# Developing A Multi Tenants REST API with Spring Boot 3

## Spring Security 6, OAuth 2 and KeyCloak 21

BINIT DATTA

Edition 1

# Developing A Multi Tenants REST API with Spring Boot 3, Spring Security 6, OAuth 2 and Keycloak 21

BINIT DATTA

# Preface About the Book

Spring Security is a subject that is so vast that several books may need to be written to cover if fully. However, modern applications rarely need everything that Spring Security provides to secure themselves. In this book, we will deal with a extremely popular use case to develop a Multi-Tenant Oauth2 Resource Server, a Multi-Tenant Oauth2 Client and a single Tenant Oauth2 Resource Server. While we will test the Multi-Tenant Resource Server with the Spring Boot Thymeleaf UI Client, we will use Postman to test the single Tenant Resource Server.

Today Application Security is more important than ever before. As different nations now keep cyberwarfare as a legitimate option of attacking their enemies or defending their resources, the various cryptic terminologies that float around application security need to be explained in simple English. In this book, we have described all terminologies including Oauth2 Resource Server, Roles, Grant Types, Asymmetric Encryption, and others in easily understandable concepts.

Despite Spring Security's robust support for multi-tenancy, I did not find a good source / book that describes a range of related topics starting from Spring Security, Oauth2 Grant types, Roles, Spring Boot 3 together. While demonstrating Spring Security Oauth2 Multi-Tenant features, we will do using real life MySQL backed JPA and use modern Thymeleaf Advanced Themes to build the UI. That is why I decided to write this book. Hope you will find it quite useful.

# Acknowledgements

I want to acknowledge the invaluable contribution hundreds of non-technical business users have made in shaping my thought process through the last 30 years. They have helped me believe that with all the great importance all kinds of technologies get today, they are still a means towards the common business goal at the end of the day. That mindset has helped me understand the significant similarity between competing platforms like Java and .NET, Spring Framework and Angular, Maven, Gradle, NuGet, NPM and Gulp, and the like.

I would also like to acknowledge the great effort made by Arupa in reviewing this entire book, trying the technical example independently to make sure they are accurate. Her support, silent inspiration, sacrificing weekends cannot be measured in measurable terms.

Lastly, I would like to acknowledge the rich contribution made by Mr. Biswajit Das, one of my earliest mentors. Mr. Das graduated from one of India's most prominent educational institutions, the Indian Institute of Technology, Kharagpur. Google CEO Sundar Pichai is also an alum of the same institution. Mr. Das is a Gold medalist from IIT, Kharagpur. Despite his stellar educational background, how he focused on the humblest business users/operators to the most important stakeholders alike made a deep impression on me that still motivates me to work for the users and apply the latest and greatest technology for them.  The other thing that greatly validated and inspired my own passion for learning new technology relentlessly was seeing Mr. Das do the same as a number of high-level positions (that could be non-technical) remained available.

# About the Author

Binit Datta has over thirty-one years of in-depth experience in business computing. He is an Enterprise Architect at home with both business and technology professionals using the latest and most remarkable cutting-edge technologies. He draws heavily growing up in the 90s, where lack of job divisions helped him understand the depth of business requirements and then design, build, and implement systems all by himself and his colleagues. His decades old experience directly interacting with end customers and stakeholders of all stripes, eliminates the disadvantage of only knowing and focusing on technology alone without knowing the relevance of their application.

Binit has spent the last ten years architecting and leading technology teams building modern high traffic eCommerce websites and scalable enterprise APIs / applications for multiple fortune 50 companies in AWS and Azure Cloud environments. Continuing from his multiple comfort zones, he spearheaded in User Interface Feasibility, Usability and Architecture, and Microservices based REST API Architecture (CRUD and CQRS), Security-related discussions, Event-Driven Streaming Architectures, among others. While his AWS Cloud Certifications (AWS Solution Architect Professional) prove his Cloud credibility's, he has led multiple real-life Cloud Migration Programs to enrich his Cloud experience.

# Table of Contents

# Source Code

The source code for the book is available in the following three GitHub repositories.
https://github.com/binitdatta/spring-boot-3-multi-tenant-resource-server-keycloak
https://github.com/binitdatta/spring-boot-3-multi-tenant-oauth2-client
https://github.com/binitdatta/spring-boot-3-single-tenant-resource-server
If and when you find any issues that you want to report, go to the following and create an issue and I will respond asap.
Note:

- **Please always treat the source code from GitHub as the source of truth, now the code from the book pages**

# 1. Introduction

With the release of Spring Boot 3 several aspects of the basic Spring Data and Servlet Development has changed. Spring Boot 3 makes it mandatory to use JDK 17. However, the Java Persistence API interface and the Servlet classes have been moved from the previous java named packages to the Jakarta namespaces. Besides, Spring Security 6 also have changed several aspects of the java configuration. To keep pace with the changes, we will develop several applications in this book shown as under

- A Multi-Tenant Spring Boot 3 Spring Security 6 Oauth2 Resource Server
- A Multi-Tenant Spring Boot 3 Spring Security 6 Oauth2 Client Application
- A Single-Tenant Spring Boot 3 Spring Security 6 Oauth2 Resource Server
- A Spring Boot 3 REST API for showing One To Many / Many To One JPA Relationship CRUD Functionality
- Aspect Oriented Programming and Event Handling
- Central Exception Handling
- Thymeleaf 3 Fragment and Layout enabled Advanced UI with Drop Down Menus

We will use concepts such as Identity and Access Management (IAM), Oauth2, Open ID Connect (OIDC) and KeyCloak IAM as an Open-Source Popular authorization server for Oauth2 and OIDC. However before we further, it is critical that we understand the concepts first.

