Overview

For this project, you will use established Microservices techniques to design and implement REST services to support the application described below.

Problem Definition

Create REST services to support a customer management site.

The services should support:

- User registration
- User Logon
- Display, create, update and delete of customer records

Technical Requirements

The REST services should include the following endpoints:

- http:\\server:8080\api
- http:\\server:8080\api\customers
- http:\\server:8081\account
- http:\\server:8081\account\register
- http:\\server:8081\account\token

The Microservices projects should:

- Use the Java Spring Boot framework
- Use Gradle build system
- Include Docker setup

Development and testing environment setup

To allow the teams to succeed in the task at hand, each team should prepare a development environment that includes:

- Java
- Git
- NodeJS
- IDE for developing Java applications
- Gradle
- Docker

The above software is pre-installed on the Linux Project VM.

Project Technologies List

The following technologies will be used in to fulfill project requirements.

Technology	Description	Purpose
Virtualization Platform	Virtual box, VMWare or similar environment	Host a virtual machine that can be used for development and deployment
Linux	Operating System	OS for the VM. Works well with other technologies (Docker, Kubernetes, etc…)
Git	Source code management	Manage source code locally and push to central repository
Java	Programming language	Supports Spring framework that will be used to create RESTful APIs
Spring Boot	Java application framework	Java code library for creation of standalone server applications and RESTful APIs
Eclipse	Integrated Development Environment	Supports source editing for Java applications
Gradle	Build Tool	Building Java applications
React	JavaScript framework	Used to create the provided web browser client application
NodeJS	Programming platform and dependency tools	Supports development/running of the React based client application
Postman	Generic HTTP client	Supports development & testing RESTful APIs
Docker	Virtual container system	Used to create and deploy light weight virtual application runtime environments

The Project Zip

The supplied project zip is named: **WA3489-project-student.zip** should be copied to your development machine and unzipped in a convenient location. Note the location as you will be needing it later.

The Project Zip contains these files and directories:

Filename	Description
`Project-Overview.pdf`	This file
`Application Design.pdf`	Requirements and application design for the main project

Extended Application Design.pdf	Requirements and application design for Optional project extentions
/ProjectAssets	Directory containing solution files

Solution Files

The available project solutions have a number of uses:

- Solutions can be run and tested to see how the app is supposed to function
- Solution code can be inspected for ideas on how to proceed with your own coding
- If you run into trouble with your own implementation you can build on top of a given solution

The solution code comes in the following zip file that is available here:

ProjectAssets\solution-projects.zip

The zip file has the following structure:

```
solution-projects
├───stage02
│   ├───back-end
│   └───front-end
│
├───stage03
│   ├───back-end
│   └───front-end
│
└───stage04
    ├───back-end-auth
    ├───back-end-cust
    └───front-end
```

The front-end projects:

- Include React web applications
- Execute "npm install" in project directory to retrieve dependencies
- Execute "npm run dev" to start the development application server

The back-end projects:

- Include Java Spring Boot REST API applications
- Execute "./gradlew" in the project directory to retrieve dependencies
- Execute "./gradlew bootRun" to compile and run the app in standalone more
- Executing "./gradlew bootJar" to compile the app and create a bootable jar file

Note about project solutions:

When you have a question about how your application should be coded you can refer back to the project solution. Although your application should work like the solution does, your application code may not exactly match that of the solution. Where differences occur you should be able to explain how why you coded the way you did.

Project Stages

Stage 01: Application Design and Work Plan

Description

A virtual machine is provided that includes all software needed for project development. Your first tasks are to get familiar with the development environment and creation of design and work plans.

Tasks

- Review the application design document ("Application Design.pdf"). The design includes a list of components and services along with information about what each will be used for and how they are expected to work together.
- Review the work plan included in the design document. It gives a rough sequence of tasks needed to fulfil the application's requirements.
- Go over any questions you have about the application design and work plan with the instructor.
- Create a local Git repository in your development environment
- Use the Spring Initializer web site to get started creating a RESTful API (https://start.spring.io/)
- Unzip the project you created with the Spring Initializer and load it in an IDE where you can work on it. The dev environment includes Eclipse IDE as well as Visual Studio Code.

