**Account Management**

Implementation Details

By: Saad Shahid

Dated: 09 June 2019

Table of Contents

[Overview 3](#_Toc10995958)

[Approach 3](#_Toc10995959)

[System Flow 3](#_Toc10995960)

[ User Registration 3](#_Toc10995961)

[ Login to Accounting System 4](#_Toc10995962)

[ Accounting System Dashboard 5](#_Toc10995963)

[ Accounting System User List 5](#_Toc10995964)

[ Accounting System User Account 6](#_Toc10995965)

[ Hal Browser Display at server side 9](#_Toc10995966)

[ Swagger 9](#_Toc10995967)

# Overview

Application has been designed to support micro-service architecture where User Interface and Services can be executed independently on different docker container using nginx server for front end and tomcat for backend. Application support complete authentication and authorization using spring security. Password has been kept encrypted using custom encrypted algorithm to add addition security of the application. Some of the standard that has been implemented in the system are as following

**Front-end**

1. Mock-server has been written so to allow user to keep on using application without backend support
2. E2e execution cycle that support cucumber test and protractor to provide e2e automation execution
3. Interceptor, Local storage, principal has been used in front-end
4. Hateoas services has been used in front-end
5. Routing and child routing has been used in application.
6. Preload strategy has been used in code

**Back-end**

1. Liquibase has been used for schema version control and schema creation
2. Hal-browser has been integrated in application
3. Swagger has been used to support custom end-point
4. Lambok has been used to support code quality
5. Pmd validation is present to support code quality
6. Test-cases are written to support CICD process
7. Added spring security for authentication/authorization
8. Utilize JPA repository

# Approach

Application has been designed to support micro-service architecture where User Interface and Services can be executed independently on different docker container using nginx server for front end and tomcat for backend. In the application UI can be access through link <http://localhost:5200> and services can be access through <http://localhost:8080>

