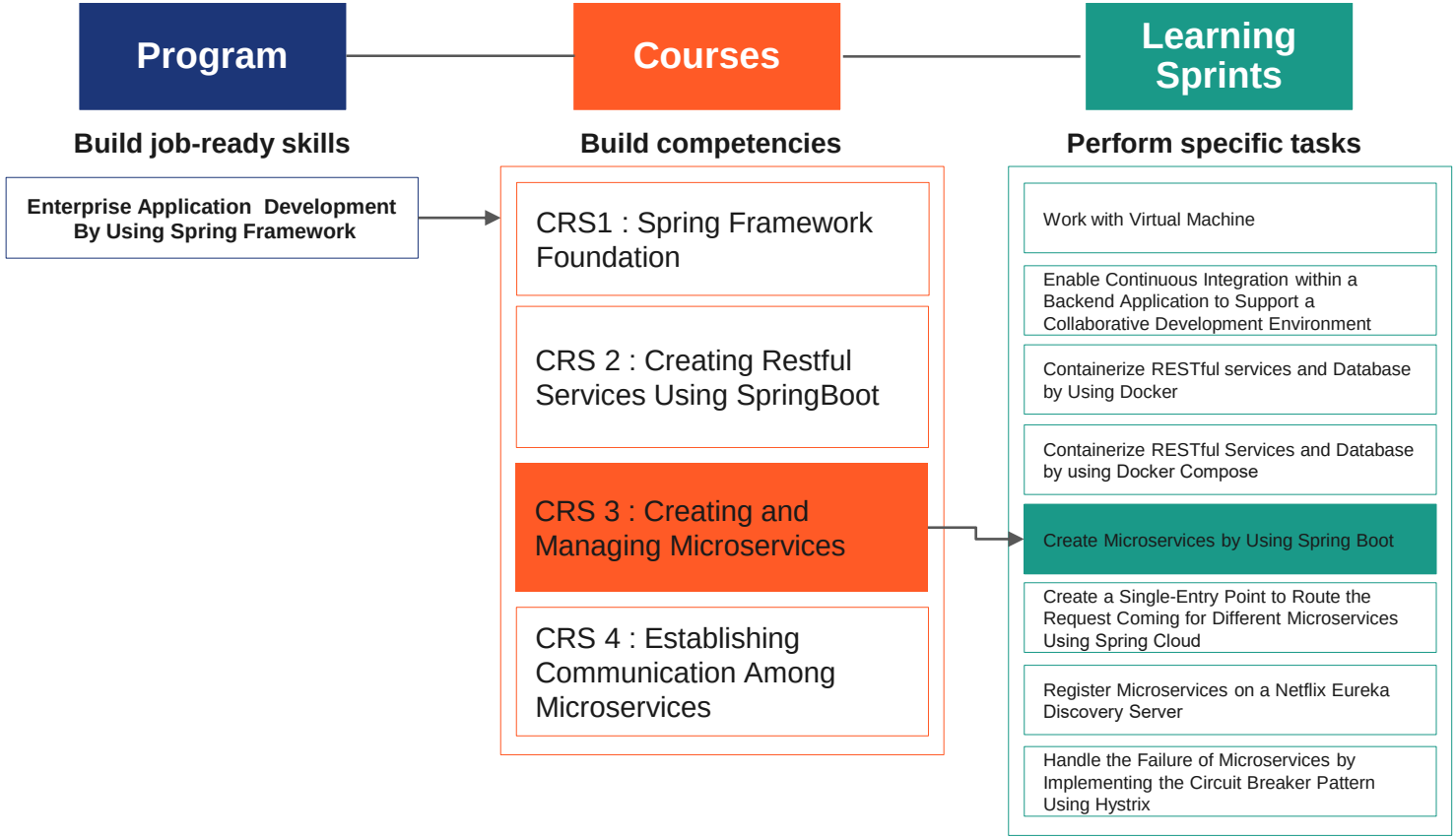
Backend Program: Course 3: Structure



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 and produces a report and writes a method `calculateGrade()`.

Think and Tell

- Do you think the code is written in a modular manner?
- If developer Jim 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 the testing be redone for all the modules of the application or only the part that Sam changed?

Think and Tell

- The application is deployed after creation. There is a new requirement where the total marks for which the student writes the exam is reduced from 100 to 80.
- Sam must now calculate the total marks in his method for 80 and 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?

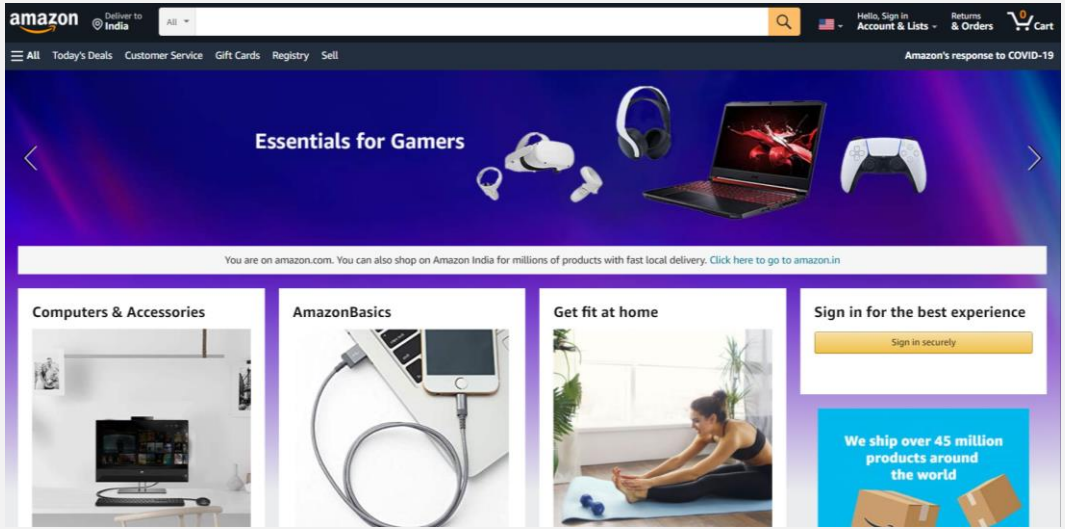
Think and Tell

- 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 subject be stored ?
- If the database for the application changes from MySQL to MongoDB. Will the code be changed by all developers while performing database operations?