## 1.1 What we will Develop

The home / landing / login screen

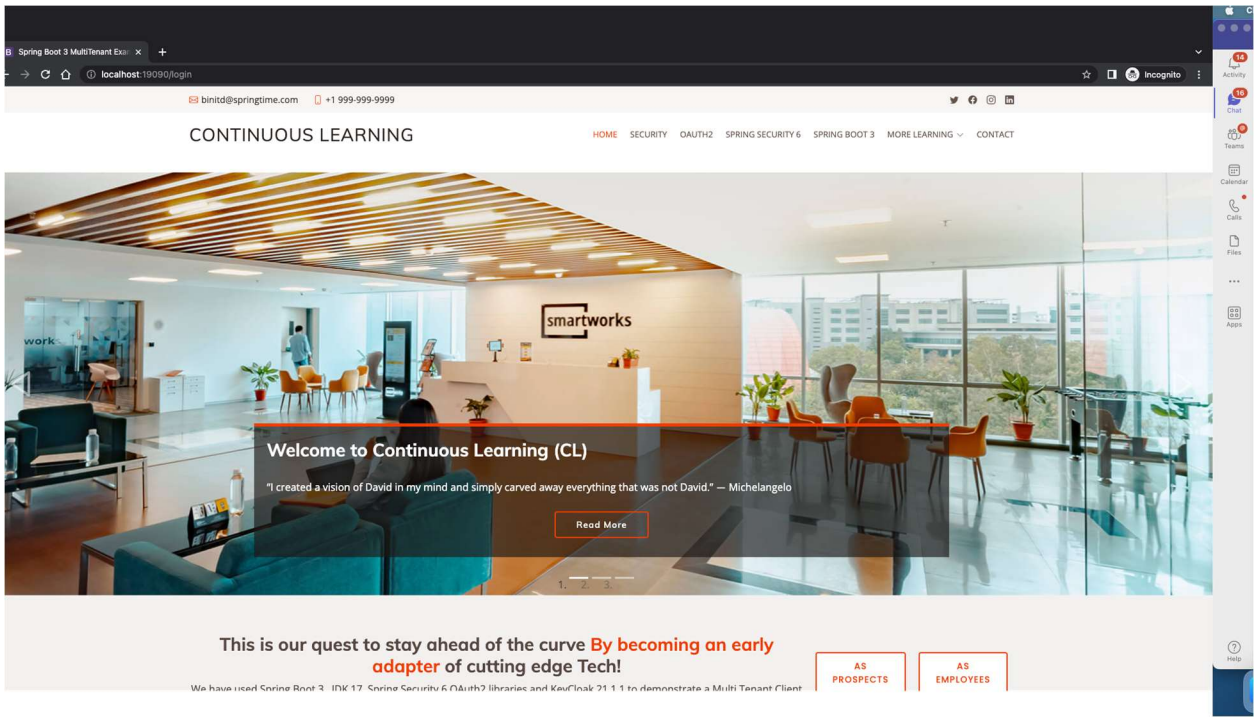


Figure 1-1 The Login Page of the Application

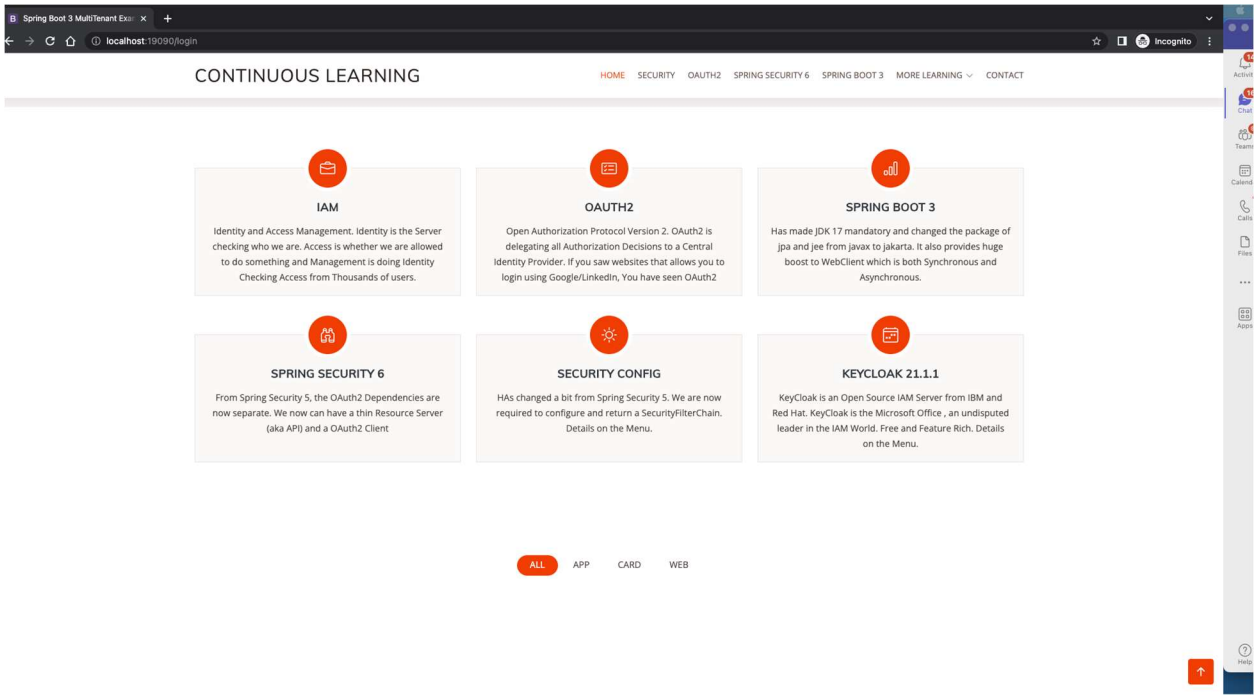


Figure 1-2 The Login Page of the Application Continued..