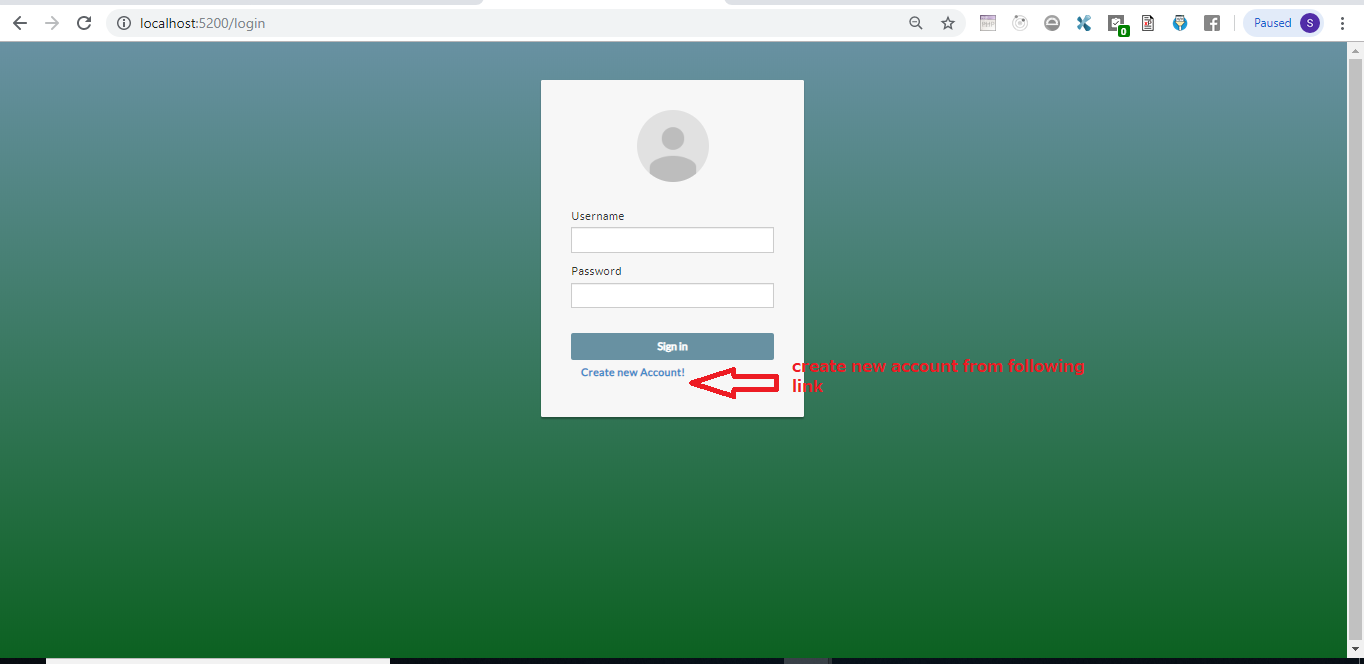
# System Flow

Application has been design to support real world scenario where user need to be registered before using the system.

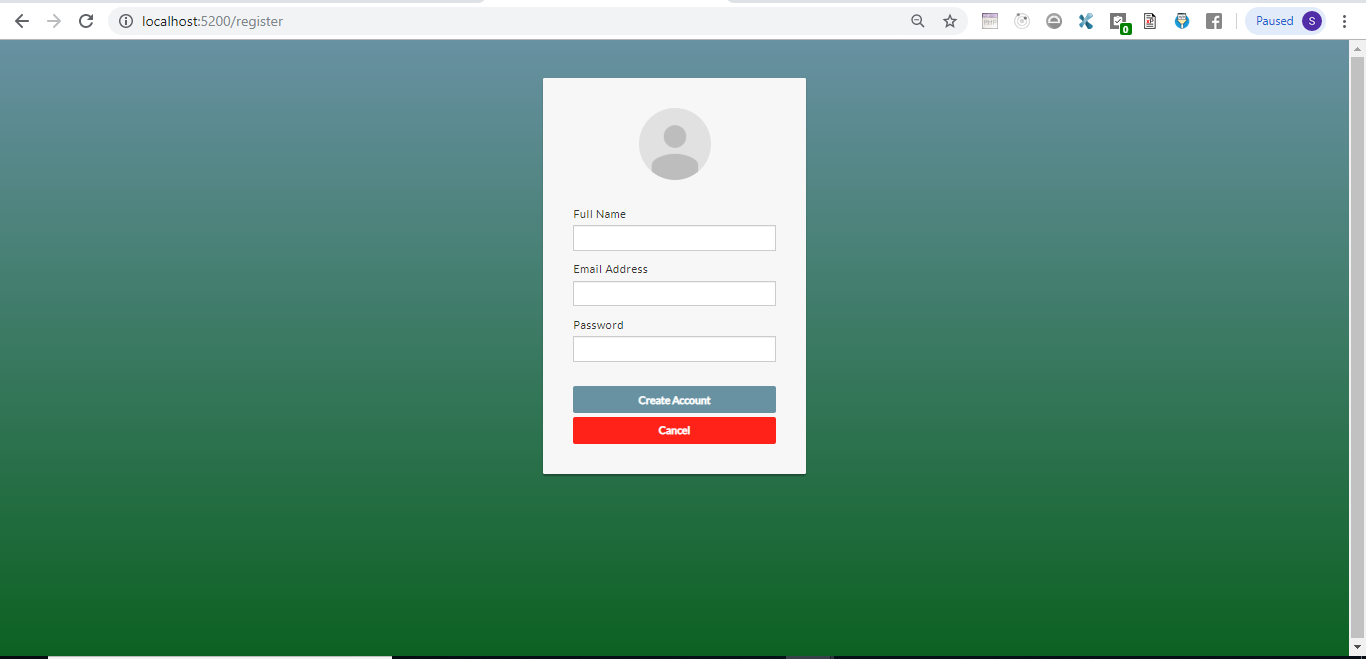
Start application server and application ui from step given in readme file.

