Fitness Tracker

Contents

[1 Problem Statement 2](#_Toc78266661)

[2 Proposed Fitness Tracker Wireframe 3](#_Toc78266662)

[2.1 Home/Landing page 3](#_Toc78266663)

[2.2 View Appointments 3](#_Toc78266664)

[2.3 Place Appointment 4](#_Toc78266665)

[3 Application Architecture 5](#_Toc78266666)

[4 Business-Requirement: 6](#_Toc78266667)

[5 Rubrics/Expected Deliverables 8](#_Toc78266668)

[5.1 Rest API (Products & Frameworks -> Compute & Integration): 8](#_Toc78266669)

[5.2 Database (Products & Frameworks -> Database & Storage): 8](#_Toc78266670)

[5.3 API Documentation (Products & Frameworks -> Compute & Integration): 8](#_Toc78266671)

[5.4 Log/ Monitoring (Products & Frameworks -> Governance & Tooling): 8](#_Toc78266672)

[6 Platform 10](#_Toc78266673)

[6.1 Compute 10](#_Toc78266674)

[6.2 Compute, Identity & Compliance, Security& Content Delivery 10](#_Toc78266675)

[7 Methodology 10](#_Toc78266676)

[7.1 Agile 10](#_Toc78266677)

# Problem Statement

Fitness Tracker is SPA (Single Page Application) for placing a request for appointments, view appointments, contact us.

The core modules of fitness tracker app are:

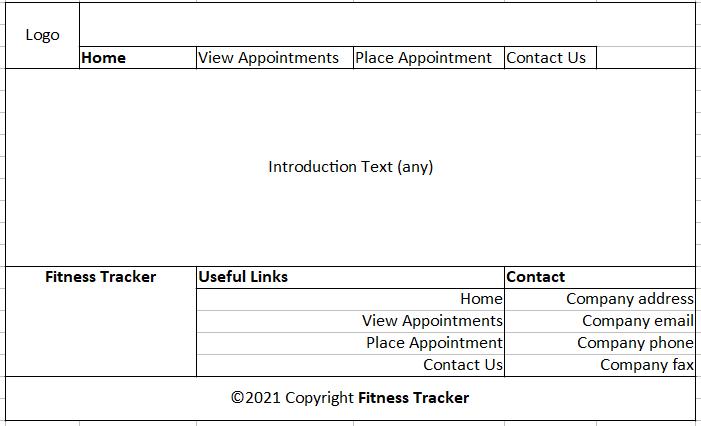
1. Landing Page
2. View Appointments
3. Place Appointment

The scope includes developing the application using toolchain mentioned below.

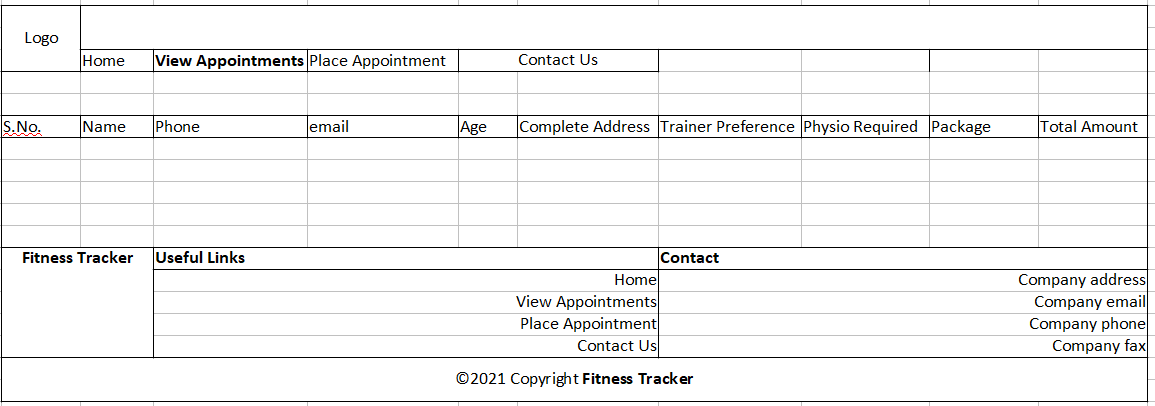
# Proposed Fitness Tracker Wireframe

UI needs improvisation and modification as per given use case.

