|  |
| --- |
| **Fitness App** |
| Use Case document for Full Stack development |
| This document covers Software Requirements of Fitness app, along with list of Technologies to be used to develop this Software System, and also includes some details on the Architecture |

Table of Contents

[1. Business Requirement (Fitness App) 2](#_Toc74906679)

[2. Technical Spec – Solution Development Environment 4](#_Toc74906680)

[*Front End Layer* 4](#_Toc74906681)

[*Middle Tier Layer* 4](#_Toc74906682)

[*Database & Integration Layer* 4](#_Toc74906683)

[*Ancillary Layer* 4](#_Toc74906684)

[*Security* 4](#_Toc74906685)

[*Deployment & Infrastructure* 4](#_Toc74906686)

[*Editors* 5](#_Toc74906687)

[Coding Standards and Guidelines 5](#_Toc74906688)

[Designing for Testability 5](#_Toc74906689)

[3. UI Components 6](#_Toc74906690)

[3.1. UI Wireframes 6](#_Toc74906691)

[3.2. UX – Angular Components 7](#_Toc74906692)

[4. Mid-Tier Java Components 8](#_Toc74906693)

[4.1. Architecture Diagram 8](#_Toc74906694)

[4.2. Database Constructs, Model Classes and Entity classes 8](#_Toc74906695)

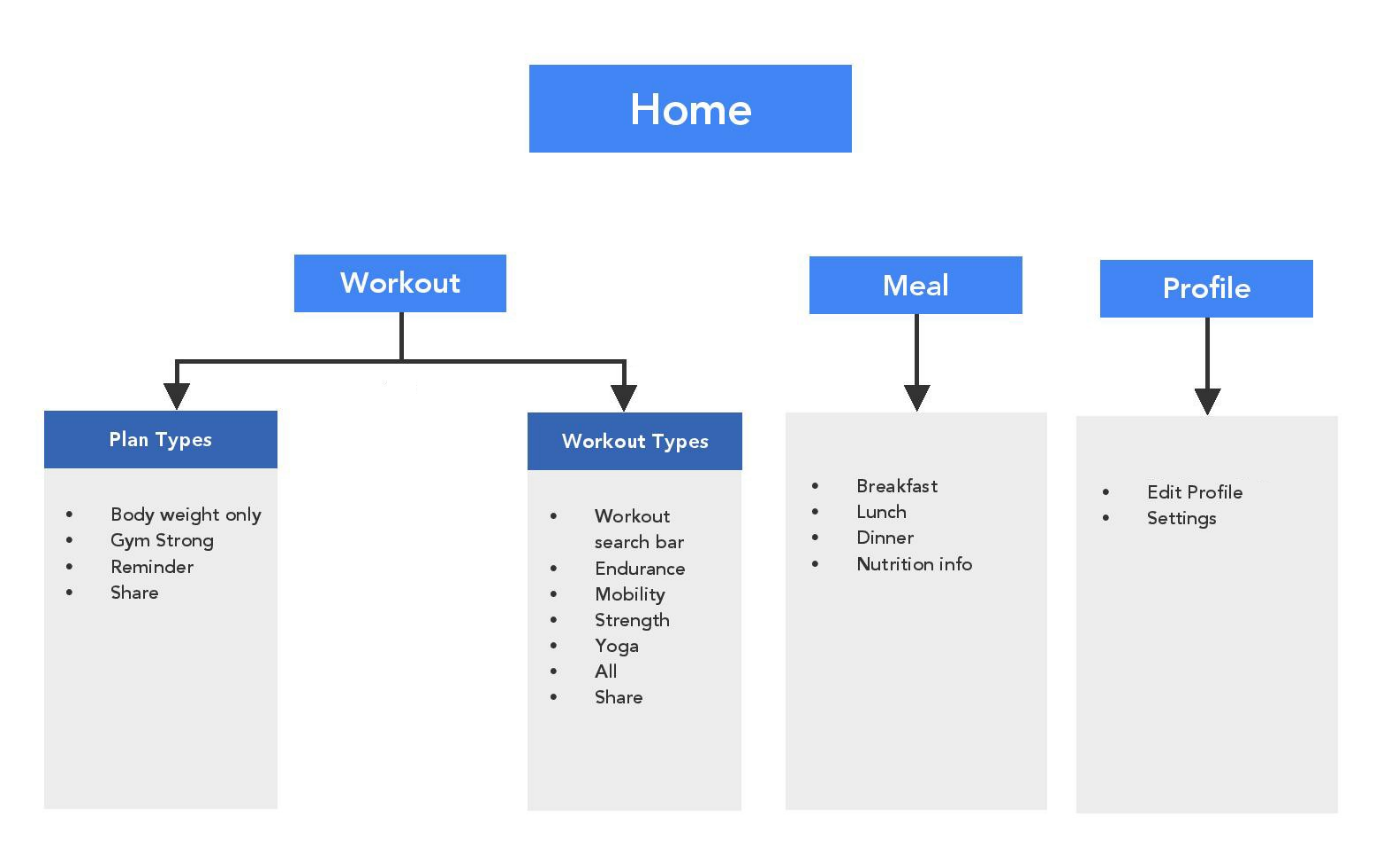
[5. DevOps Activity 9](#_Toc74906696)