Step by step screen execution is given below

# User Registration

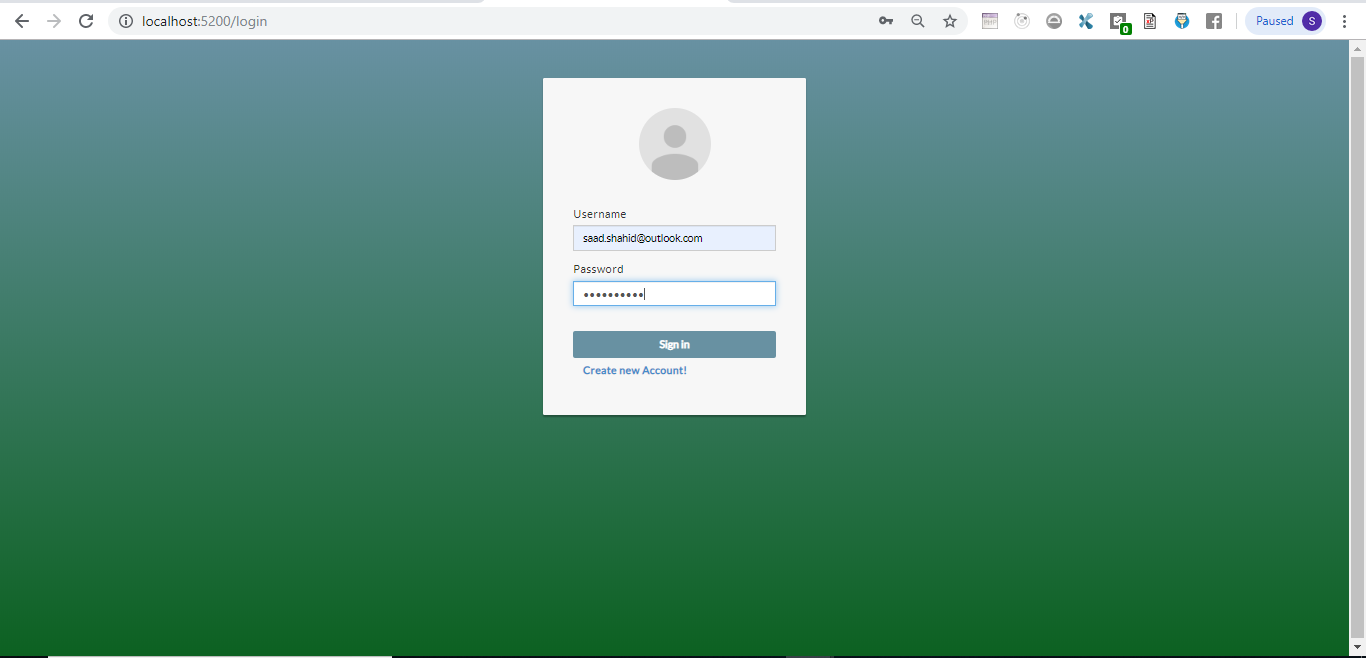


Provide all the information in the form to create user account



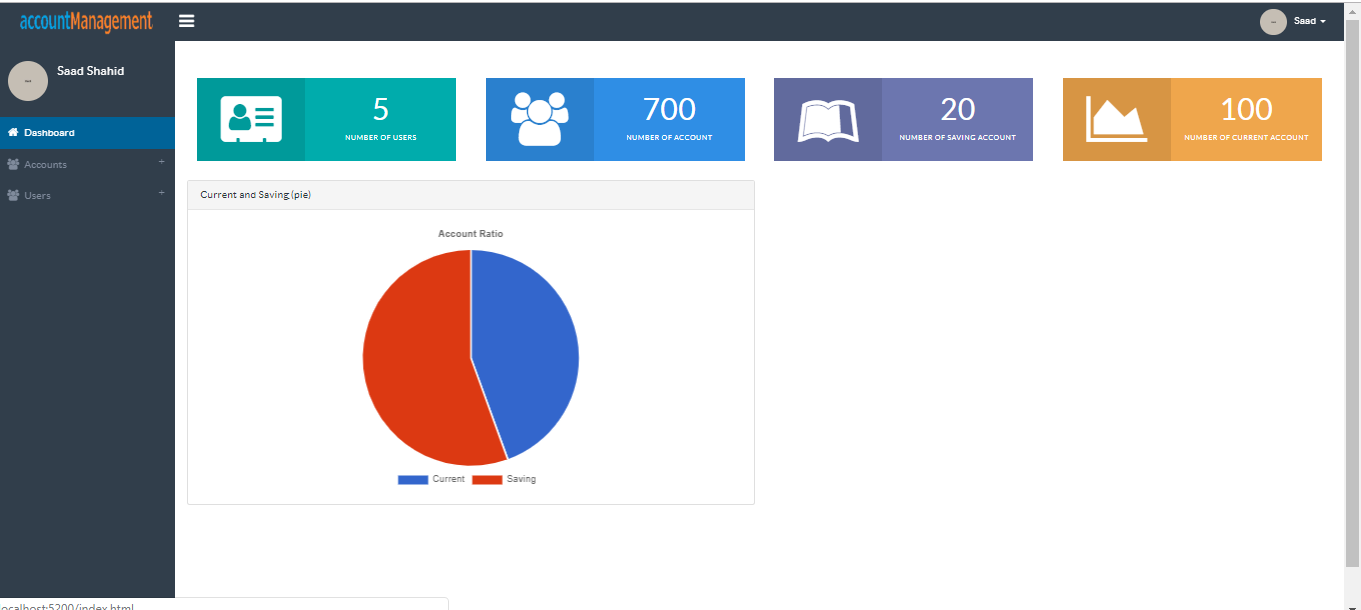
# Login to Accounting System

Provide email address