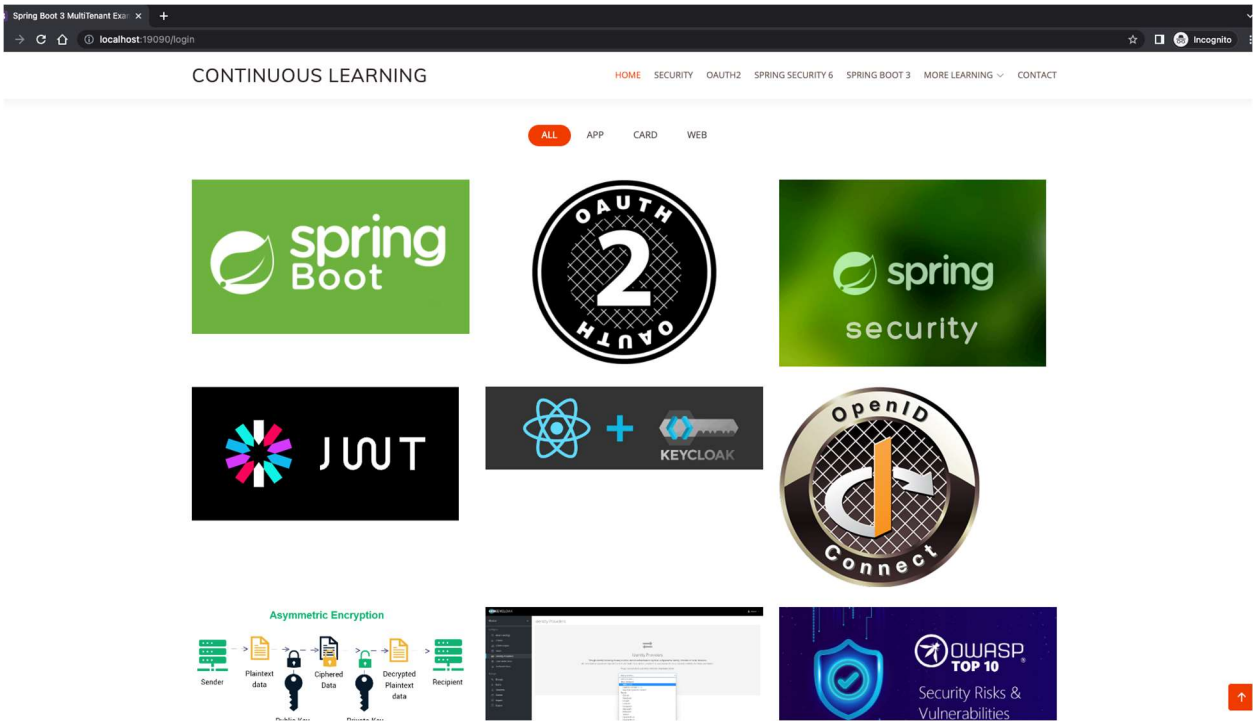


Figure 1-3 The Tech Documentation Section of the Login Page of the Application

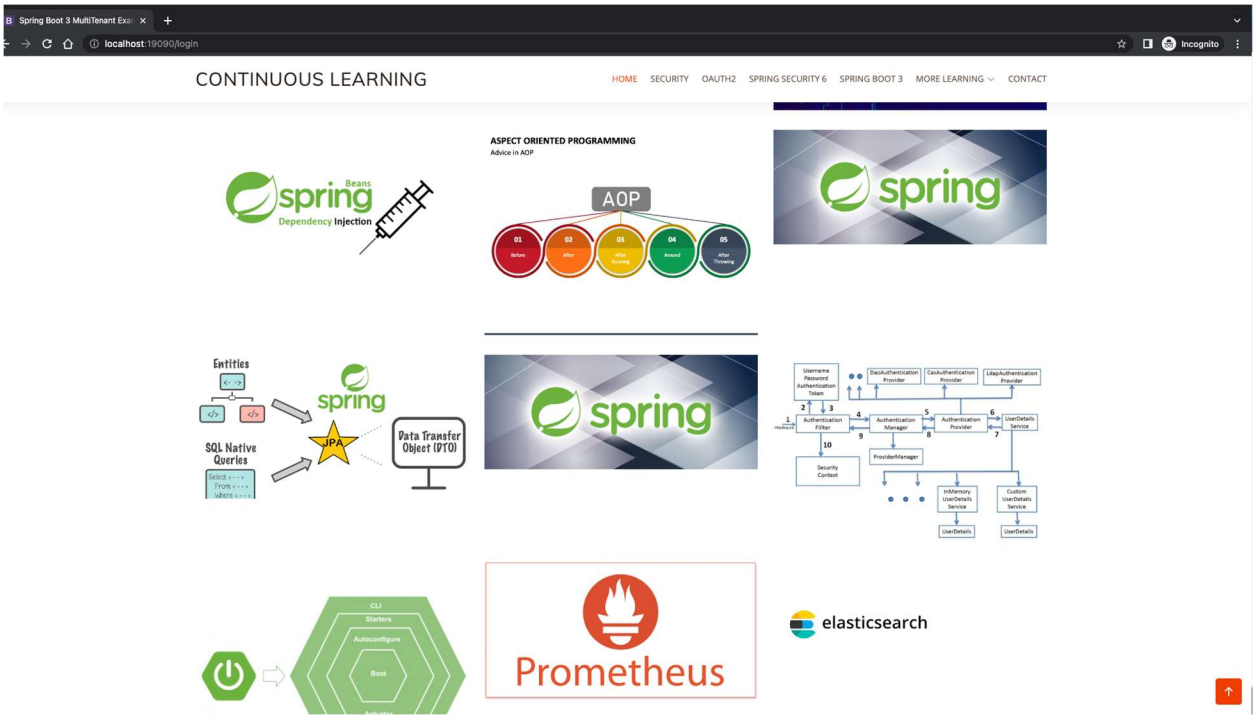


Figure 1-4 The Tech Documentation Section of the Login Page of the Application

