Section 1: Introduction

* Introduction
* Rest Web Service
  + A web service is software that makes itself available over the internet.
  + A web service can be accessed over http ftp and many other protocols.
  + Data is typically exchanged in XML or JSON format over the HTTP protocol.
  + REST REpresentative State Transfer
* Understanding Microservices

**Monolithic Application**

Course Catalog

Database

Course

User

**Microservice Architecture**

Course Catalog App Service

Course App Service

Rest

**Microservice 2**

**Microservice 1**

Rest

User App Service

**Microservice 3**

* The key behind microservice is that you build individual modules and then expose them as services.
* Typically, you expose microservices through REST API so that they can be accessed over HTTP by remote clients.
* Microservices are loosely coupled. You can build test and deploy them independently.
* Microservices Advantages
  + You can do faster changes.
  + Better handling of failure and greater availability of application.
  + You can scale easily.
* Java Microservices Learning Path

Project Work

Microservices with Spring Cloud

Spring Boot & Spring Framework

Java



* With Spring Cloud, you can implement Microservices design patterns easily.
* Spring Cloud is optimized for Spring Boot, but you can use it with other frameworks.
* Netflix technologies are available in Spring Cloud.

Section 2: Java Basics

* POJO
  + Class must be public.
  + Properties(fields) must be private.
  + Default Constructor is mandatory.
  + Can have optional Constructor with arguments.
* POJO’s, you make your programs more readable and increase reusability.
* A Bean is a POJO and additionally it must be Serializable.
* What is Serialization?
  + A Serialized object is converted to Byte Stream and stored in memory.
  + A Byte Stream can be deserialized to construct the Object.
  + Serialization helps in sharing object with multiple systems and Caching for faster data retrieval.

Section 3: Setting the stage for Spring Framework.

* Annotations
  + Java Annotations
    - Annotations are used to provide additional information to classes, methods, variables, constructors.
    - With Annotations you tell the compiler that the program will behave in a certain way.
    - Prior to Annotations Spring Framework configuration was mainly done through XML file.
    - Annotations will generate certain code or configuration, and the complexity of the code is hidden from the developers.

Section 4: Spring Framework and Spring Boot

* Spring Framework and Spring Boot Introduction
  + Spring Framework
    - <https://spring.io/>
    - Using the Spring Framework, you can build enterprise Java Applications.
    - Spring provides a template for common services like Transaction Management.
    - With the Spring Framework you focus on writing business logic. The Framework takes care of common concerns.
    - Spring provides infrastructure support like DB connectivity.
  + Spring Boot
    - Using Spring Boot, you can bootstrap or quickly start a Spring Application.
    - Spring Boot is an Opinionated Framework.
    - As of January 2021, Spring Boot is now the most popular way of creating Spring Applications.
    - You can run a Spring Boot application with minimal configuration.
    - Quick development with a stater project.
    - Spring Boot takes an opinionated view on how a Spring application should be built.
    - You can build an application with front end pages, controllers, and backend databases quickly.
    - Spring Boot has an embedded Tomcat Server inside the application and produces the self-contained jar file that can be executed in any Java environment.
    - Spring Projects
      * <https://spring.io/projects>
    - Spring Initializer
      * <https://start.spring.io/>
* A Hello World REST service using Spring Boot.
  + Create a new Spring Boot Project called SpringHelloWorld.
  + Talk about dependencies in the POM file.
  + Create a controller package.
  + Create the HelloService class and then add the Annotations for Rest Controller, Request Mapping, and Get Mapping.
* Dependency Injection Using Spring Framework.
  + Create a new Java Project in either Intellij or with the Spring Initializer with no dependencies.
  + Then locate the main entry point and sout Dependency Injection Demo.
  + Add the Animal interface.
  + Add Dog, Cat, and AnimalSpeak classes.
  + Add Component annotation to each class so that Spring will wire up the dependencies.
  + Now demo how dependency injection works. Look over your code before demoing.
* Maven for build and dependency injection
  + Maven is a tool for build and dependency management.
  + With Maven we are able to declare all our dependencies in a pom.xml file.
  + Put all your dependencies in your pom.xml file.
  + All required JAR files will get downloaded form the Maven repository.
* Create the spring boot Course application with multiple dependencies.
  + Dependencies
    - Spring Web
    - Spring Data JPA
    - Rest Repositories
    - Note: Until we add the SQLite database you will have to remove Spring Data JPA.
* Returning an object with REST API
  + Add demo\_hard\_coded package.
  + Add ExampleCourse Controller
    - Annotate the class with @RequestController
    - Add a getCourse method and annotate it with @RequestMapping
    - Have it return “My First Course…”
  + Add a class called ExampleCourse with
    - Fields courseId, courseName, and authorName
    - constructor
    - getters and setters
  + Now add the getCourses method annotated with the @Request mapping. It will return a list of courses that is hardcoded.
* JPA, ORM, Hibernate and Spring Data JPA
  + Java Persistence API (JPA)
  + Object Relational Mapping (ORM)
    - You map Java classes with Database tables.
    - JPA is the way to use ORM.
    - It provides APIs to manage relational data in Java applications.
  + Hibernate
    - Is an ORM Framework with implements ORM specification.
    - Its main job is to map java objects to database tables.
    - JPA is the specification and Hibernate is one of the ways to implement JPA.
  + Spring Data JPA
    - Spring Data JPA makes it easy to work with JPA and simplifies database interaction.
* Installing an SQLite Database
  + <https://www.sqlite.org/download.html>
  + Text