and password from the screen below. Information should be the same as that



# Accounting System Dashboard

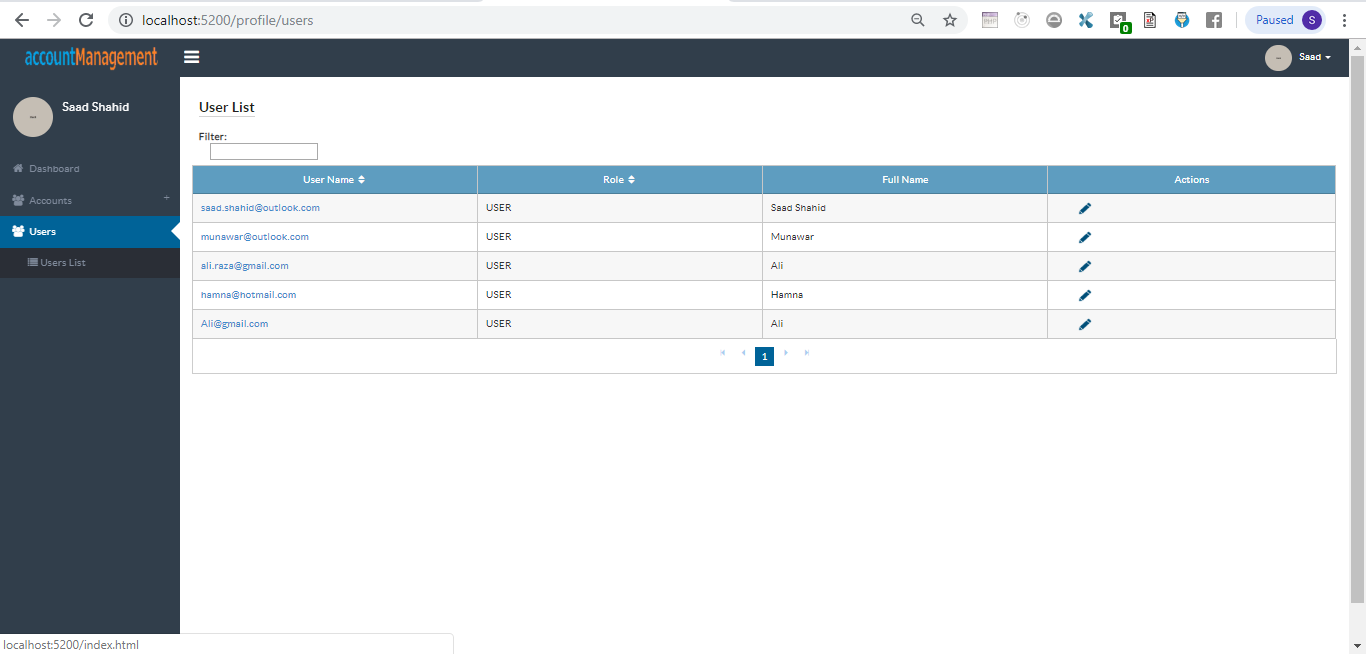
When user login to the screen, they can see following screen. If screen doesn’t looks fine ctrl+F5 will fix the problem



