## OBJECTIVE

Use JPA to write a production quality RESTful web service application.

## REQUIREMENTS

### a. Functional Requirements

Create a RESTful with the following functionalities;

i. A service that retrieves a JSON array of all unique dates (ignoring time) in the table. The resulting JSON array needs to be sorted in ascending order.

ii. A service that retrieves a JSON array of all unique users for which there is a login record between the start and end date. Both parameters are optional, so there can be a start date, an end date, or both. The resulting JSON array needs to be sorted in ascending order.

iii. A service that retrieves a JSON object where the key is the user name and the value is the number of times a user logged on between the start and the end date. All parameters are optional. The values used for the attributes are used as filters, i.e. only the records should be counted for which the attribute values are equal to the ones specified in the parameters.

### b. Non-functional Requirements and Assumptions

i. Basing the given functional requirements, the intention is somewhat to audit every login activity of every authenticated users. A functionality that actually audits login activities is assumed.

ii. To facilitate login auditing, application security is assumed to be added.

iii. Retrieval of login audits are highly confidential. It is desirable to not let any user retrieves them, but only to specific group of users, such as users with administrative rights.

## SOLUTIONS

**a.** Use Java Persistence API, a Java EE standard for object-relational mapping, to manage object persistence into a relational database. Use Hibernate as a JPA provider.

**b.** Use Spring Data on top of Hibernate to rapidly implement data access objects.

**c.** Use Spring Framework to facilitate service implementation.

**d.** Use Spring Web MVC to facilitate exposure of services in a RESTful Web API fashion.

**e.** Use Spring Security with Spring Cloud OAuth2 for security, facilitating authentication and authorization.

**f.** Use Spring Boot and Spring Cloud for rapid application development, making the application cloud-ready by adhering microservices architecture.

## ARCHITECTURE

### a. Components

Wikipedia defines microservice architecture as a specialisation of and implementation approach for service-oriented architectures (SOA) used to build flexible, independently deployable

software systems. In microservices architecture, application is partitioned into individually deployable services. This is fitting if application components are desired to be simple, decoupled, pluggable, and scalable.

The diagram below shows the components or services for the application.

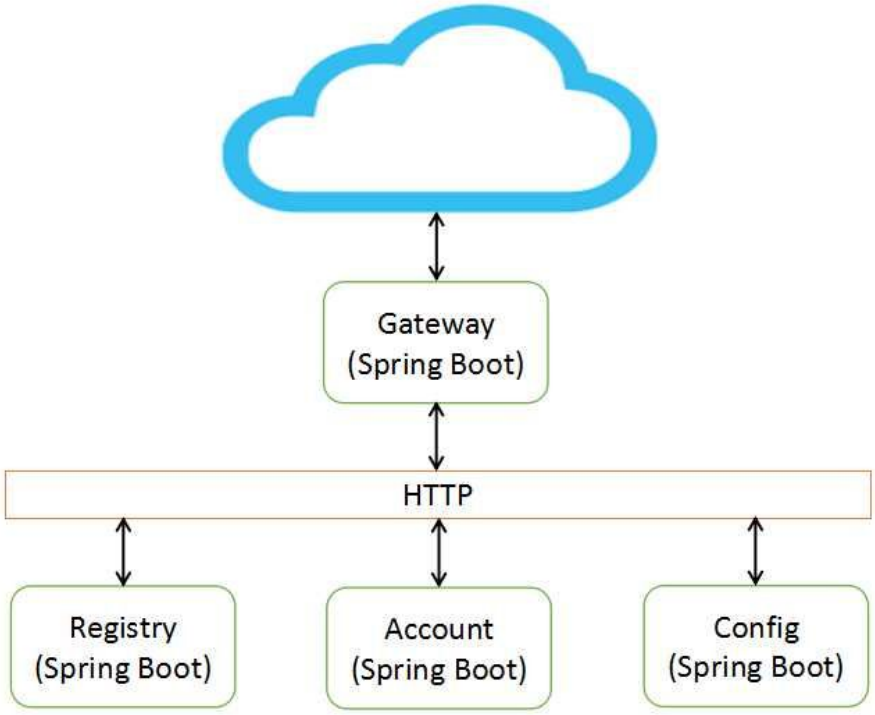


**Figure 1:** ***High level services of the application***

a. **Gateway**
   The gateway service defines how clients access the services in a microservices architecture. It provides a single entry point for clients, and external services to access the application.
   The application uses Spring Cloud Netflix Zuul to implement the gateway.

b. **Registry**
   Service discovery pattern is used to route requests for a client to an available service instance in a microservices architecture, where services and service instances can grow and scale dramatically. WIth this, service-to-service calls are eased with added benefit of load balancing.
   The application leverages Spring Cloud Netflix Eureka to implement service discovery. Eureka is a client-side service discovery implementation, where services register themselves to the central service registry, and query for location of other services.

c. **Config**
   The application uses Spring Cloud Config as a central configuration server to all services. With this, services can be reconfigured anytime based on environment, profile, or demand, without shutdowns.

d. **Account**
   The account service is the only business service momentarily.