    Description automatically generated
  + Cmd into the sqlite folder
  + Then type sqlite3 to get into your in-memory database.
  + Add the sqlite3 folder to your path variables.
  + Use the sqlite3 .text file to insert some data.
  + You can use .quit to exit.
* Fetching data from database using Spring Data JPA
  + Database Creation
    - First, we need to make sure the sqlite3.db file is in the same folder as a java code.
    - Locate the Spring Boot CourseApp application and then enter into it.
    - Then click on the path at the top and erase the path and then type in cmd and hit enter.
    - This will open the cmd in the CourseApp application so you can add your database.
    - Create the database and use the sqlite3.txt file to help.
  + Fetch the two records we added and display on the web page.
    - Add dependencies in your pom.xml for sqlite3.
      * Spring-boot-starter-data-jpa
      * Sqlite-jdbc
      * Sqlite-dialect
      * Link: <https://mvnrepository.com/>
    - Create a controller package.
      * Add a CourseController.
    - Create an entity package.
      * Add a Course POJO Entity.
    - Create a repository package
      * Add the Course Repository interface
    - Add sql connections to application.properties
    - Now, run it and then fetch all the data from the database and show in on the webpage.
* CRUD operations using Spring Data JPA
  + First add the @JsonIgnoreProperties({“hibernateLazyInitializer”, “handler”})
  + Add the getSpecificCourse controller method.
  + Add the saveCourse controller method.
  + Add the deleteCourse controller method.
  + Test everything with postman.
* Exporting the application to a JAR file.
  + Stop the project from running.
  + Select the Maven Tab at the top right.
  + Click on M.
  + The type mvn you are looking for the mvn package.
  + Then double click it and it will build the Java File.
  + The jar file will be in the target folder.
  + java -jar target\CourseApp-0.0.1-SNAPSHOT.jar
* Integrating services with RestTemplate

Course Catalog Service

Course Application Service

* + Create a new Spring Boot project called CourseCatalog
  + Make sure you are using Java 1.8 with the pom version 2.2.4release
  + Dependencies
    - Spring Web
    - Rest Repositories
  + Change the port to 8002
  + Add the Controller package
    - Add the CatalogController class
    - Add the getCatalogHome method with default mapping.
    - Run it and make sure its running on server port 8001
    - Now go back to the CourseApp and locate the CourseController
      * Add a new method called getCourseAppHome and have it return the string “Course App Home!”.
    - Now Integrate the CourseCatalog with the CourseApp
    - Within the CourseCatalog CatalogContorller add the RestTemplate data so that it fetches the String “Course App Home” from the CourseApp and append it to the string “Welcome to Pyramid’s Course Catalog!”.
    - We just integrated two rest services using RestTemplate. 😊
  + Add the Course entity to the CourseCatalog
  + Add getCatalog mapping.
  + Add the getFirstCourse method. And look at the browser to see the data.
  + This is cool we are getting data from spring project and displaying with the other spring project!