Data on the screen is fictitious and data is not retrieved from backend services

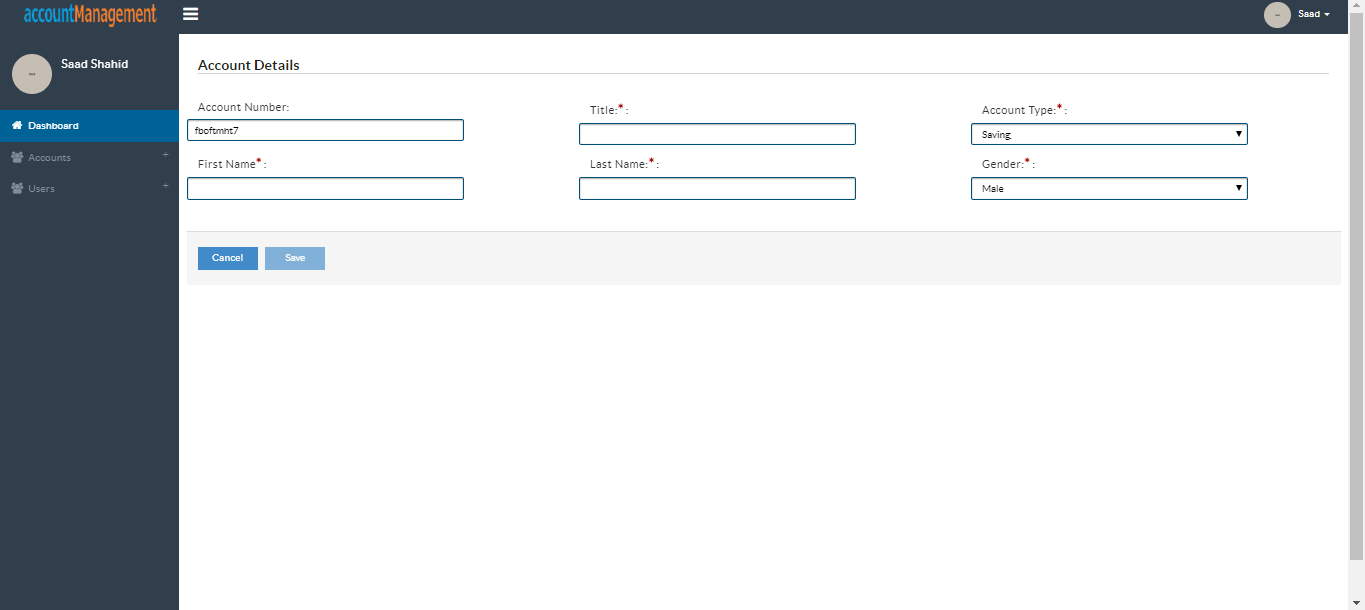
# Accounting System User List

Click on User list option from