Milestones

By the end of this part you should understand the application design and development plans. In addition you should have gotten familiar with the setup of the development VM and created a basic Spring Boot Project.

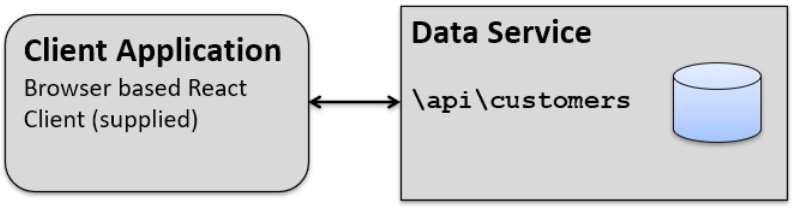
- Review and understand the application design document
- Get familiar with the provided development environment
- Create a Spring Boot project,
- Create a local Git repository for the project

Stage 02: Initial Customer REST API

Description

During development you may use the build capabilities built into the Eclipse Ide but remember that the project must also be buildable from the command line via Gradle. In this stage you should make sure this is the case.

In this stage you will create a customer REST API. The API you create must work with the provided React client. You will find the client application in the "stage02" directory of the ProjectAssets\solution-projects.zip zip file.



Tasks

- Develop a RESTful service that accepts parameters and returns Customer data based on them.
- Design a build process using Gradle that can be used to build your APIs from the command line
- Run the API and use a browser to check the functioning of the created service's GET endpoints
- Commit the completed API code and build.gradle file to your local repository
- Review your work and be able to explain and demonstrate the service and build process

Deliverables

By the end of this part your pair should be ready to explain your build process and demonstrate an initial RESTful API.

- A functional RESTful API that serves hard coded Customer data
- The API should support these endpoints:
    - GET \api
    - GET \api\customers
    - GET \api\customers\{cust-id}

- The API should allow data from the service to be displayed in the provided client application
- A functional Gradle build process

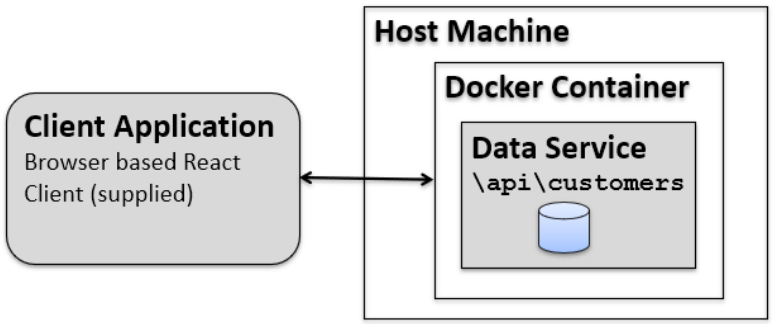
Stage 03: Full CRUD API and Docker

Description

Using the Spring Framework you will develop a statefull back-end API The data being sent to the REST service will be stored in an embedded SQL database. Your RESTful service at this point should CRUD operations against the customer database.

After completing the service it should be tested by running the front end and back end apps in standalone mode on the same machine.

After testing the end-to-end application running in standalone mode you will create images and run them under Docker.



Tasks

- Implement full CRUD capabilities for the Customer API using Spring Boot
- The API should use data from an embedded database
- Use Gradle to build and run the Spring Boot API project
- Run the new API and check the endpoints using Postman Requests
- Commit the completed API code and build.gradle file to your local repository
- Test the front and back end in standalone mode (not using docker)
- Create Dockerfiles for the front and back end and use them to create working Docker images.
- Commit the Dockerfiles to Git
- Bring up the back-end API using Docker
- Verify that the API functions correctly when run using Docker
- Bring up the front end application in Docker
- Verify that all customer functions still work when the app is run under Docker

Hints

- Dockerfile for front-end should be a one-stage Dockerfile that runs the app using "npm run dev"
- Dockerfile for the back-end should be a two stage Dockerfile that builds a jar and then runs the jar usng "java -jar jarname.jar"

