# Todo

# Scope

The purpose and scope of this documentation is to provide and overview of the Todo Service along with information like how the functionality was implemented. This document will also list the possible functionality for next version.

# Introduction

Todo-Service app is a multi-user Spring Boot app with a possibility to have a full fledge Angular front end in future. The app allows the user to see a list of existing to-dos for the user and add/update/delete to-dos.

# Technology

## Backend:

1. Spring 5 and Spring Boot 2
2. Maven
3. Java 8
4. Swagger for documentation and end user testing (in the absence of front end)
5. Hibernate – H2 Database (in memory)
6. Junit 5 and Mockito
7. SpringBootTest and MockMvc for Integration Testing
8. STS as IDE
9. Lombok
10. Jacoco
11. Postman

## Frontend

1. Angular 9
2. VSCode as IDE
3. NPM and Node
4. Jasmine for Testing

# Git Repository

I have provided the link to git repo because as part of test. I have only developed the basic backend app with ability to be tested via swagger by user.

The front end is still in WIP.

<https://github.com/Ranjana7/todo.git>

# Thought Process

I started developing the app by utilizing the start.sporing.io to create a simple spring boot app with maven, spring jpa etc.

The first draft of the app was to enable the functionality to be able to do CRUD for to-do.

The second draft of the app was to enable the to be able to perform CRUD for to-do based on user.

I followed TDD and started the development by writing model todo and unit test for a todo-service that can get a list of todos from the H2(in memory) database and slowly build it into the app that can perform CRUD based on User information.

In a real-world scenario, we should keep the user view of the request separate from the Service view. However, as this is just a POC and also, is not a very large application with complex functionality. I decided to utility the todo model for both user view and Service view. This saved me some time on applying mapping using libs like model-mapper (which I have used in my current project in Fidelity).

I have also used Lombok to make the code cleaner and to avoid manually adding getters and setters and for logging.

Central Exception handling is being implemented via Spring’s @ControllerAdvice and @ExceptionHandler to ensure clean code and proper error reporting to user.

Swagger is used for documentation and can be easily used by user for testing via following url while the app is running on localhost:

[http://localhost:8080/swagger-ui.html#](http://localhost:8080/swagger-ui.html)

The app is packaged as war as requested and run on any machine without the need of any additional environment setting or an application server. This because possible because of the use of Spring boot that provides embedded tom cat server.

By using Spring, we have applied a lot of design patters for example DI/IOC, Factory Pattern, Singleton Pattern etc. Also, with this I have followed standards and good practice for url naming, property inject vs constructor injection etc.

Due to Time constraints, I have not implemented security. However, it will not be very complex to implement security using Spring-Security and Spring OAuth2 clinet. There are many ways to do this, we can either use the open source authorization providers like okta, auth0 or can create our own Authorization micro service.

# What’s Next

1. Develop a working front end to deliver MVP
2. Implement Authentication and Authorization
3. Establish ManyToMany relationship between todos and users
4. Role based Access?
5. Ability to add a user by adding a microservice
6. Performance Test
7. Dockerize the Frontend and Backend
8. Deploy to AWS – for starter we can use s3 bucket.