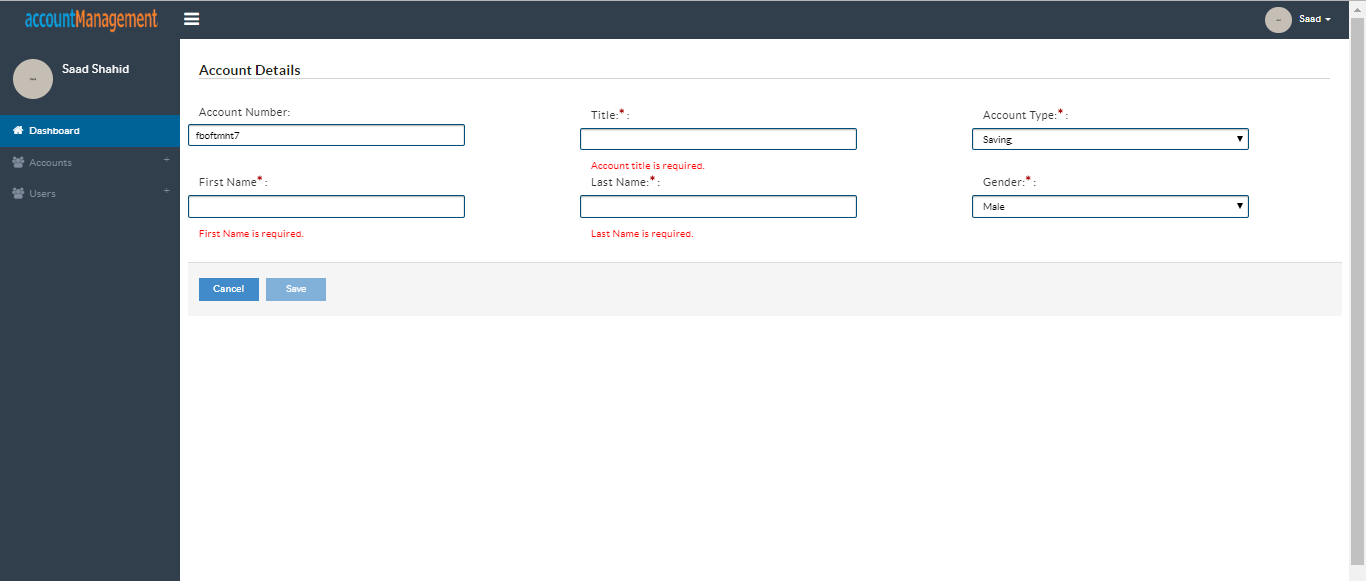


# Accounting System User Account

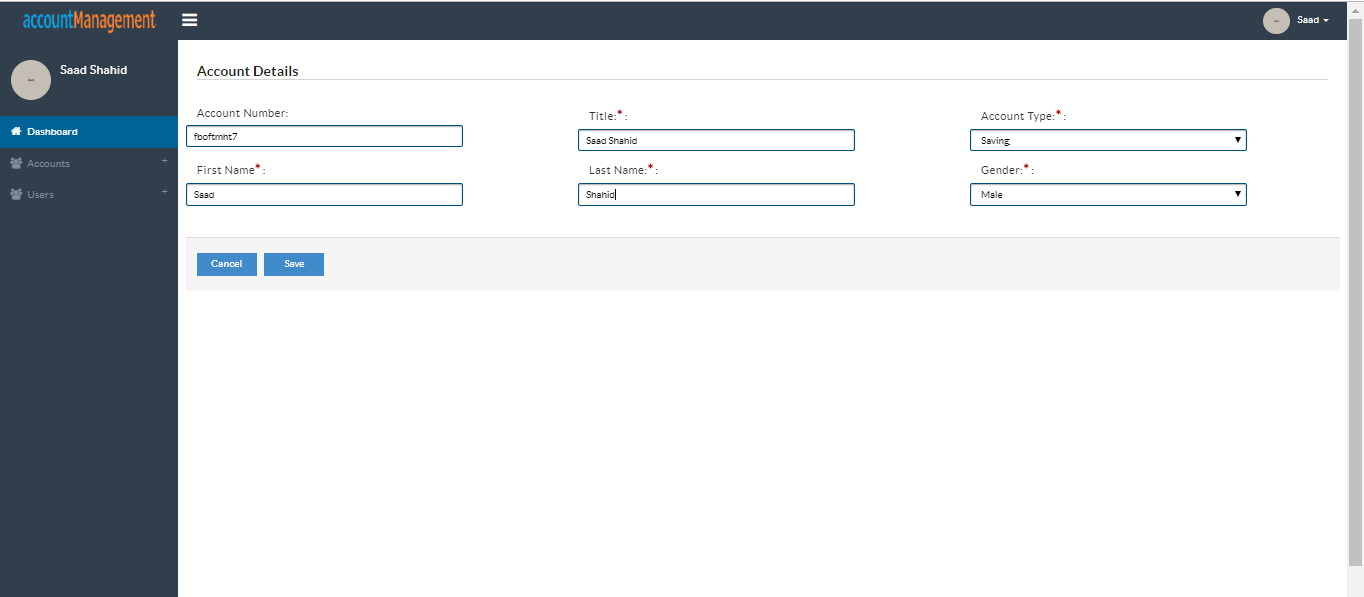
On clicking action () next to user, you can create account against the user. Screen will look as following