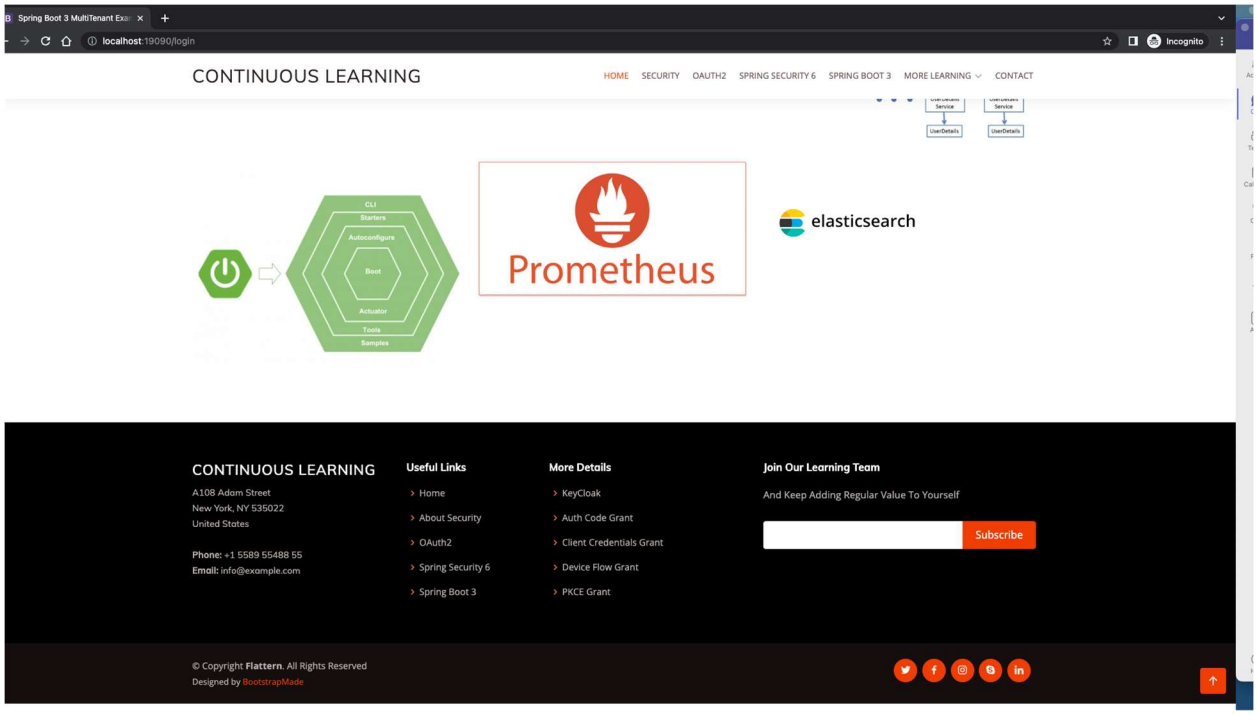


Figure 1-5 The Footer Section Of the  Login Page of the Application

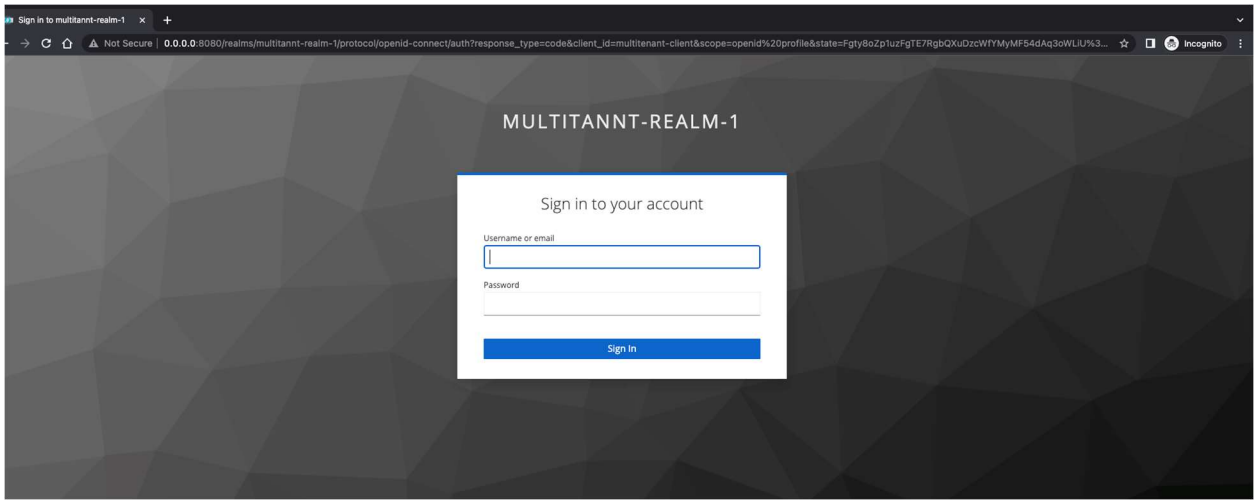


Figure 1-6 The KeyCloak Login Screen for the First Realm