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

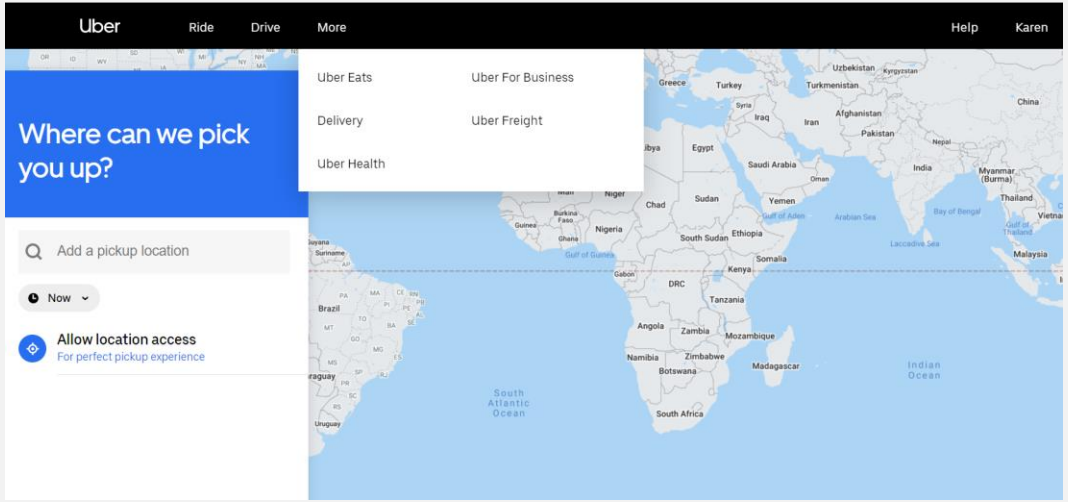
E-commerce Application

- 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

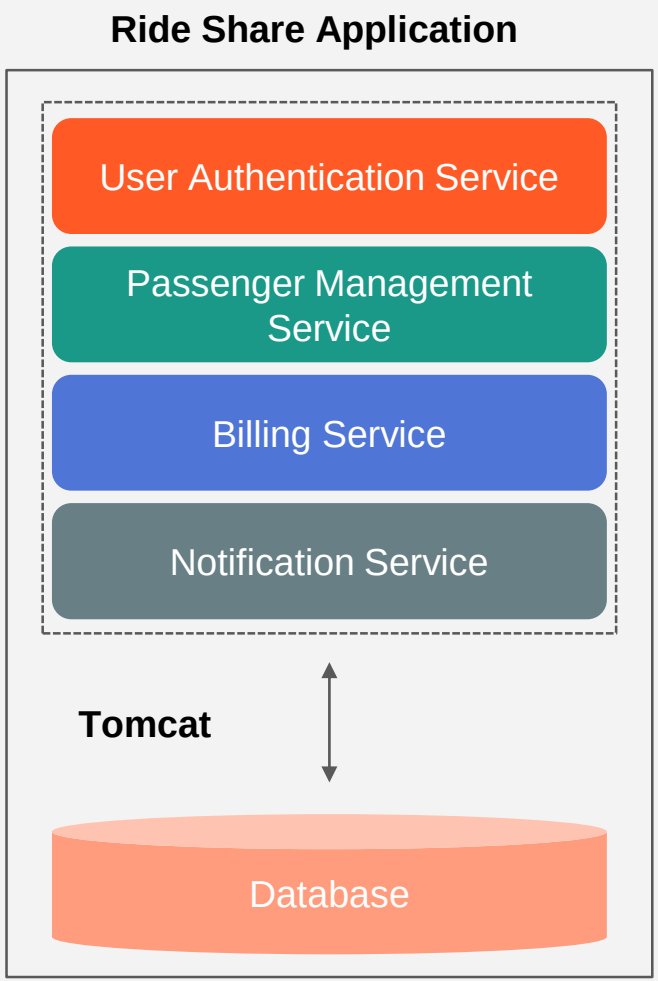
- Let us imagine that you are building a ride sharing application that helps commuters book cabs at anytime of the day or night by accessing their live location.
- If we add a new feature to the application, what do you think will happen to the live location sharing feature?
- Will the feature go down at the time of redeployment?
- Will this impact the business?



Create Microservices by Using Spring Boot

Learning Objectives

- Explain monolithic applications
- Define microservices
- Explore microservices architecture
- Develop microservices using Spring Boot

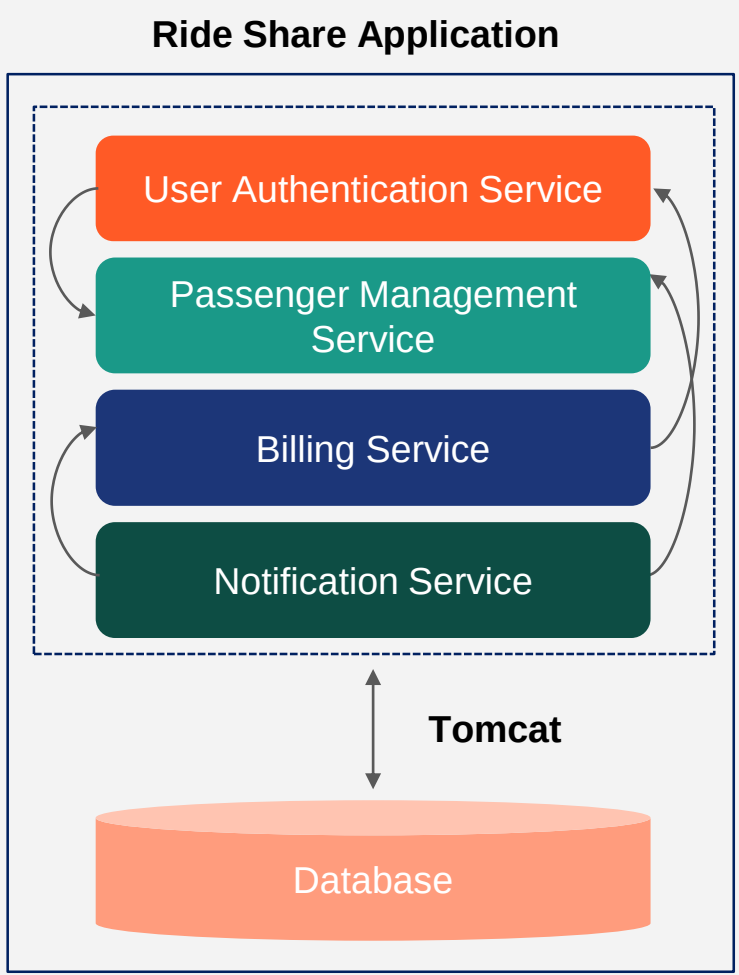


Monolithic Applications

- 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.
- It is a self-contained, and independent from other computing applications.
- It is 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.