Section 5: Microservices – Services Discovery

* Service Discovery
  + Problem?????
    - How do you find a service rather than hardcoding the URL?
    - You could have multiple instances running of multiple services.
    - Hardcoding URL will be unmanageable in a large application with multiple services and each service having multiple instances.
    - What if a service is not available?
    - How do you manage a failure?
  + Service Discovery is a way once service can find other services.
    - Services can register at a central server and deregister when not available.
    - With service discovery you can monitor and do health checks on services and remove unhealthy instances.
  + Spring Cloud Netflix project
    - <https://spring.io/projects/spring-cloud-netflix>
    - With Spring Cloud Netflix, you can implement Service Discovery!
    - 2 Main Projects:

Spring Cloud Netflix Eureka Server

Spring Cloud Netflix Eureka Client

* + Concept:

**Discovers Service 2**

Discovery Server

Service 1

Registers

Service 2

* + Discover Server is an actively managed registry for service location.
  + Discovery Server monitors deployed services.
* Setting up a Eureka Discovery Server
  + Create a new Java project
    - Name: MyEurekaServer
  + Dependencies
    - Eureka Server
  + After project creation
    - Locate the main entry point MyEurekaServerApplicaiton class.
    - Then Add the @EnableEurekaServer annotation
  + Now, locate the application.properties file
    - Give it a port to run on. server.port=8761
    - Give the app a name: spring.application.name=my-discovery-server
    - Add eureka.client.register-with-eureka=false
    - Add eureka.client.fetch-registry=false
    - We do not need to register the Discovery server with itself. This is why the two properties are set to false.
    - We will be using the discovery server to discover other services.
  + Go to pom.xml
    - Look at the new dependency spring boot added.
  + We are now ready to launch the Eureka server.
  + Open browser at localhost:8761 to see the Eureka dashboard.
  + You will notice not instance because we have not registered any clients with the server.
  + Next, we will register different services with the Discovery Server.
* Registering the Course app as a Eureka client
  + We will add the course app and register it on the Eureka Server.
  + For all clients that will be registered on the Eureka Server it must have the Spring Cloud Netflix Eureka Client dependency.
  + Look on Maven Central and search for the Netflix Eureka Client Stater.
  + Add the Dependency Spring Cloud Starter Netflix Eureka Discovery Client
  + Now in the course Application main class add the Annotation @EnableEurekaClient
  + Now locate the Application properties and add:
    - Spring.application.name=course-app-service
    - Server.port=8001
  + Make sure Eureka Server is running
  + Now go to the main class and run the application.
  + Now go to localhost:8761
  + You will see that the course-app-service is now registered.
* Integrating the Course Catalog app with the Course app through the Eureka Server.
  + Up till this point we have created a Discover Server (Eureka Server) and registered the course app.
  + Now let us invoke the course app from the course catalog app.
  + In the course catalog app add the Eureka Discovery Client dependency.
  + Navigate to the application.properites file and add
    - Spring.appliction.name=catalog-service
    - Server.port=8002
    - Eureka.client.register-with-eureka=false
  + We do not need to register the course catalog app. We will be discovering it from the course app.
  + Now navigate to the catalog controller.
  + Comment out the hardcoding of the URL for all mappings.
    - Line 27: String courseAppURL = "http://localhost:8080/”
  + Add the Using Eureka Client code for all mappings. Or for me just uncomment it.
  + Add to the CourseCatalogApplication the annotation @EnableEurekaClient
    - This will allow the catalog to interact with the Eureka server.
    - Note: This is not registered with the Eureka Server.
  + Now let us run the application.
    - Run the Eureka Server
    - Run the Course App
    - Run the Course Catalog
  + Go to localhost:8761 and you will see the course app is registered but the catalog is not.
  + Now lets go to the browerser
    - Localhost:8002/
    - Localhost:8002/catalog
    - Localhost:8002/first
  + **We have now created a Eureka Server, registered an app, and then discovered that app form another app.**
  + This is how service discover works in the microservices world.