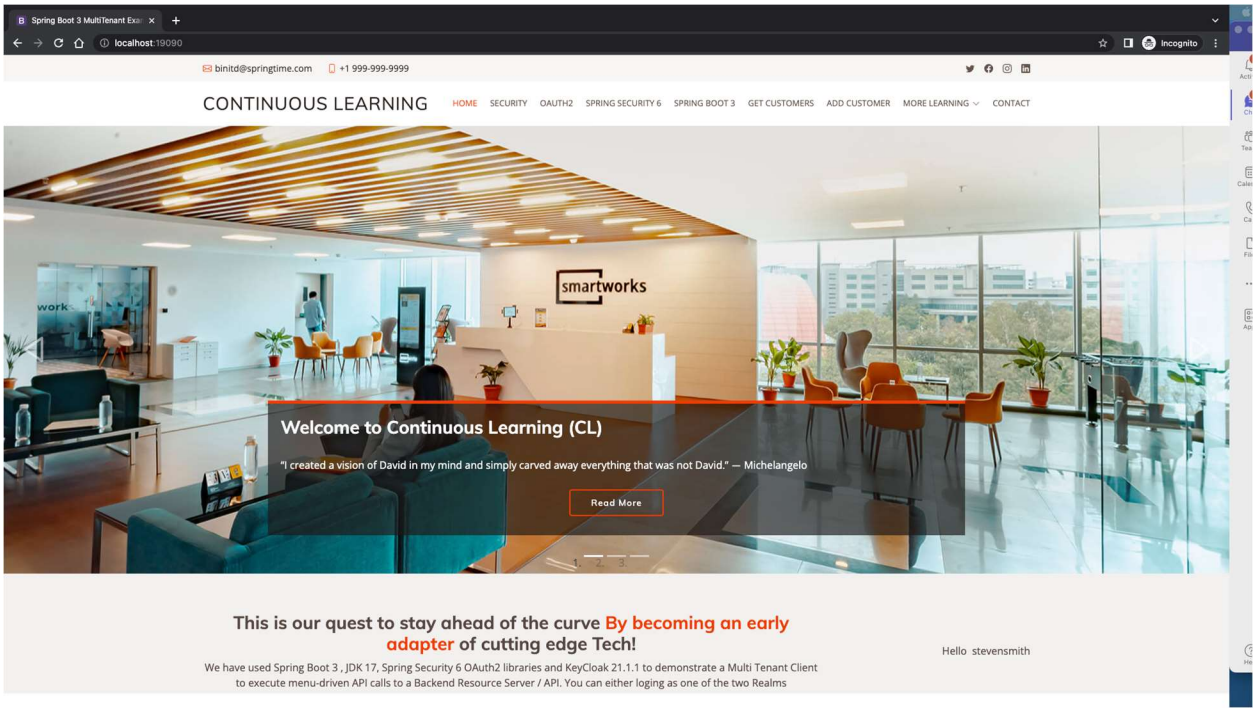


Figure 1-7 The Logged In Screen of the Application

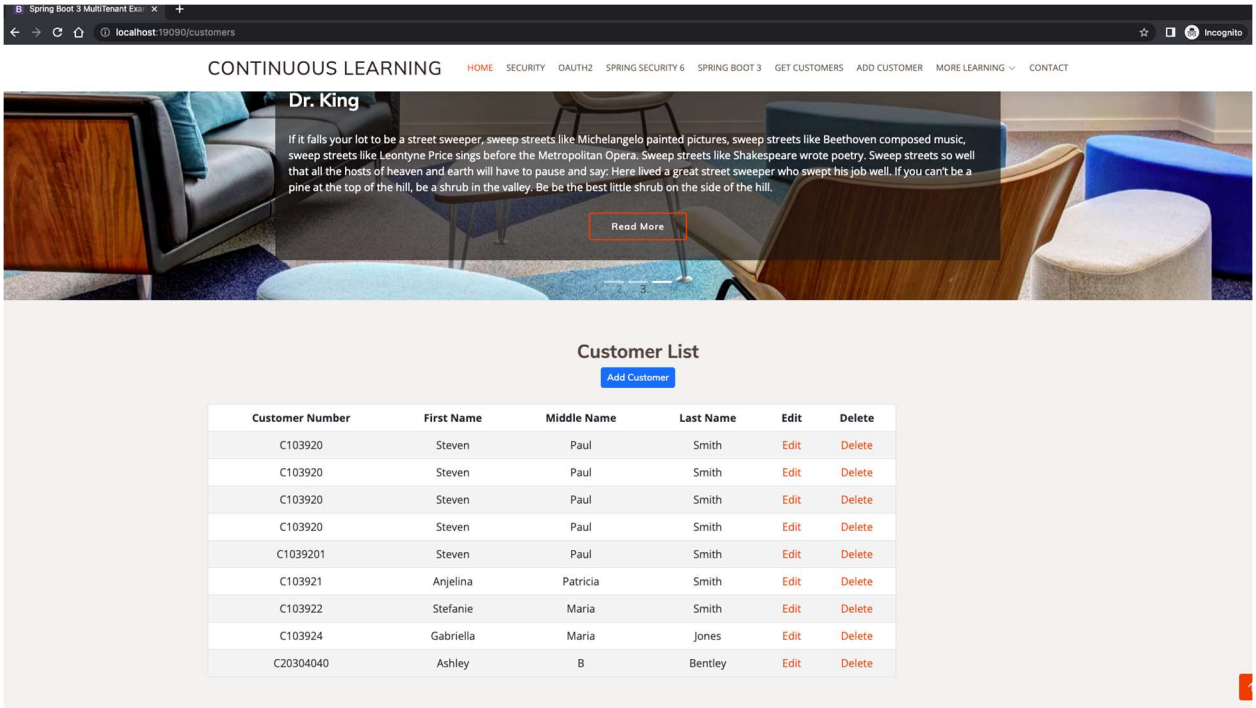


Figure 1-8 The View Customer Page

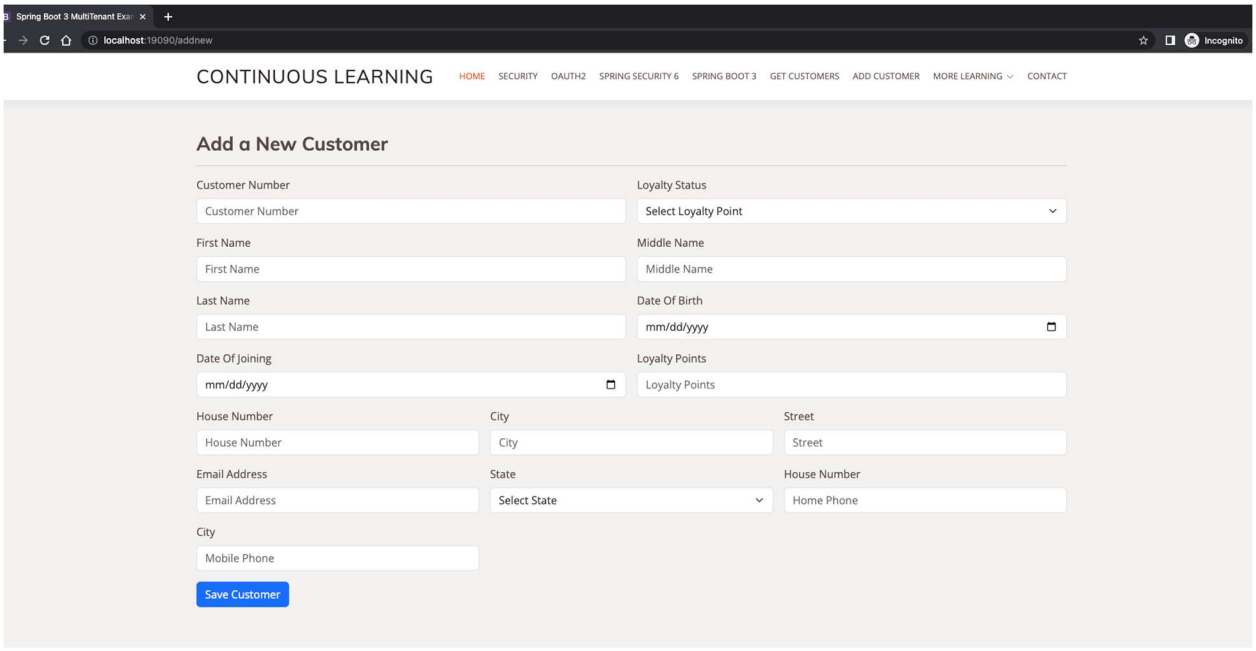


Figure 1-9 The Enter New Customer Page