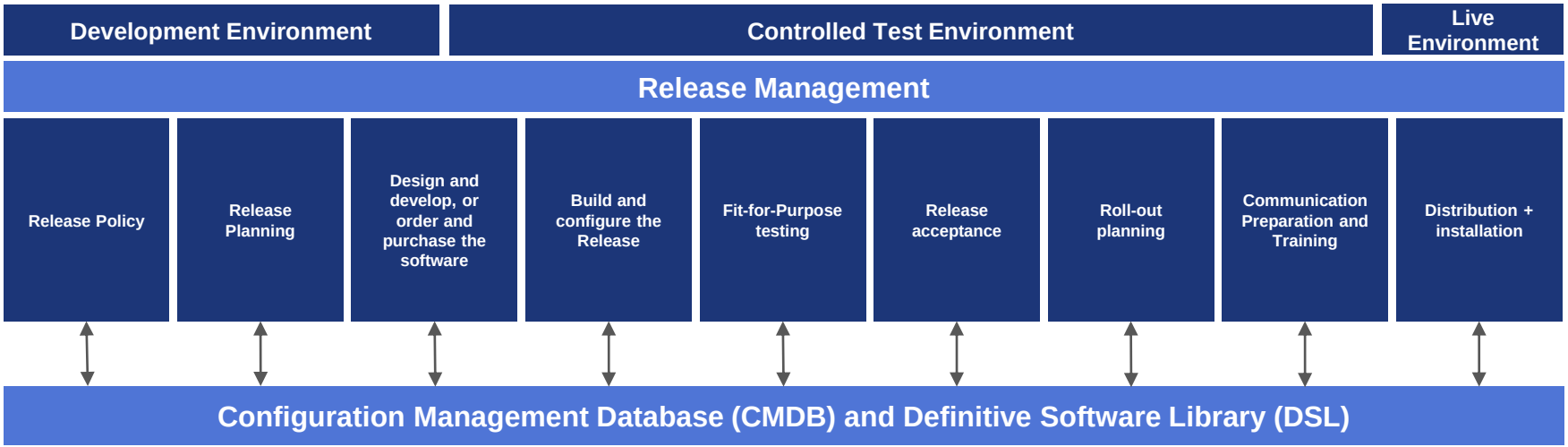
Problems With Monolithic Applications

- Ride share application currently has four services.
- Imagine the complexity involved if now 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 between them.
- A severe complexity is related to the change impact, small changes in an existing component can have a ripple affect, since the components are interconnected and dependent on one another.



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 component in the application the entire application should be redeployed.



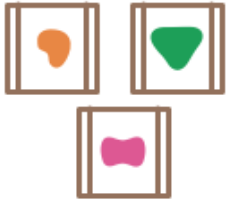
Disadvantages of a Monolithic Approach

- 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

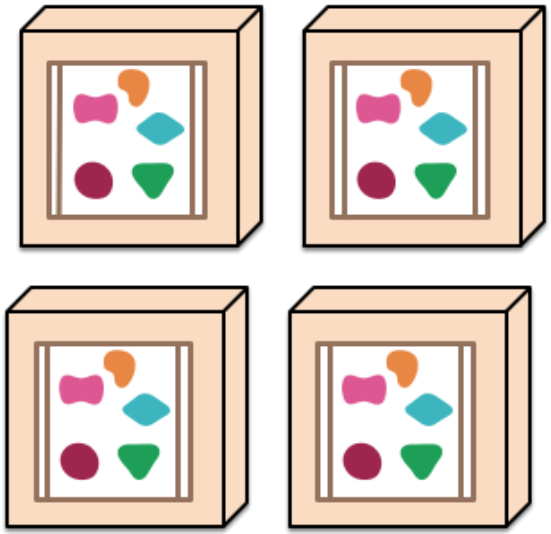
Shifting From Monolithic to Microservices

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

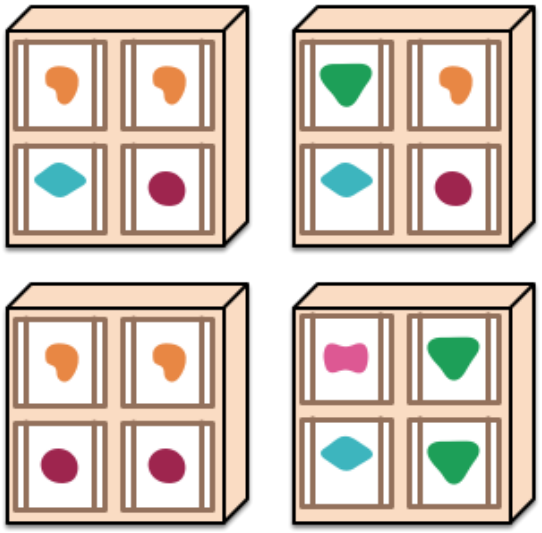
A microservices architecture puts each element of functionality into a separate service...



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



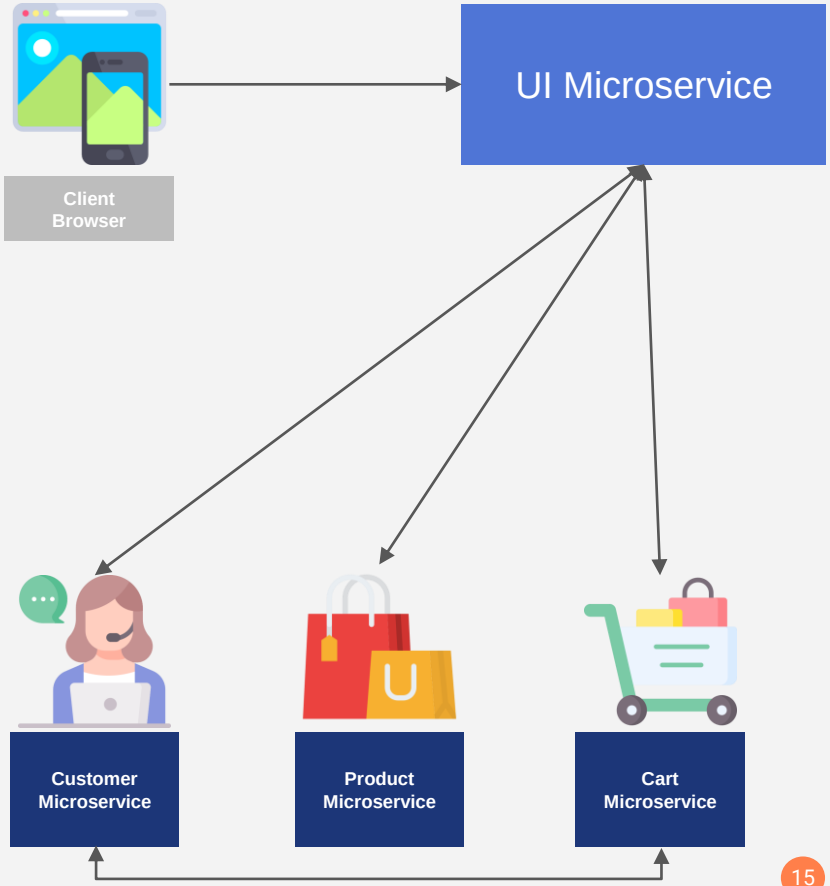
... and scales by distributing these services across servers, replicating as needed.



Microservices

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

- They are an architectural style, in which large complex software applications are composed of one or more services.
- They 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(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.