Milestones

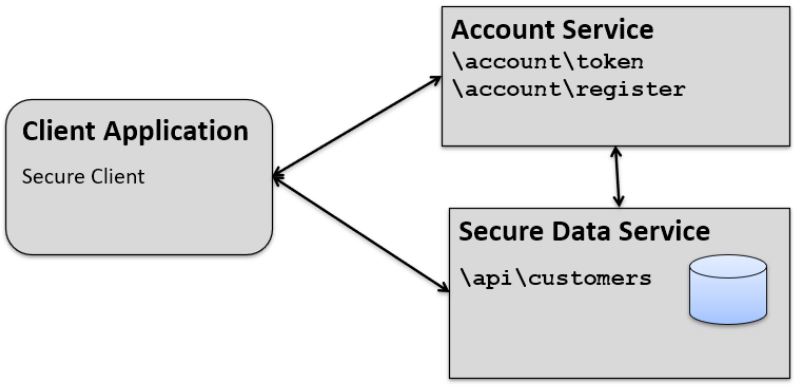
By the end of this stage you should be able to show and explain the completed RESTful API. You should also be able to show the API working when run in a Docker environment.

- Complete full CRUD endpoint support for the customers API
- Implement the API with an embedded SQL data source
- Test the API with the supplied client
- Run the end-to-end application using Docker

Stage 04: Securing the REST API

Description

In this part you will create an API that generates a simulated JWT token for authentication and authorization of your existing Customer API. You will update the Customer API to deny any requests that do not include the simulated token. You will test the APIs using Postman to ensure that the Customer API is able to work properly with the Account API.



Tasks

- Create a new local Git repository for the Account API project
- Implement the Security API using Spring Boot and the Auth0 java-jwt JSON web token library.
- Use Gradle to build the Account API project
- Start up the account API
- Test the account API endpoints using Postman
- Try accessing the Customer API using a token provided by the Account API
- Use Postman to verify the integration points between the two APIs
- Commit the completed API code and build.gradle file to your local repository
- Run the end-to-end application (front-end, back-end-account, back-end-customer)

- Test that the front end works as expected. (register, login, view, create, update and delete customer records)

Deliverables

By the end of this stage you should be able to show the account API working with the Customer API. Time permitting you should also be able to show the new APIs working with the provided front end application.

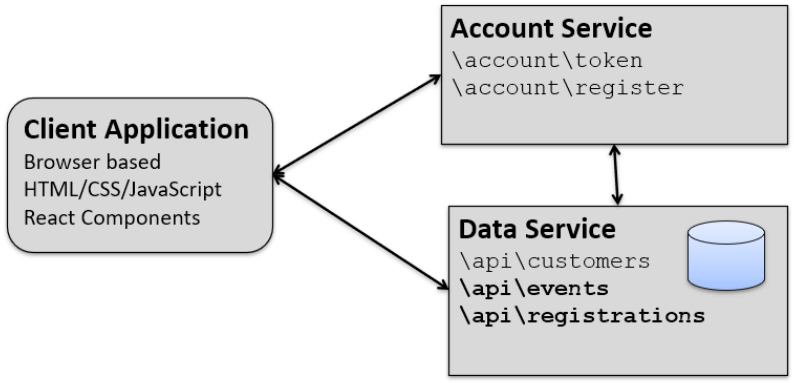
- A functional Docker image/container for the Account API project
- A functional RESTful API that checks credentials and returns simulated JWT tokens.
- An updated Customer API that rejects any requests that don't include a valid simulated token
- Manual Postman procedures to test and verify the account API and its integration with the Customer API

Optional Work: Extending the Application

Description

In this optional part you will extend the application. You will build an event management system on top of the existing application. This will involve the creation two APIs – one that manages Events and another that Manages customer's registering for events.

This section is optional. If you have completed the initial four stages and still have time left before the project ends then you can consider working on it. Extending the application is also something you might consider doing after the course is completed to help you practice the programming skills you've learned.



Requrements and Design

More information about the requirements, design and work steps can be found in the "Extended Application Design.pdf " document.