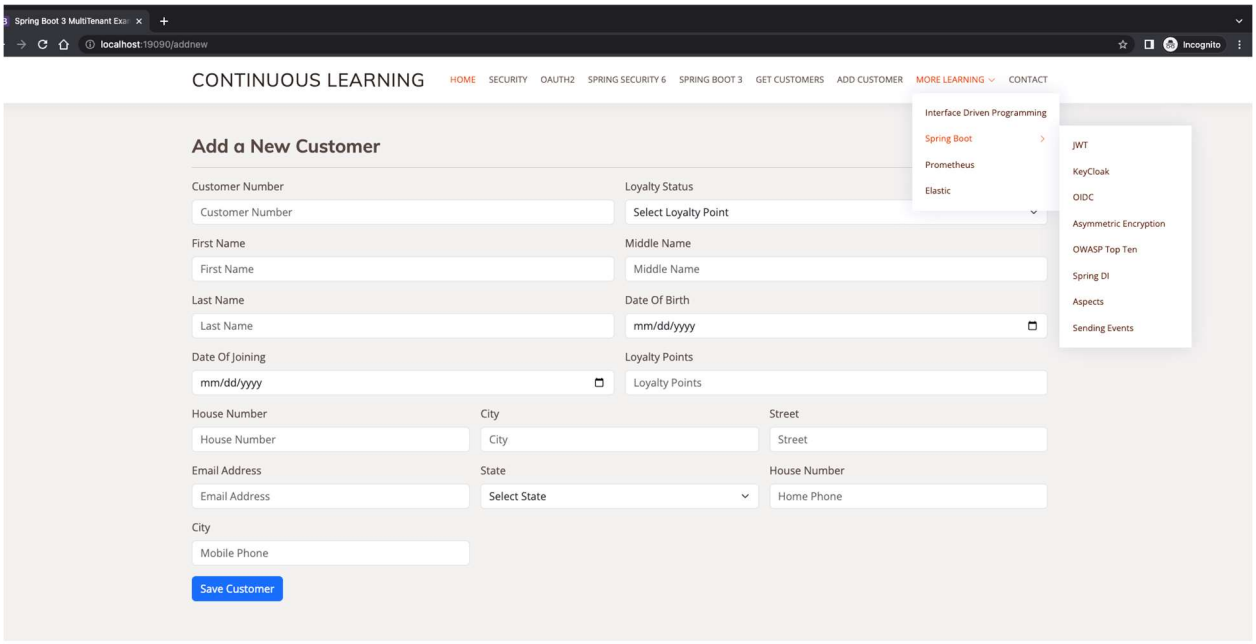


Figure 1-10 The Drop-Down Menu for Documentation Pages

## 1.2 Identity Access Management

The table below summarizes the three terms Identity, Access, and Management.