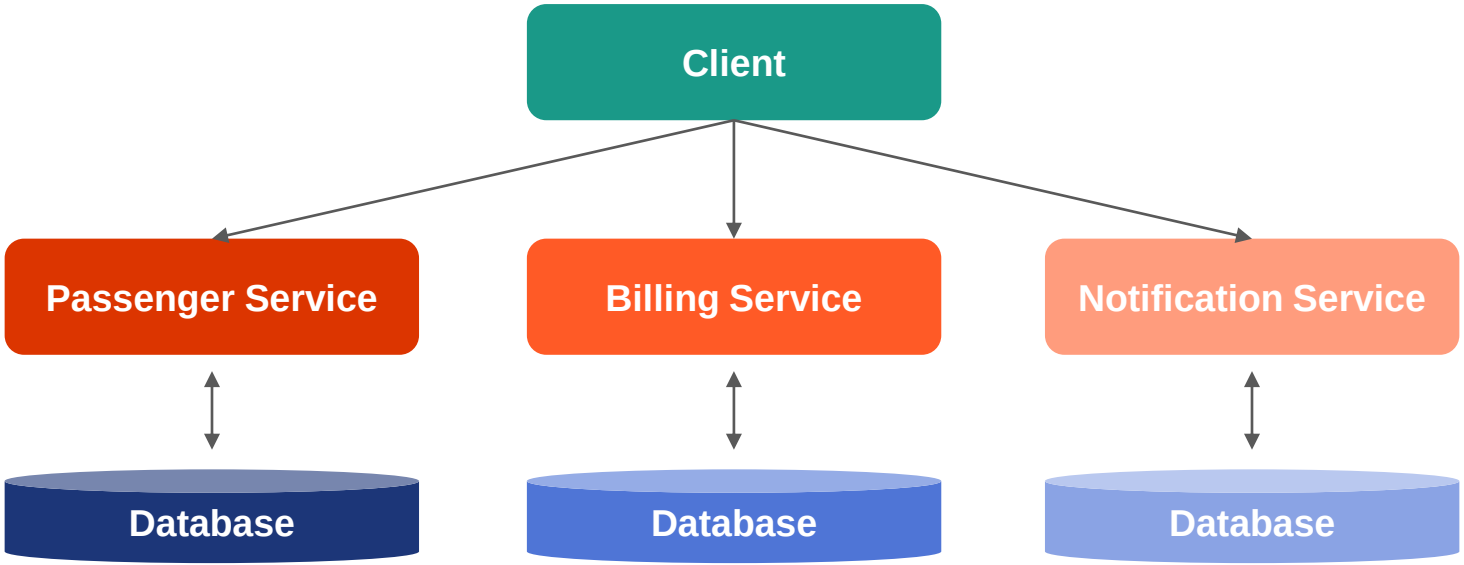
Challenges with Microservices

- **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:



Quick Check

Microservices are _____ by default.

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

Quick Check: Solution

Microservices are _____ by default.

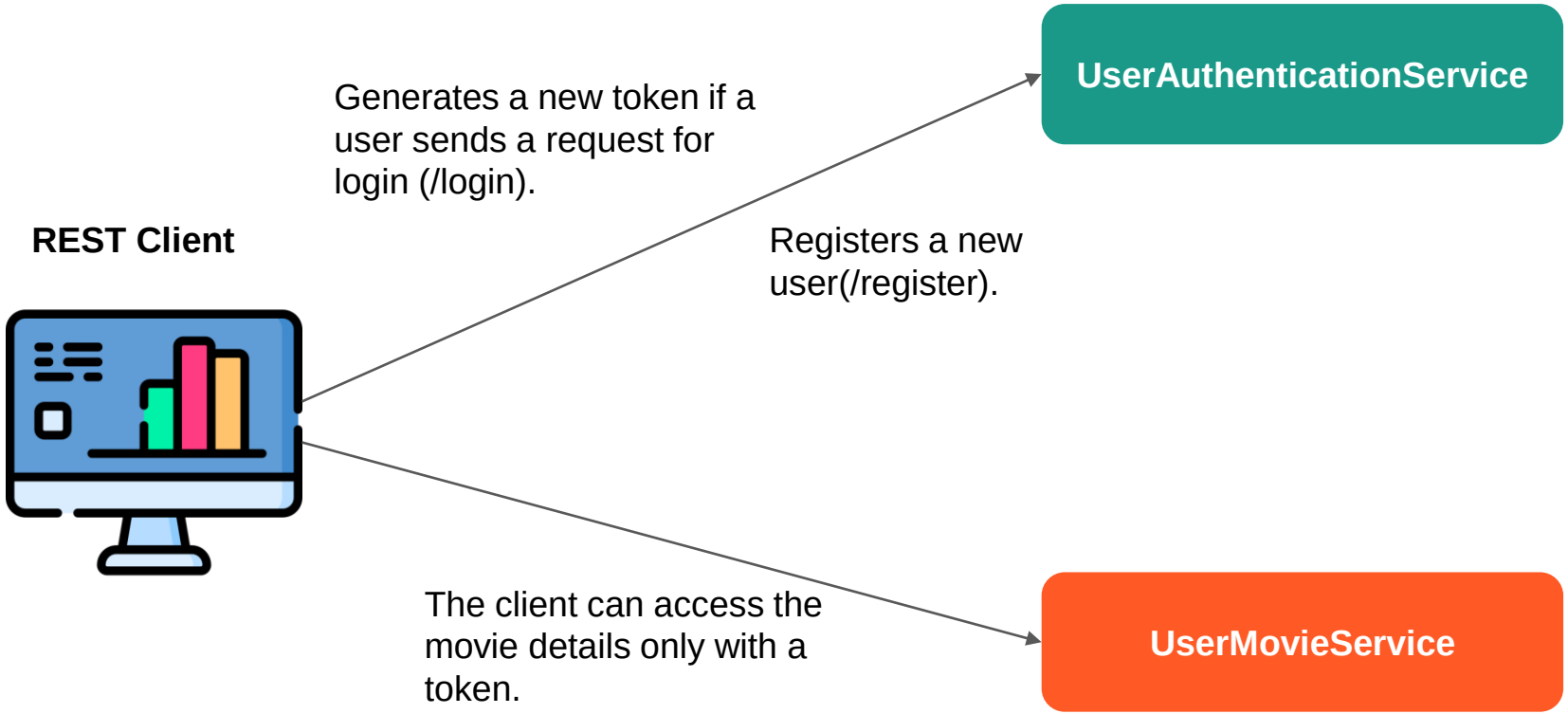
1. stateful
2. singleton
3. **stateless**
4. secure