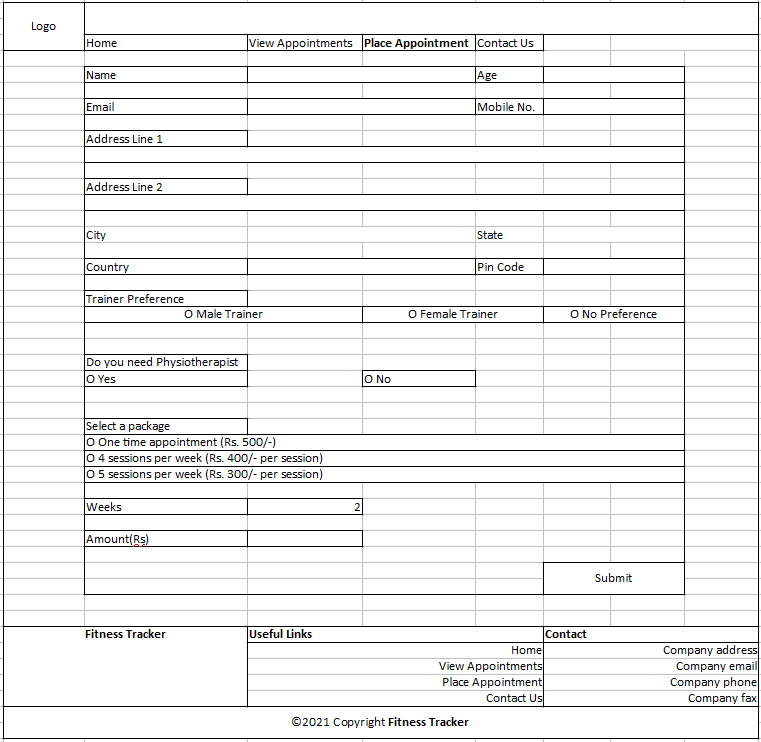
## Home/Landing page



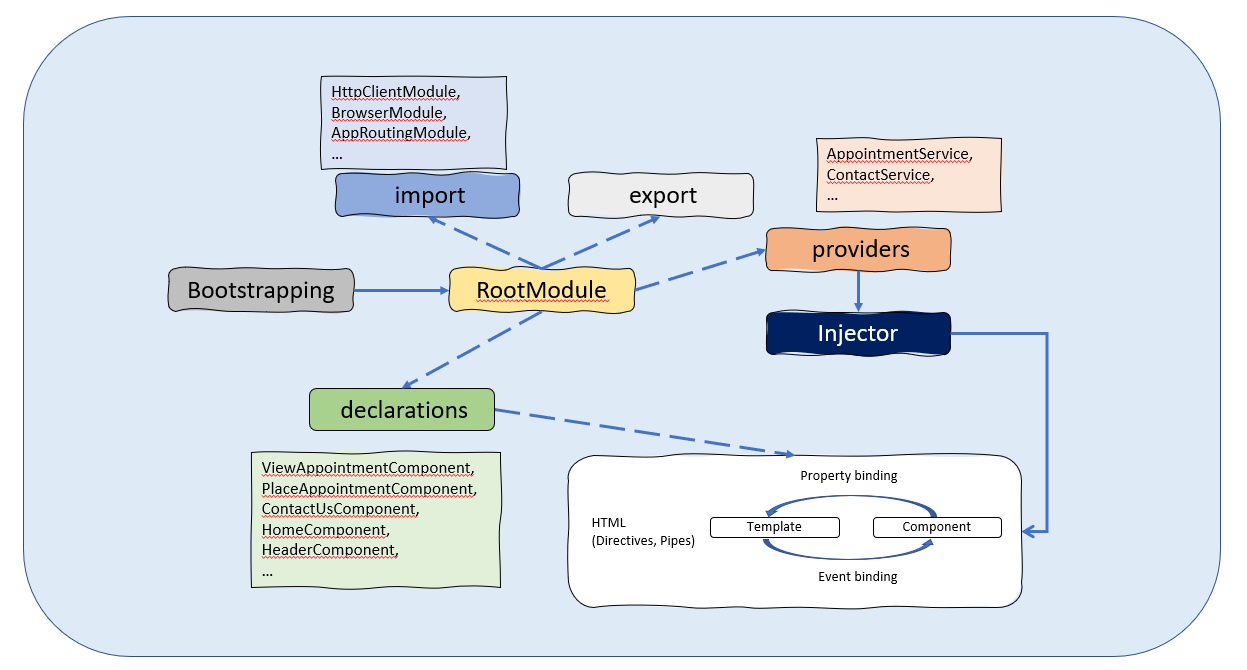
## View Appointments



## Place Appointment



# Application Architecture



Presentation (Products & Frameworks)

Code Quality (Engineering & Quality)

Sonarcube

GIT

Docker

Spring Boot Rest API

Junit

Logstash

Compute & Integration (Products & Frameworks)

OpenAPI/Swagger

Actuator

Compute & Integration (Products & Frameworks)

Config server

DB

Database & Storage (Products & Frameworks)