Discovers Course App

Discovery Server

Course Catalog

Register

Course App

Section 6: Microservices – Fault Tolerance

* What happens when the Course App goes down?
  + This will cause everything to crash. Not good…
* Handling failure with Circuit Breaker Pattern
  + We can use the Circuit Breaker Pattern to handle this problem.
  + Spring Could and Netflix Hystrix

If more than 50% of the request fail in a 10 second rolling window, then break the circuit.

Service 2

Service 1

Allow a request every 5 seconds to check if the service is up.

* Implementing Circuit Breaker with Netflix Hystrix
  + Doing this will create a fall back in case the Course App Service goes down.
  + First add the spring-cloud-starter-netflix-hystrix dependency.
  + Now in the course catalog application class or the main entry point add the
    - @EnableCircuitBreaker
  + Locate the CatalogController and add the @HystrixCommand(fallbackMethod = "displayDefaultHome") annotation to the home mapping.
  + Add the fallback to all mappings.
  + Create the displayDefaultHome method.

Section 7: Adding the UserApp Service

Course Catalog Service

Rest

Course App Service

Microservice 1

Microservice 2

Rest

Fetch Course Informaiton from CourseApp and corresponding User information from UserApp and display in the CourseCatalog Service.

User App Service

Microservice 3

* Building the User App
  + Create the UserApp
  + Dependencies
    - Spring Web
    - Spring Data JPA
    - Rest Repositories
    - Eureka Discovery Client
    - Manually add the:
      * Sqlite-jdbc dependency
      * Sqlite-dialect dependency
  + Locate the project in the file directory.
    - Then highlight the path and type cmd. This will put you in the project directory.
    - The type:
      * Sqlite3 user.db
    - Use the userdb.txt to create
      * Courseuser database
      * Then create the user table
      * Then insert the 4 objects
    - Note we are using a composite key so one user cannot enter for the same class twice.
  + Locate the main entry point which is UserAppApplication class.
    - Add the EnableEurekaClient annotation.
  + Create the entity package
    - Add a class called UserId in the entity package. This is for the composite key. Make sure it implements serializable.
    - Add the User class that will map to our database.
    - Add the annotation to the user class
      * @Entity
      * @IdClass(UserIdclass)
      * @JsonIgnoreProperties({“hibernateLazyInitializer”, “handler”})
    - Add the fields that match up with the database.
    - Add the @Id
    - Add default constructor
    - Add pam constructor
    - Add getters and setters
    - Add toString
  + Add Repository Package
    - Add interface UserRepository
  + Add Controller package
    - Add the User Controller
  + Add the SQLite info in the application properties
  + Make sure you give this service a new port to run on. All microservices should always be running in there on ports.
  + Now Run the application and go to the browser and type in:
    - Localhost:8003
* Querying the User App on a specific field in the User Table
  + In the UserRepository add the abstract method
    - Add the findBycourseid abstract method.
  + This will allow you to be able to find all the user that are taking course by its id.
  + Go to the controller and add the mapping for getting all users for a course.
  + Run the localhost:8003/1
* Enhancing the Course Catalog App to fetch data from the User App
  + Let’s integrate CourseCatalog with both apps.
  + We will be fetching course information for the course app service and then fetch corresponding information from the user app.
  + So, both Course APP Service and User App Service are being registered with the Eureka Server.
  + Ok start all applications.
    - MyEurekaServer
    - CourseApp
    - UserApp
    - CourseCatalog
  + Check to see if all is working.
    - Localhost:8001
    - Localhost:8002
    - Localhost:8003
  + Let us add code to fetch users for the first course.
  + We will get the course id and then use that to invoke the user app service.
  + In the CourseCatalog we need to create a user class and map the output of the user app service to that class.
  + Copy the User class from the user app service and paste it into the Course Catalog entity and remove all the database annotations.
  + Go back to the controller in then Course Catalog and define a new User object in the getFirstCourse method.
  + Un comment the code
  + Now, all services should be working
  + Localhost:8002/fist