Developing Microservices

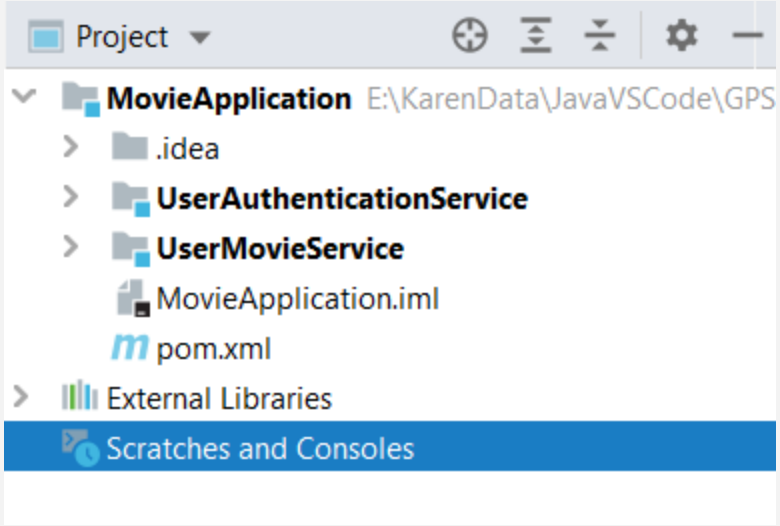
Steps to Create Microservices

- 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



Multi-Module Project - Maven



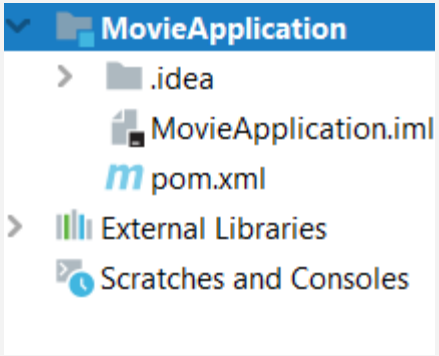
- The microservices together form a single application, thus it is logical to build a multi-module project using Maven
- It is wise to use Maven's multi-module feature to logically separate services from each other because the project contains multiple services.
- This creates an explicit boundary between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

Creating a Multi-Module Application

- A multi-module application is like a directory which holds the different microservices as modules.
- In order to create a multi-module project follow the below steps.
  - 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.

- Note : The project pom.xml will be called as the parent pom that will hold common dependencies of all the microservices.