With this architecture, the application is prepared for additional services to cater new requirements. Services are also ready to scale.

### b. Class Diagram

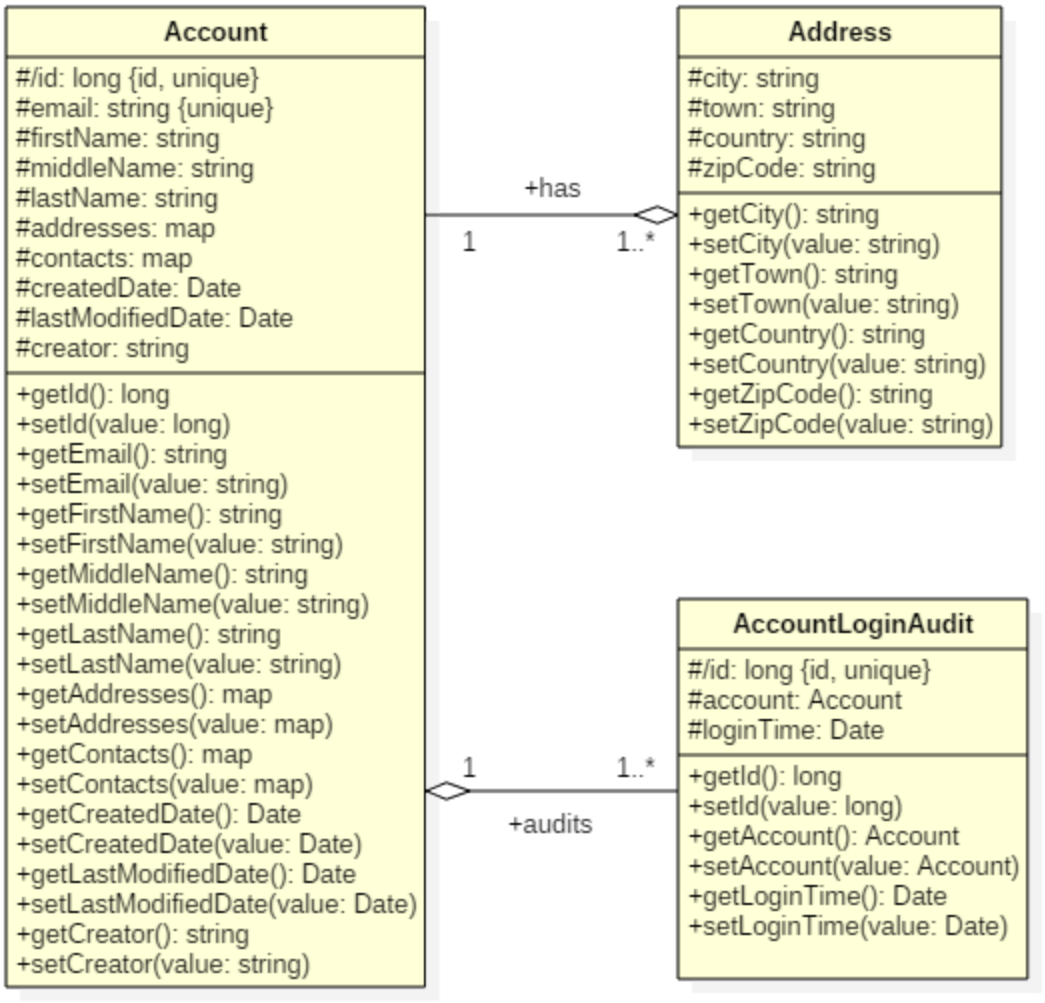


**Figure 2:** ***Account business model***

## OUTPUT

### a. Project Repository

https://github.com/warrenmnocos/application
https://github.com/warrenmnocos/application-config

### b. Deployment

http://104.199.173.188/account/api/rest/account

### c. Security Token

All endpoints are secured with Spring Security and Spring Cloud OAuth2. OAuth2 is an open standard for authorizing clients to access resources. The following endpoint shows how to get an access token, which follows the resource owner password credentials grant type.

| Endpoint | Purpose | Method | Parameters |
|---|---|---|---|
| http://104.199.173.188/account/oauth/token | Retrieves access token | **POST** | HTTP Basic Credentials (Required)<br>a. the `username` is the `client_id`<br>b. The `password` is the `client_secret`<br>POST data (Required)<br>a. `grant_type`: string, value must be 'password'<br>b. `username`: string (i.e. warrenwevicknocos@gmail.com)<br>c. `password`: string (i.e. 1234) |

Example to get access token:

```
curl -XPOST
d56h2sf5c5drsdgr4xpl234cm085021r:of7ho3vc86t00hd6bg7qrwpl12h754pl@104
.199.173.188/account/oauth/token -d grant_type=password -d
username=warrenwevicknocos@gmail.com -d password=1234
```