[5.1. Jenkins CI/CD 9](#_Toc74906697)

[6. Important Instructions 10](#_Toc74906698)

# Business Requirement (Fitness App)

Create an app where users can store the Fitness they have completed, their meal details and can manage profile.



*Top level use case diagram for the Fitness app.*

# Technical Spec – Solution Development Environment

The technologies included in Full Stack are not limited to following but may consist of:

### Front End Layer

|  |  |
| --- | --- |
| Framework(s)/SDK/Libraries | Version |
| Angular | 8 |
| TypeScript compatible with Angular 8 | 3.4 |
| Bootstrap | 4 |
| CSS | 3 |
| HTML | 5 |
| JavaScript | ECMAScript 2019 / June 2019 |
| JQuery | 3.5.1 / (May 4, 2020) |

### Middle Tier Layer

|  |  |  |
| --- | --- | --- |
| Technology | Framework(s)/SDK/Libraries | Version |
| Java Stack | Spring Boot | 2 or above |
| Spring MVC | 4.3.x |
| JDK | 1.8.0\_211 |
| Maven | 3.6.3 or above |

### Database & Integration Layer

|  |  |  |
| --- | --- | --- |
| Technology | Framework(s)/SDK/Libraries | Version |
| Java Stack | Hibernate | Hibernate 4.1.9 Final and 3.3.2.GA or above |
| Jersey JAX-RS / Spring Restful | 2.x |
| MySQL | 8 CE (Community Edition) |
| MongoDB | MongoDB | 3.4 |

### Ancillary Layer

|  |  |  |
| --- | --- | --- |
| Technology | Framework(s)/SDK/Libraries | Version |
| Source Code Management Tool | GIT | 2.25.1 or above |
| Build Tool/JAVA Stack | Maven | 3.6.3 or above |
| Testing Tool/JAVA Stack | JUnit | 4.x.x |
| Testing Tool/JAVA Stack | Spring Test | 4.x.x |
| Controllers can be tested using Postman Tool | | |

### Security

|  |  |
| --- | --- |
| Name | Version |
| Spring Boot Security | 5.x |
| JWT |  |

### Deployment & Infrastructure

|  |  |  |
| --- | --- | --- |
| Technology | Framework(s)/SDK/Libraries | Version |
| Docker | - |  |
| Apache Tomcat | - | 8.5.4.3 |
| Jenkins(CI/CD) | - | 2.204.5 and above  2.54 is Preferred |
| Node | - | 10.16.0 |

### Editors

|  |  |
| --- | --- |
| Name | Version |
| STS(Spring Tool Suite) | 4.6.0 |
| Visual Studio Code | 1.4.2 |
| Eclipse Jee | 4.11.0 |

### Coding Standards and Guidelines

Throughout the implementation of use case, standard naming and coding guidelines needs to be followed for GUI and Java in order to

* produce easily readable,
* consistent code with proper naming conventions
* achieve robustness,
* maintainable code through proper exceptional handling and logging,
* assist software portability,

Refer these links for more details around coding standards.

* <https://docs.google.com/document/d/1XXMvReO8-Awi1EZXAXS4PzDzdNvV6pGcuaF4Q9821Es/pub>
* <https://itnext.io/clean-code-checklist-in-angular-%EF%B8%8F-10d4db877f74>
* <https://docs.oracle.com/cd/E12517_01/back_office/pdf/141/html/pos_impg2/developmentstandards.htm>
* <https://www.oracle.com/technetwork/java/codeconventions-150003.pdf>

### Designing for Testability

Emphasis should be given to Test Driven Development (TDD) and wherever needed, Integration test cases should also be written.