Quick Start Project

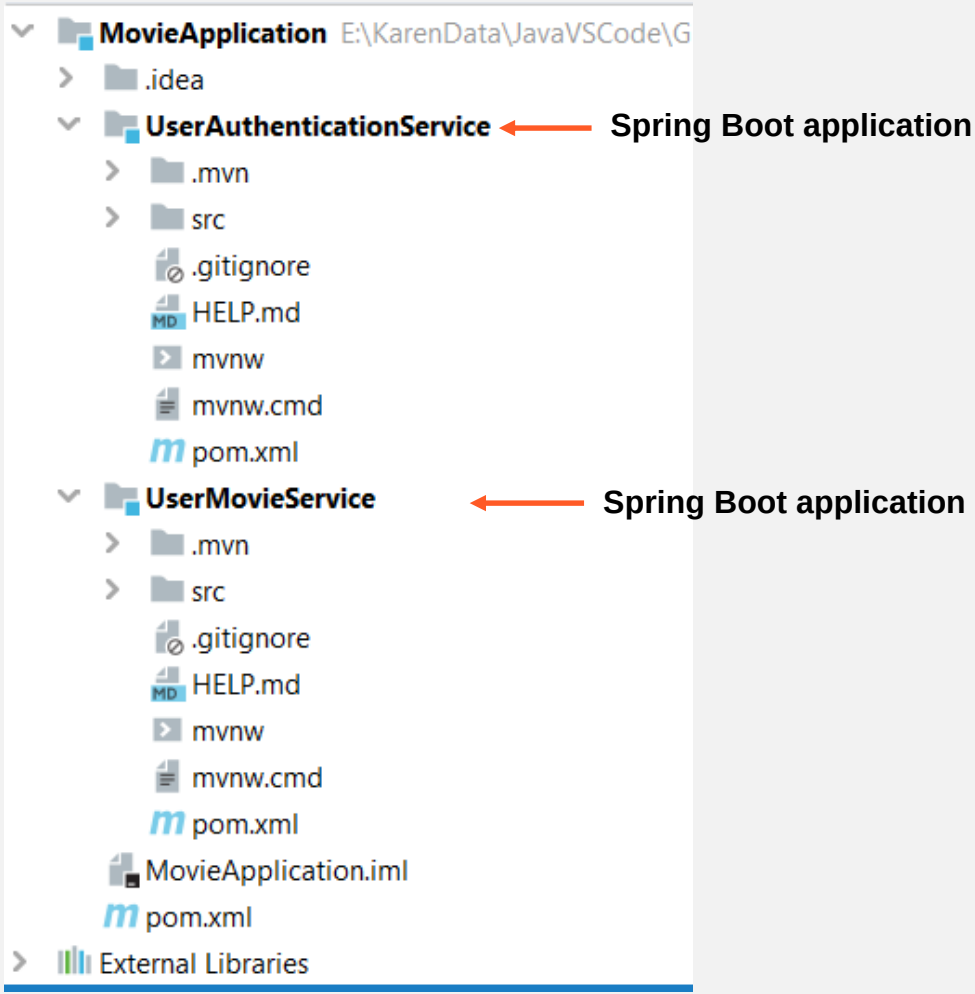


- Create a Maven quick start project which is the parent project.
- Delete the **src** folder.

```
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.niit</groupId>
  <artifactId>MovieApplication</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>MovieApplication</name>
  <packaging>pom</packaging>
  <description>A movie application using microservices</description>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.5.0</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>
  <dependencies>
  </dependencies>
  <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

- The packaging must be a pom.
- Add the Spring Boot starter parent dependency.
- 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.



Modules