| Comparison | Normal Life | In Software Applications |
|---|---|---|
| **Meaning of Identity** | We want to know if the person is from Door Dash, USPS, A Family Member, A Friend and so on | REST APIs want to know if the caller is a trustworthy one |
| **Why Identity** | Because Physical Security is important | Because Data Security, Legal obligations, DDOS protection is important |
| **Meaning of Access** | Should the Food Delivery Delivery Representative have access to our living room or only at the main door | Should anyone and everyone have access to static assets such as JS CSS etc.? |
| **Why Access** | Should our family members have access to all the house | Should the warehouse shift supervisor access all tasks returned from an API? |
| | Should a Plumber installing a Water Heater have limited access in the house? | Should one contractor/employee access only his/her tasks? |
| **Meaning of Management** | Needs No Explanation | We have hundreds of applications, people accessing them, credentials, passwords, life cycle rotation, certificates and what not.... |

Figure 1-11 Summary Description of IAM

If we speak in the context of IT Applications, the following figure further classifies IAM.


Figure 1-12 IAM in A Nutshell

## 1.3 The challenge of Application Security

In the earliest days of Web Development, we started using HTTP, a protocol that was not developed with security in mind. HTTP was a plain text client server communication protocol that had nothing built in for securing the communication between the client browser and server. As a result, developers used shortcuts of sending the username and passwords along with the API calls they made. However, HTTP being plain text, anyone could intercept and read those username and password mid-flight.

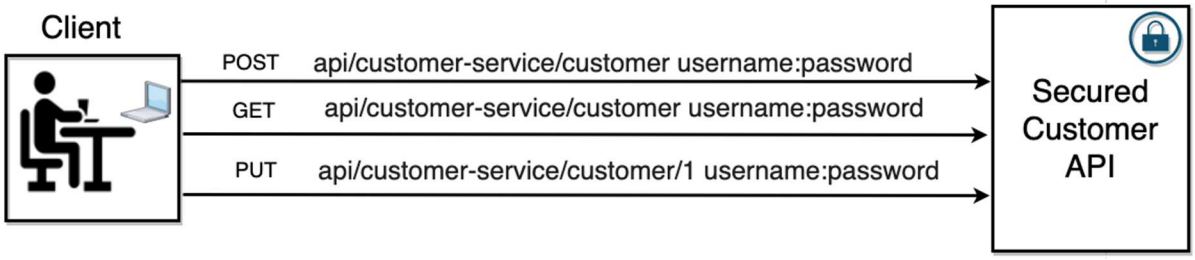


Figure 1-13 Sending UN/PW in Each and Every Request

While sending user credentials in plain text was a huge problem, each application / api having its own user database for authentication was another bigger one. If the user changed one password in one of the applications, he/she had to change the same in the other applications used. If one of the application databases got compromised , users had to change all their credentials across all the applications.

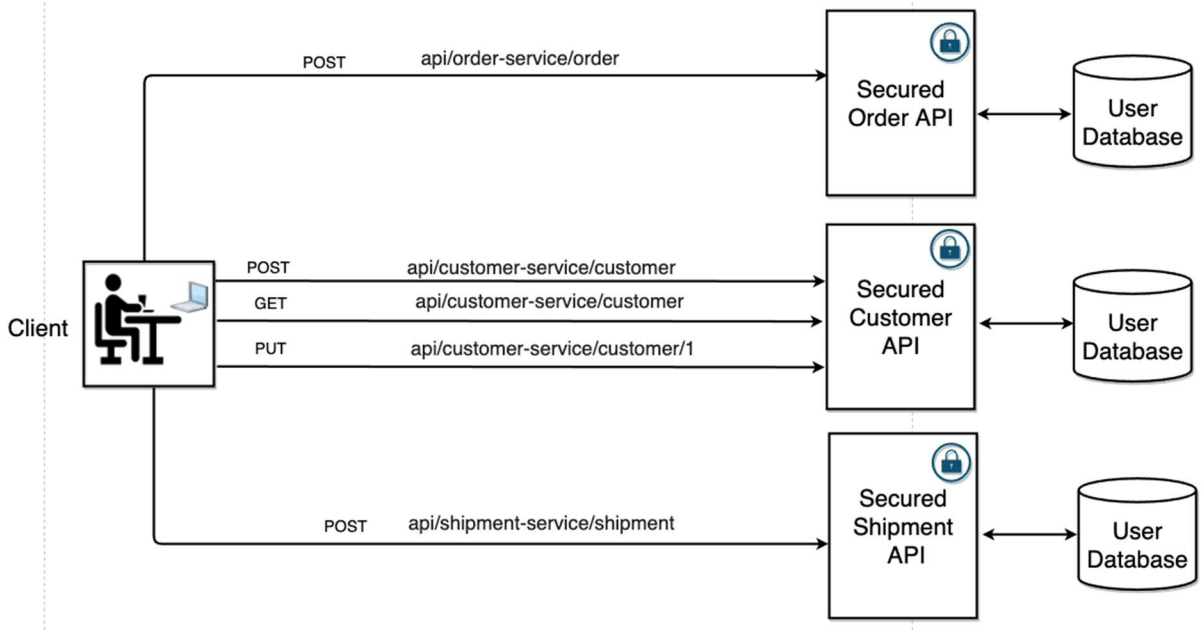


Figure 1-14 Each Application Having Its Own User DB

If we, however, look at the following diagram it should make sense to us. Why? Imagine an existing organization of any type. It could be a government organization, or it could be a private one as well. The organization has many departments like Sales, Marketing, Finance, Payroll, Production, Maintenance and so on. However, all the employees who work in these different departments get their access card from a central Security

department. Once an employee checks in through the main gate by swiping his/her card, he/she can have access to the rest of the building based on his/her access levels. The gates of the office buildings and floor doors are controlled by the same central security department shared by all other departments. This is the same setting humans are accustomed to since a very long time. Why would IT applications be different. With the following diagram, now IT applications can use the same Single Sign On (SSO) Setup with a single central security application (Authorization Server).