### d. Endpoints

The following endpoints exposes several business service operations.

| Endpoint | Purpose | Method | Parameters | Rights |
|---|---|---|---|---|
| http://104.199.173.188/account/api/rest/account | Retrieves all accounts | **GET** | Authorization (Required)<br>a. `access_token`: String (i.e. access_token=xxxxxxx), or Authorization header of bearer type (i.e. Authorization: Bearer xxxxxxx)<br>Pagination (Optional)<br>a. `page`: Integer (i.e. page=0)<br>b. `size`: Integer (i.e. size=10) | **ADMIN** |
| http://104.199.173.188/account/api/rest/acc | Retrieves an account by its | **GET** | Authorization (Required)<br>a. access_token: String (i.e. | **ADMIN** |

| | | | | |
|---|---|---|---|---|
| ount | unique identifier | | access_token=xxxxxxx), or Authorization header of bearer type (i.e. Authorization: Bearer xxxxxxx)<br>Logical Parameter (Required)<br>a. `id`: Long (i.e. id=3) | |
| http://104.199.173.188/account/api/rest/account | Retrieves an account by its email | **GET** | Authorization (Required)<br>a. `access_token`: String (i.e. access_token=xxxxxxx), or Authorization header of bearer type (i.e. Authorization: Bearer xxxxxxx)<br>Logical Parameter (Required)<br>a. `email`: String (i.e. email=war@gmail.com) | **ADMIN** |
| http://104.199.173.188/account/api/rest/account/me | Retrieves currently authenticated user | **GET** | Authorization (Required)<br>a. `access_token`: String (i.e. access_token=xxxxxxx), or Authorization header of bearer type (i.e. Authorization: Bearer xxxxxxx) | **ADMIN, USER** |
| http://104.199.173.188/account/api/rest/account/access/dates | Retrieves all unique dates (ignoring time) with login activity | **GET** | Authorization (Required)<br>a. `access_token`: String (i.e. access_token=xxxxxxx), or Authorization header of bearer type (i.e. Authorization: Bearer xxxxxxx)<br>Pagination (Optional)<br>a. `page`: Integer (i.e. page=0)<br>b. `size`: Integer (i.e. size=10) | **ADMIN** |
| http://104.199.173.188/account/api/rest/account/access/users | Retrieves all unique users for which there is a login record between the start and end date | **GET** | Authorization (Required)<br>a. `access_token`: String (i.e. access_token=xxxxxxx), or Authorization header of bearer type (i.e. Authorization: Bearer xxxxxxx)<br>Pagination (Optional)<br>a. `page`: Integer (i.e. page=0)<br>b. `size`: Integer (i.e. size=10)<br>Logical Parameter (Optional)<br>a. `start`: String with format yyyymmdd (i.e. 20161001)<br>b. `endt`: String with format yyyymmdd (i.e. 20161001) | **ADMIN** |
| http://104.199.173.188/account/api/rest/account/access/logins | Retrieves a JSON object where the key is the user name and the | **GET** | Authorization (Required)<br>a. `access_token`: String (i.e. access_token=xxxxxxx), or Authorization header of bearer type (i.e. | **ADMIN** |

| | | | | |
|---|---|---|---|---|
| | value is the number of times a user logged on between the start and the end date. Results can also be filtered using email, first name, middle name, and last name. | | Authorization: Bearer xxxxxxx)<br>Pagination (Optional)<br>a. `page`: Integer (i.e. page=0)<br>b. `size`: Integer (i.e. size=10)<br>Logical Parameter (Optional)<br>a. `start`: String with format yyyymmdd (i.e. 20161001)<br>b. `endt`: String with format yyyymmdd (i.e. 20161001)<br>c. `email`: String (i.e. email=xxx@xxx.xxx) repeatable (i.e. email=xxx@xxx.xxx&email=yyy@yyy.yyy or email=xxx@xxx.xxx,yyy@yyy.yyy)<br>d. `firstName`: String (i.e. firstName=Lou) repeatable (i.e. firstName=Lou&firstName=Rica or firstName=Lou,Rica)<br>e. `middleName`: String (i.e. middleName=Lou) repeatable (i.e. middleName=Lou&middleName=Rica or middleName=Lou,Rica)<br>f. `lastName`: String (i.e. firstName=Lou) repeatable (i.e. lastName=Lou&lastName=Rica or lastName=Lou,Rica) | |

Example to access service endpoint:

```
http://104.199.173.188/account/api/rest/account/access/logins?access_token
=9eae0f89-d547-4a9b-8568-ed04736616bc&firstName=Lou&firstName=Rica
```

## LIMITATIONS

a. The application is not served in HTTPS.
b. Cannot add, and update user accounts.
c. Cannot add client accounts (OAuth2).

## REFERENCES


a. https://en.wikipedia.org/wiki/Microservices
b. http://microservices.io/