- Create the Spring Boot project using the initializer and extract the project to the parent project.
- There are two modules created under the parent MovieApplication project.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>com.niit</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, 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.
- Note that only the starter Mongodb dependency is needed for the UserMovieService is only specified.

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

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

Streaming Application

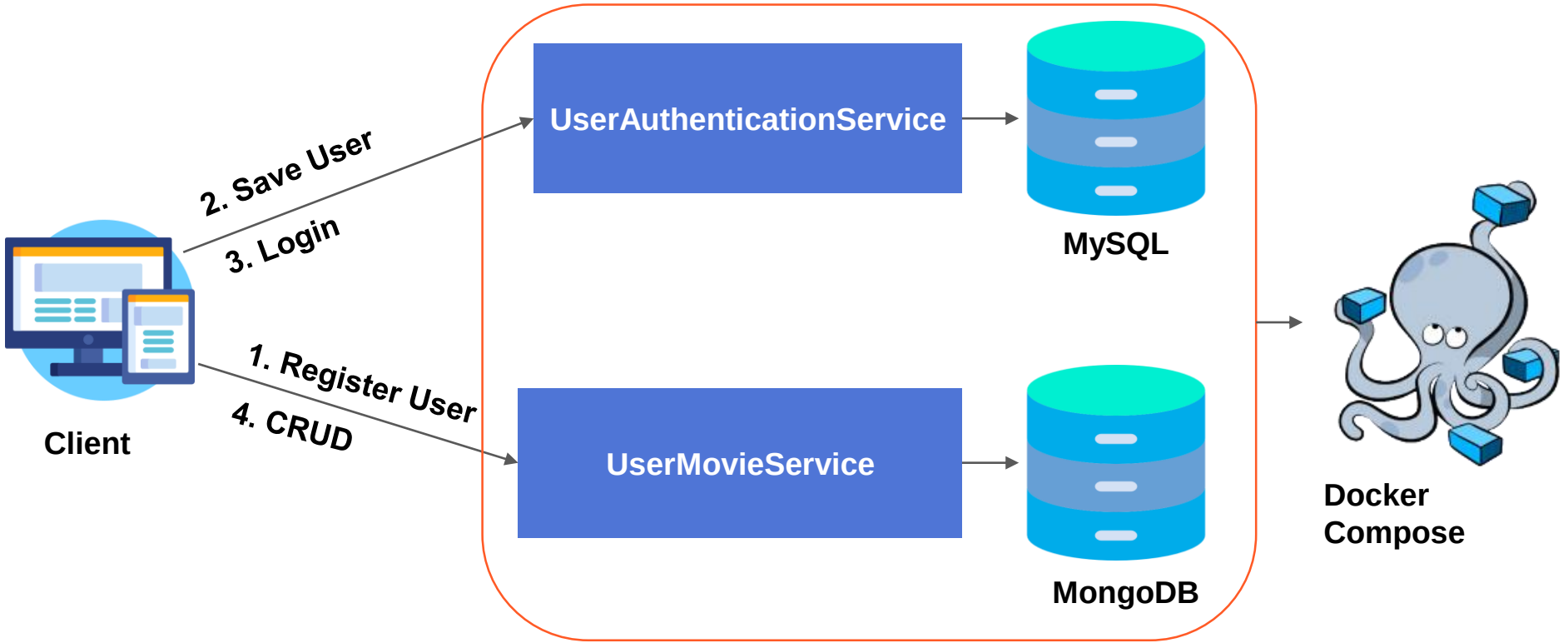
Consider a streaming application that enables users to watch movies on any smart device. The application provides multiple features to all its registered users. A user needs to register with the application in order to access some of its features. Let us create multiple microservices for the streaming 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.