Account number cannot be edited and string is created randomly. All the other field on screen is required. Validation check can be seen as below

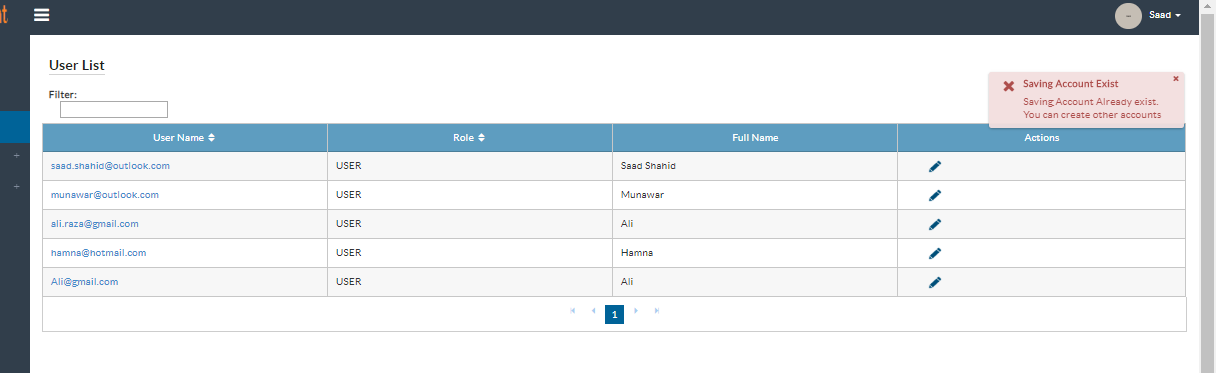


Account type in dropdown is retrieved from backend service. Whatever the value will be present in account\_type table that value will be shown on screen.