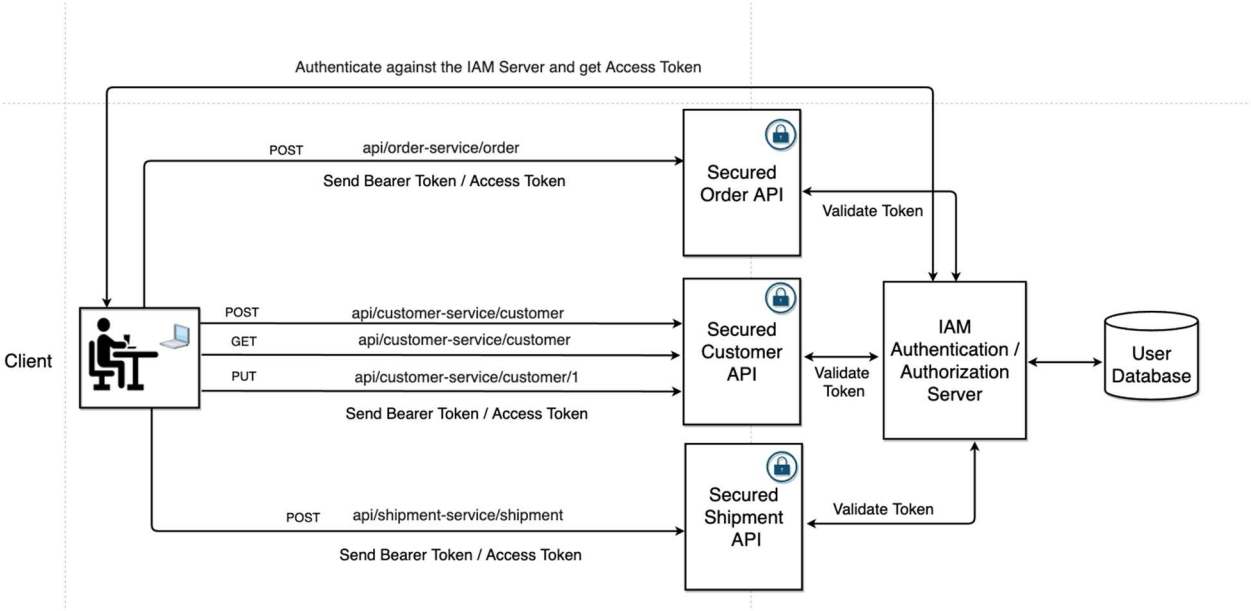


Figure 1-15 A Central Authentication and Authorization Application

## 1.4 Oauth2 and OIDC

We are living in a collaborative world. In this collaborative world, organizations work together sharing data in a safe and secure way. Whether we want a photo printing app to access our Apple/Google photos or the Mortgage company we have applied for a home loan needs to check our finances directly online from our banks using APIs.

Earlier (ten years back) it was common to have these client organizations (banks / photo app) ask for our credentials (username and passwords) and store them for accessing our resources from the Photo Server (Apple / Google) or Banks. This is bad for multiple security reasons.

- Our username and passwords are valid for 90 days or less.
- There are more than one organization storing our critical assets (username / passwords for bank accounts)
- Many people use the same credentials for multiple accounts and sharing one with another organization increases the vulnerability from multiple aspects.

- **It is not good for the organizations accepting and storing external users credentials for other external organizations (One Bank storing another bank's user account credentials and vice versa). A subsequent breach exposes these organizations to Federal and State Laws and penalties.**

Considering, the drawback of this legacy password driven system for organizations to call each other's APIs, several large Organizations (Google, FB, Microsoft, Oracle, Amazon, and others) have joined forces to design a more secure specification of ensuring both open access and tight security.

OAuth which stands for Open Authorization (for accessing / invoking APIs) using short lived Access Tokens has seen two versions V1 and V2. Only the V2 (OAuth2) is valid now.

**OAuth2 is only a specification : US Department of Transportation and Environmental Protection Agency (EPA) publishes specifications for Vehicle safety and emission standards that all manufactures must adhere to sell their vehicles in the US. Likewise all OAuth2 providers (Okta, Google, Amazon, Microsoft, IBM, Oracle, and others) provide implementations of this OAuth2 specification. This means companies using one OAuth2 provider should be able to migrate from their existing OAuth2 provider to another, like how we can replace Eclipse Link JPA provider with Hibernate, replace MySQL with Oracle using JDBC etc.**

While OAuth2 only authorizes, and issues access tokens (and refresh tokens to), people soon felt the need to also have access to the accessing user's information, names, addresses, phone numbers, email ids etc. Oauth2 by itself was only meant for generating access tokens which did not have user information in it. Thus, another specification called Open (O) Identify (ID) Connectivity (C) was created to satisfy that need. OIDC works on top of OAuth2 and provides a /user-info API to provide user profile information. So far we came across three tokens. Let us clarify what they are :

- Access Tokens : To get them from the Authorization Server and present them (like access cards in security gates) to the REST APIs (Resource Servers) to get data or create / update / delete data.
- Refresh Tokens : To keep getting access tokens without login in again. Access tokens are short lived like 5-15 minutes. Refresh tokens live a little longer like a day or so. We can call the Authorization Server to get a new Refresh Token so that we will continue to get access tokens without login in again. More on this later.
- ID Token : Generated by the Authorization Server to send user information to the clients following OIDC.