## 3. UI Components

### 3.1. UI Wireframes

In this Phase you will develop, responsive UI of the application using HTLM5, HTML5 API, CSS3 and Bootstrap/Material. (This Phase does not include development of Angular Components)

#### Below are wireframes which need to be developed in this Phase

1. Login/Signup for Fitness App
2. Fitness App’s Landing Page and Navigation Links
3. Workout - Plan type page
4. Workout types page
5. Meal details page
6. User profile page

Based on the requirements, Front End need to be divided into multiple components to accommodate above Wireframes. Angular Routing can be used to create navigation Links. For Authentication, store JWT token in Local or Session Storage. REST APIs are invoked from the corresponding Services,

As known JWT token is generated on the Server side and received by Client on successful authentication. Angular’s HttpInterceptor can be used to automatically send JWT Token thru Header of every HttpRequest.

To complete this case study, you should be comfortable with basic single page web application concepts including REST and CRUD. You may use angular-cli to create your template project. All web pages need to be responsive.

Ref1: <https://cli.angular.io/>

Ref2: <https://github.com/angular/angular-cli>

#### Deliverables of this Phase

1. Wireframes covering menu options and major webpages, in any format
2. Html templates
3. Screen shots of sample Html templates

### 3.2. **UX – Angular Components**

In this Phase you will develop, UX of the SPA application using Angular and Typescript. You will use Angular, Javascript and Jquery where required.

#### This Phase includes development of

1. Angular components using Templates(Templates are created in Phase1)
2. Angular Routing
3. Angular Services, to invoke REST End points.

Angular Routing feature is used to display Navigation links on Web Pages. When User clicks a Navigation link, the corresponding Component is displayed.

As known, Angular Services is used to interact with the REST end points, using Observable or Promises.

Based on the requirements, Front End need to be divided into multiple components to accommodate above Wireframes. Angular Routing can be used to create navigation Links. For Authentication, store JWT token in Local or Session Storage. REST APIs are invoked from the corresponding Services,

As known JWT token is generated on the Server side and received by Client on successful authentication. Angular’s HttpInterceptor can be used to automatically send JWT Token thru Header of every HttpRequest.

###### Deliverables of this Phase

1. UI screens ready to be integrated with backend REST API endpoints.
2. Demo-able screens with mock data.

## Mid-Tier Java Components

### Architecture Diagram

Architecture with REST Controller, Service, Model & Entity Classes and Repository classes



### Database Constructs, Model Classes and Entity classes

This Phase comprises identifying and developing following components

1. Database Constructs like Tables, Views, Integrity Constraints, Indexing, Sequences.
2. Entity Classes and
3. Model Classes

which are required to be used in the next Phases.

Entity classes are the ones which are mapped with underlying Database tables, Entity classes also includes corresponding mappings between them, such as

* One to One Mapping
* One to Many Mapping
* Many to One Mapping
* Many to Many Mapping



## DevOps Activity

### Jenkins CI/CD

**Jenkins CI/CD:** As already known Jenkins is popular tool to perform CI/CD. When the code is pushed to GIT, build need to be automatically created and deployed. If possible create a Docker image and run the Container on Docker Host

This Phase also includes completion of Integration of Front end with Mid Tier.

#### Assessment Deliverables

1. Jenkinsfile, Jenkins screenshots
2. Sharing github repository details with mentor

# Important Instructions

Adhere to the design specifications mentioned in the case study.

1. Please make sure that your code does not have any compilation errors while submitting your case study solution.
2. The final solution should be a zipped code having solution. Solution code will be used to perform Static code evaluation.
3. Implement the code using best design standards/family Design Patterns.
4. Use Internationalization for all the labels and messages in Rest API Development.
5. Do not use System out statements or console.log for logging in Rest API and FrontEnd respectively. Use appropriate logging methods for logging statements/variable/return values.
6. If you are using Spring Restful or Jersey JAX-RS to develop Rest API, then use Maven to build the project and create WAR file.
7. Write web service which takes input and return required details from database.
8. Use JSON format to transfer the results.