# Business-Requirement:

As an application developer, develop frontend, middleware and deploy the Fitness Tracker App (Single Page App) with below guidelines:

|  |  |  |
| --- | --- | --- |
| User  Story # | User Story Name | User Story |
| US\_01 | Landing Page/Home Page | As a user I should be able to visit the home page as default page.  Acceptance criteria:   1. User can click any button given in menu bar. |
| US\_02 | Post Appointment | As a user I should be able to post an appointment  Acceptance criteria:   1. As a user I should be able to furnish following details at the time of placing an appointment    1. Name    2. Age    3. Email    4. Mobile No    5. Address Line 1    6. Address Line 2    7. City    8. State    9. Country    10. Pin Code    11. Trainer Preference    12. Physiotherapist requirement (Yes or No)    13. Select a package    14. Weeks    15. Amount (Disabled) 2. Weeks number type input box should be visible when 2nd or 3rd package option is selected. 3. Amount should be disabled and should be calculated automatically based on selected package. 4. All details fields must be mandatory. 5. Address line 2 may contain same address as address line 1. 6. Email & Mobile must be unique. 7. If any constraint is not satisfied, validation message must be shown. 8. A success or failure message should be visible after submit button clicked. |
| US\_03 | View Appointment | As a user I should be able to view all appointment requests.  Acceptance criteria:   1. View all appointment requests. 2. Message should be visible if no appointment is available to show. |
| US\_04 | Post Contact Us | As a user I should be able to post a feedback/query/message  Acceptance criteria:   1. As a user I should be able to furnish following details at the time of filling contact us form    1. Name    2. Email    3. Phone    4. Message 2. Message should not go beyond 200 characters. 3. All four fields must be mandatory. 4. A success or failure message should be visible after submit button clicked. |

# Rubrics/Expected Deliverables

## Rest API (Products & Frameworks -> Compute & Integration):

* 1. Use Spring Boot to version and implement the REST endpoints.
  2. Implement HTTP methods like GET, POST, PUT, DELETE, PATCH to implement RESTful resources:

|  |  |  |
| --- | --- | --- |
| GET | /api/v1.0/fitnesstracker/contacts | Get all contacts |
| GET | /api/v1.0/fitnesstracker/appointments | Get all appointments |
| GET | /api/v1.0/fitnesstracker/appointments/<email> | Get all appointments of a user |
| POST | /api/v1.0/fitnesstracker/appointments | Post new appointment request |
| PUT | /api/v1.0/fitnesstracker/appointments/<id> | Update appointment request |
| DELETE | /api/v1.0/fitnesstracker/appointments/<id> | Delete appointment request |

* 1. Use necessary configuration in place for REST API in application.properties or bootstrap.properties or application.yml; whichever is applicable.
  2. Package Structure for Spring Boot Project will be like com.fitnesstracker.\* with proper naming conventions for package and beans.
  3. Use configuration class annotated with @Configuration and @Service for business layer.
  4. Use constructor-based dependency injection in few classes and setter-based dependency injection in few classes.
  5. Follow Spring Bean Naming Conventions

## Database (Products & Frameworks -> Database & Storage):

1. As an application developer:
   1. Implement ORM with Spring Data MongoRepository and MongoDB. For complex and custom queries, create custom methods and use @Query, Aggregations (AggregationOperation, MatchOperation, AggregationResults), implementation of MongoTemplate etc as necessary.
   2. Have necessary configuration in place for REST API in application.properties or bootstrap.properties or application.yml OR Java based configuration; whichever is applicable.

## API Documentation (Products & Frameworks -> Compute & Integration):

1. As an application developer:
   1. Document REST endpoints with OpenAPI or Swagger

## Log/ Monitoring (Products & Frameworks -> Governance & Tooling):

1. As an application developer:
   1. Containerize the complete application, which includes front-end, middleware and kafka (consumers and producers) using docker and Dockerfile.
   2. Use .dockerignore as necessary to avoid containerizing un-necessary packages.
   3. Integrate Spring Boot Actuator with Prometheus and Grafana to monitor middleware.
   4. Implement logs with logstash.
   5. Open the preconfigured Logstash in Kibana and check if it successfully connect to Elasticsearch Server.

# Platform

## Compute

* + - 1. Use ECS CLI (as an alternative to AWS Management Console) for container management and deployment of spring boot application. You should be able to explain and demonstrate the same in interview.
      2. Use NoSQL instance of AWS DynamoDB/Aurora(SQL) as a database for the Tweet Application

## Compute, Identity & Compliance, Security& Content Delivery

1. Use AWS Lambda and AWS Aurora to build a backend process for handling requests for Tweet App.
2. Use Serverless Java Container using AWS ECS and run the tweet app created with Spring Boot inside AWS Lambda.
3. Use Amazon API Gateway to expose the Lambda functions built in the previous step to be accessible on public internet.
4. Use AWS ELB to configure the auto-scaling container instances.
5. Configure AWS SNS to issue messages whenever a ELB scales-up and scale-down container instances

# Methodology

## Agile

1. As an application developer, use project management tool along to update progress as you start implementing solution.
2. As an application developer, the scope of discussion with mentor is limited to:
   1. Q/A
   2. New Ideas, New feature implementations and estimation.
   3. Any development related challenges
   4. Skill Gaps
   5. Any other pointers key to UI/UX and Middleware Development