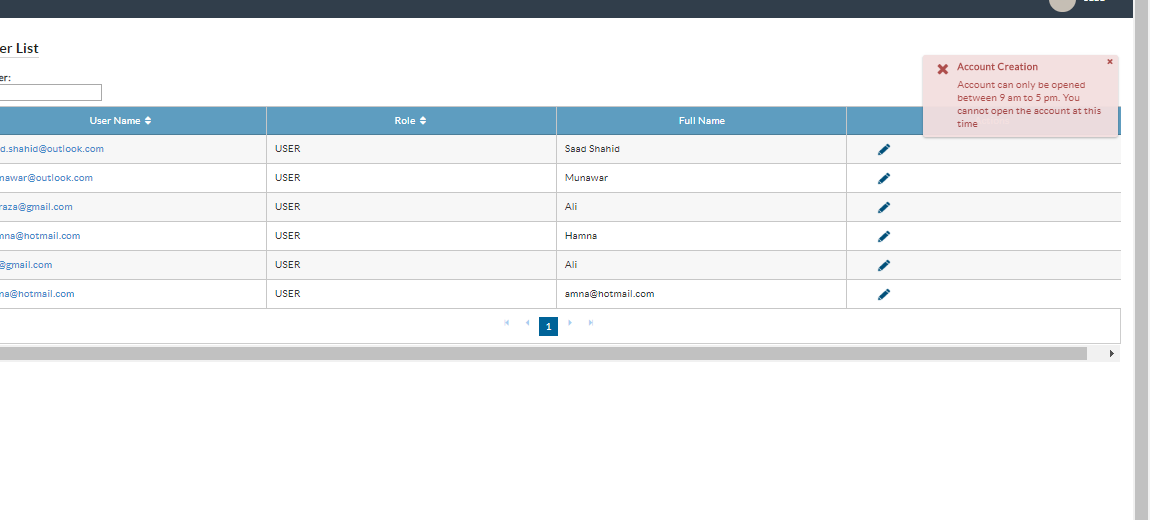


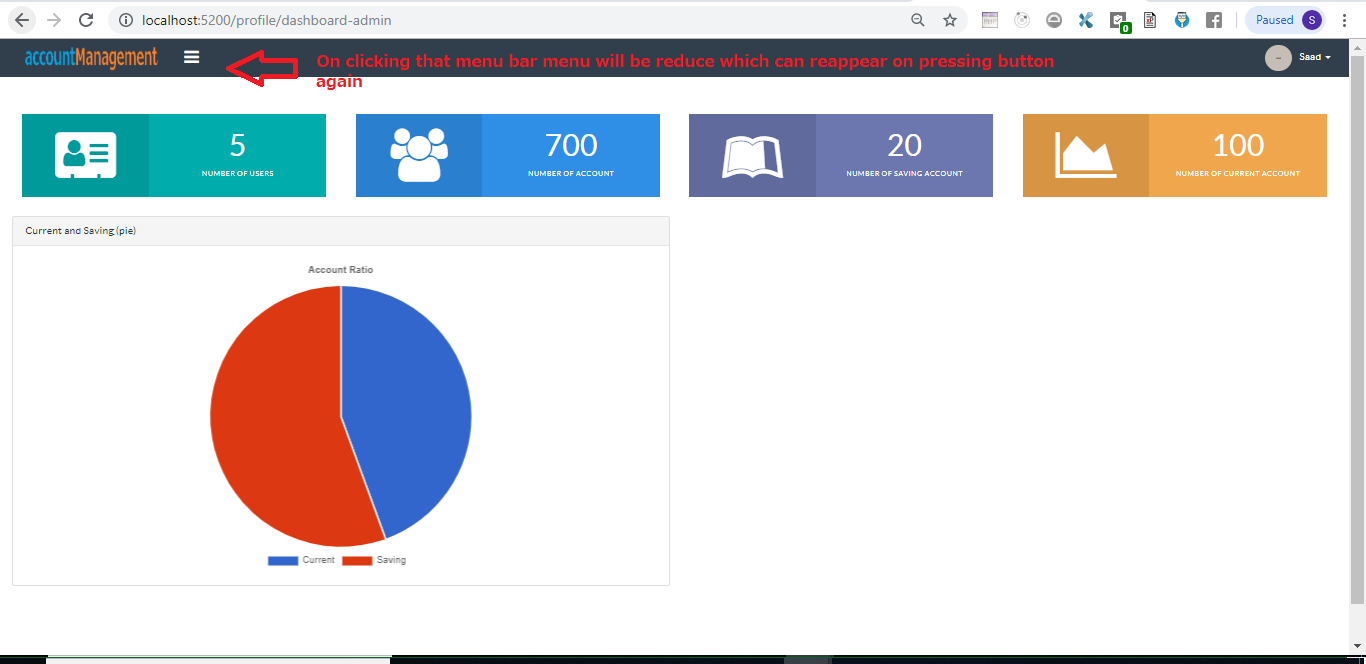
Save button gets enable when validation on the screen is performed successfully.

If saving account already exist user will be shown message as following

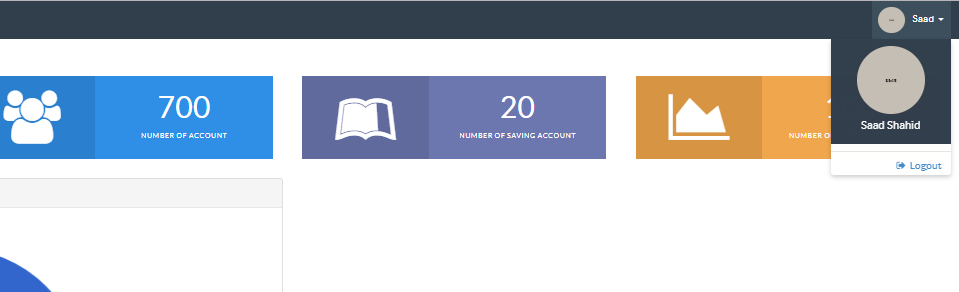


If time on machine is not between 9 am to 5 pm. Message will be shown as below





User can logout from menu as shown on screen below



# Hal Browser Display at server side

You can open hal browser by link below

<http://localhost:8080/accountmanagement>

# Swagger

User can check custom end-point from link below

<http://localhost:8080/swagger-ui.html>