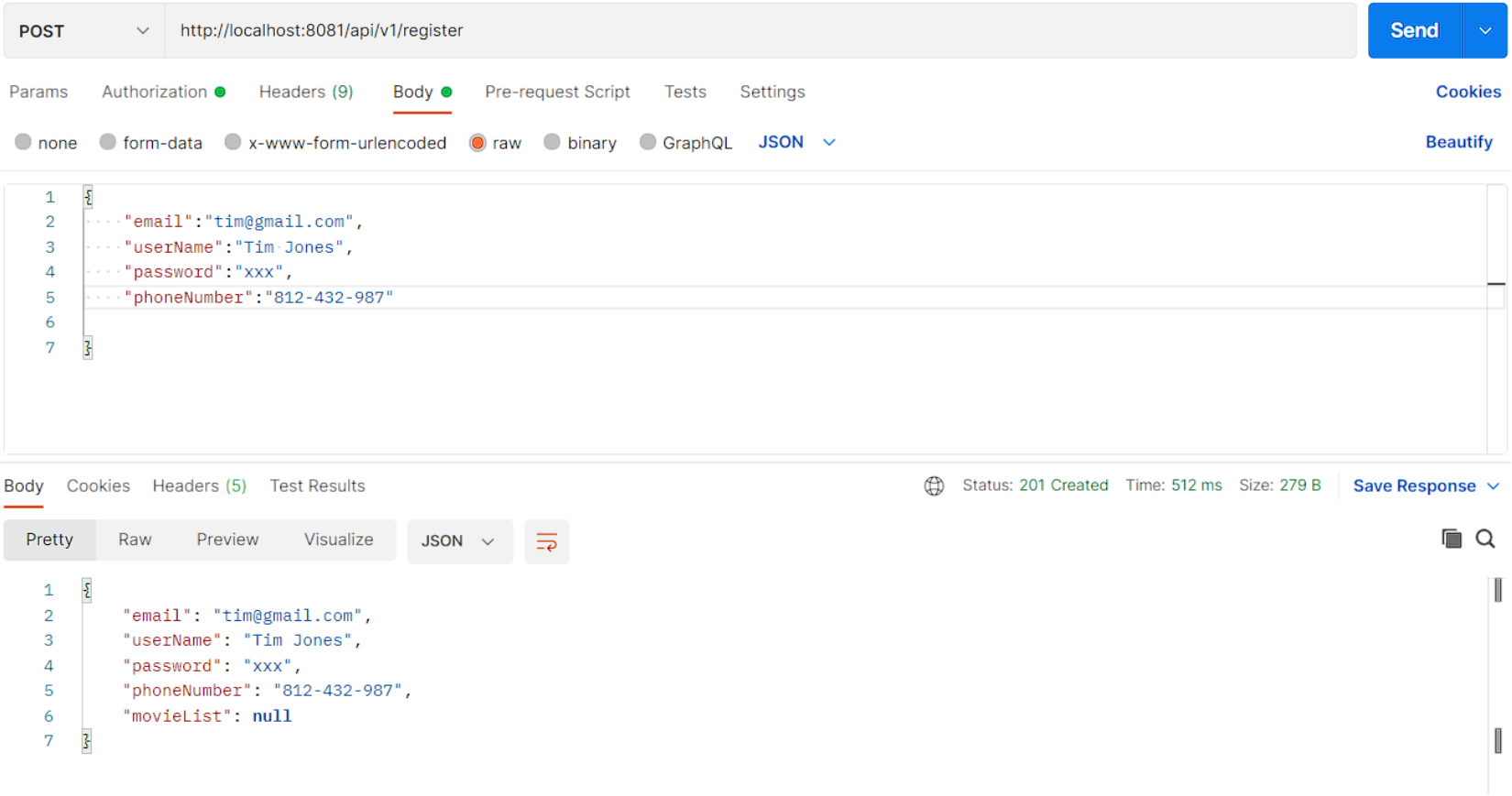
Let us create a parent project called **MovieApplication**.
This will contain the **UserAuthenticationService** and **UserMovieService** as microservices.
Using a docker compose file dockerize the application.

DEMO

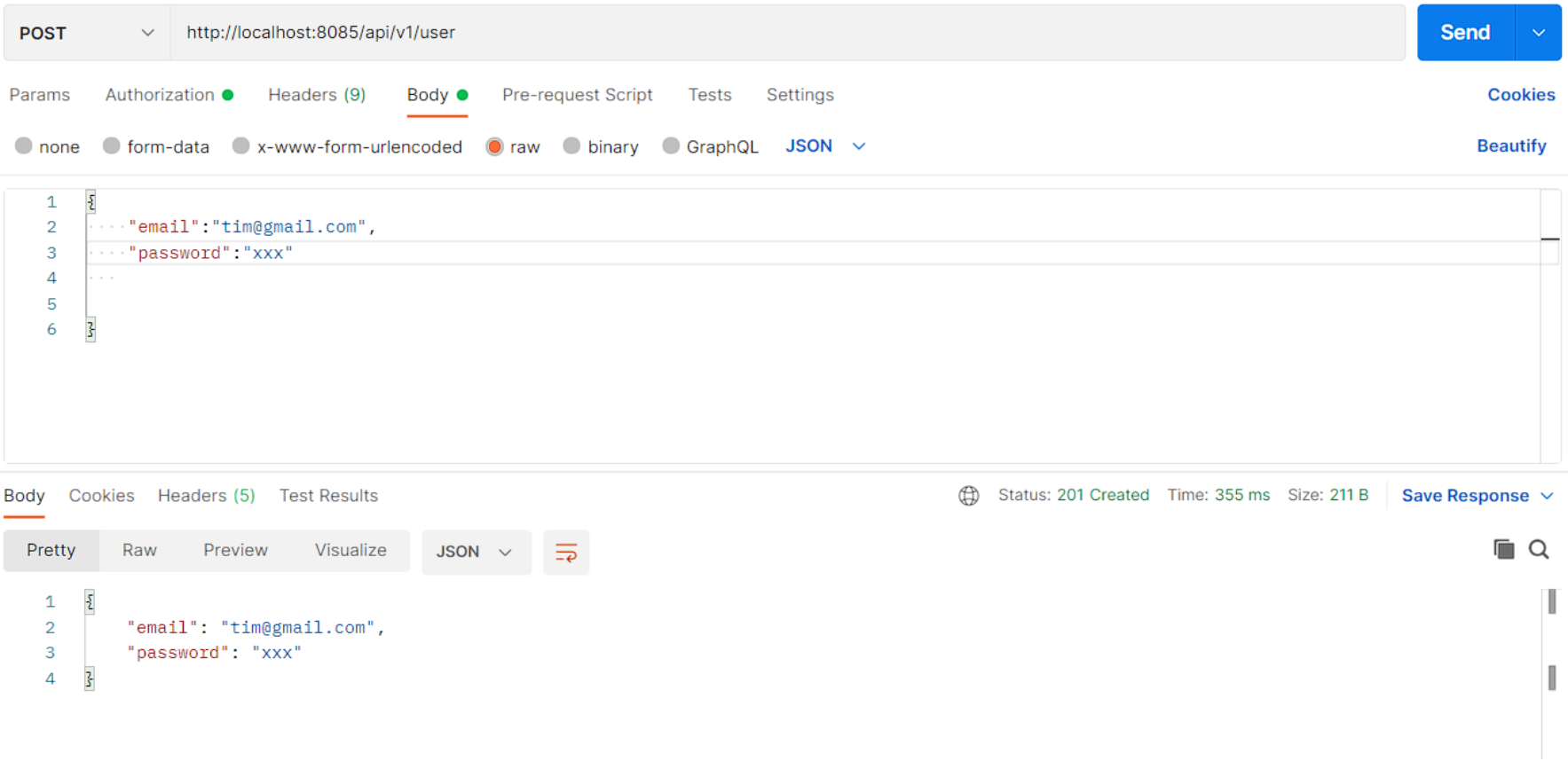
How Does the Application Work?



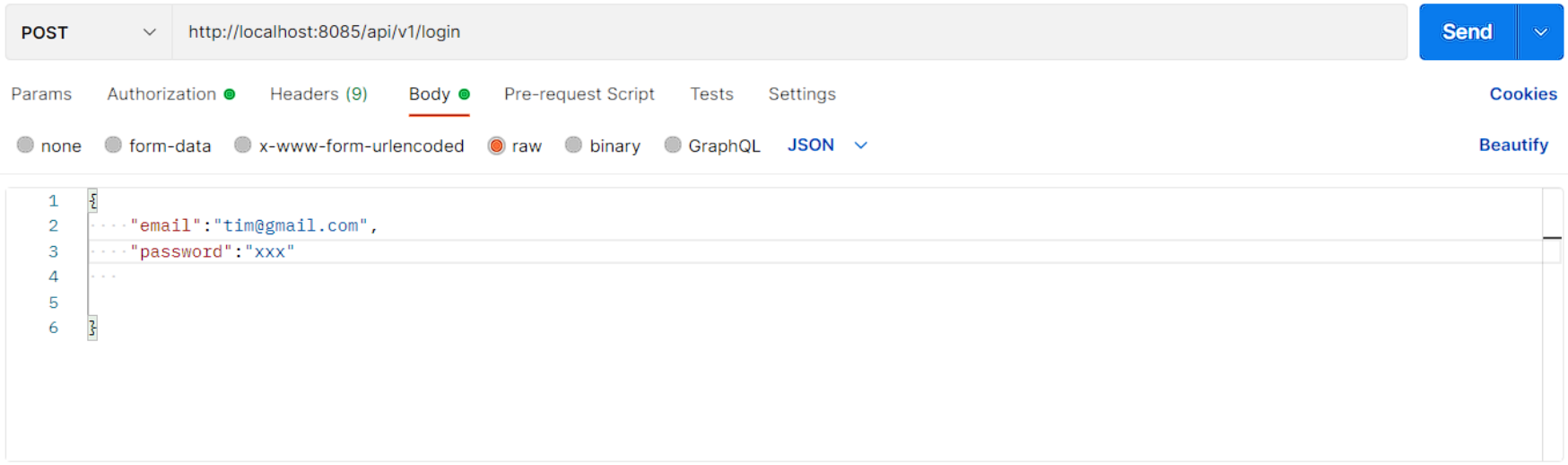
Register the User



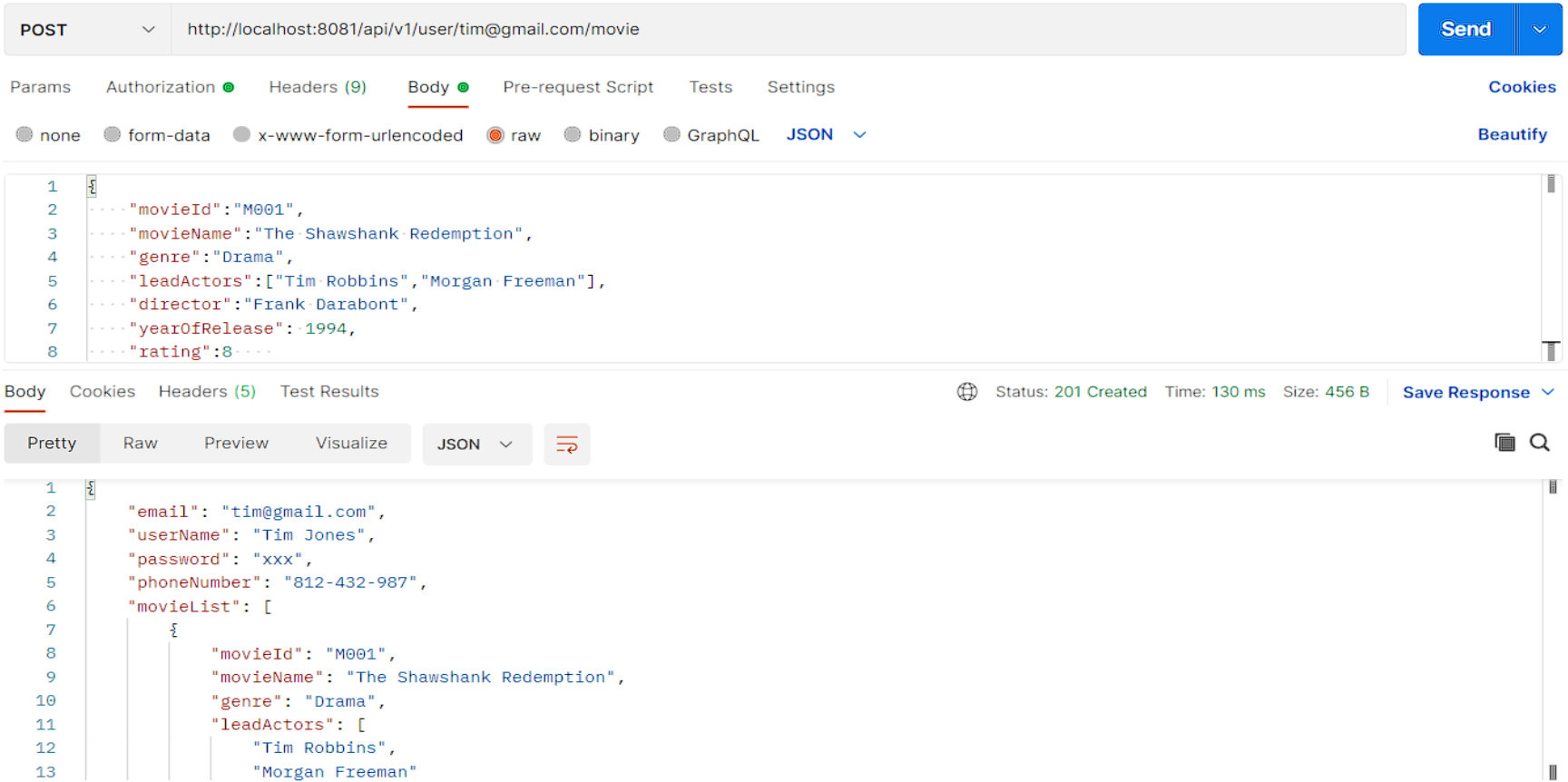
Save User



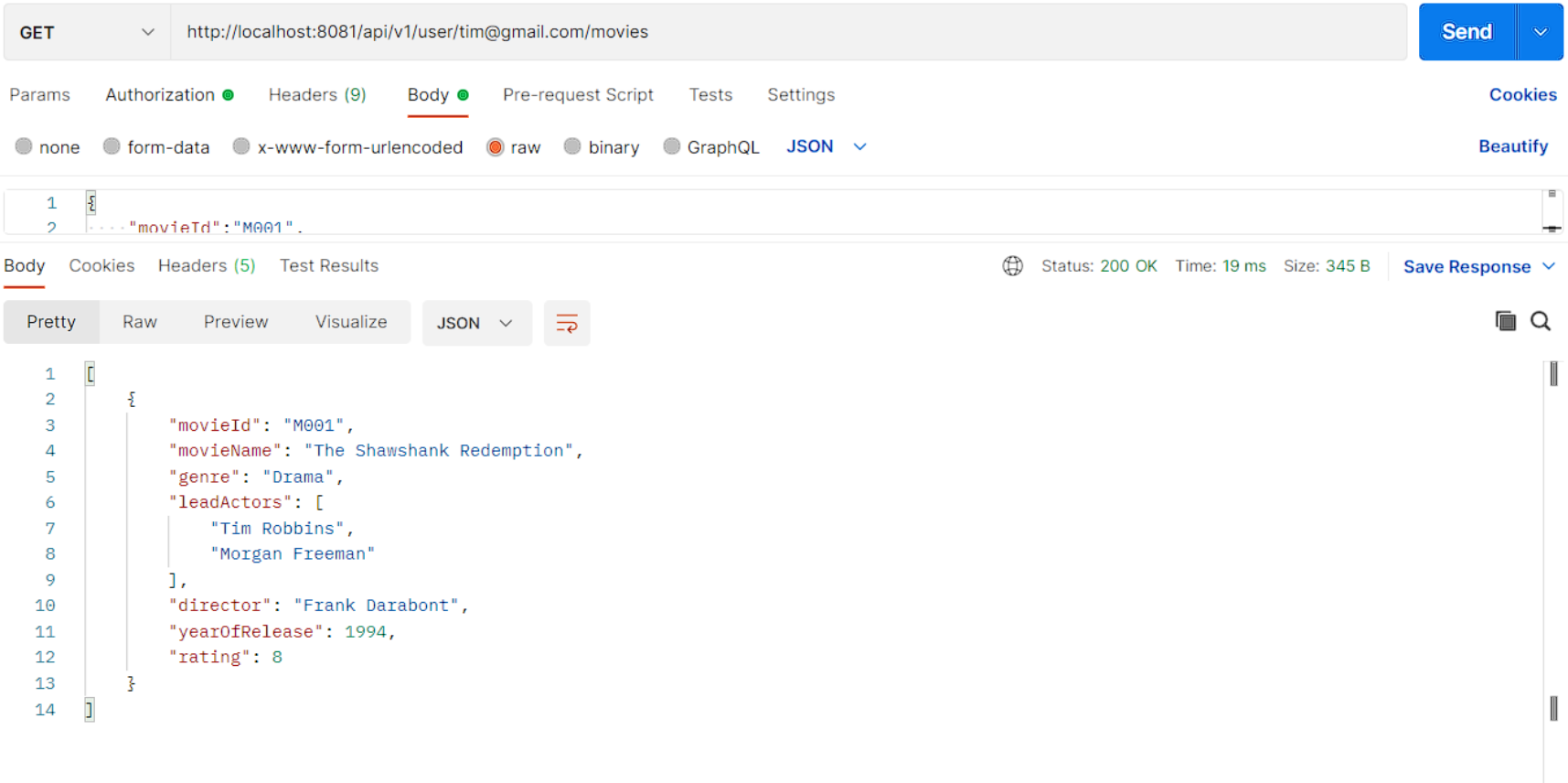
Login



Add a Favourite Movie to a User



Display All Favourite Movies for a User



Key Takeaways

- Monolithic Applications
- Advantages of using Microservices
- Challenges while working with Microservices
- Steps to create Microservices