

DESIGN OF OUR API

What API will be used and how



St10036710

OPSC7312

PART 1

Contents

[What API will be used and why? 2](#_Toc174959337)

[The 6 REST design principles 2](#_Toc174959338)

[How we will be using the API to design our application 4](#_Toc174959339)

[Creation and Hosting of the API 5](#_Toc174959340)

[UML Diagrams for our application 6](#_Toc174959341)

# References

IBM. (2024, 08 18). *What is a REST API*. Retrieved from ibm.com: https://www.ibm.com/topics/rest-apis

# What API will be used and why?

The API we will be using is a REST API in other terms RESTful API. The RESTful API is a set of rules that need to be followed for building and integrating with web services. This API allows different software applications to communicate with each other over the internet by using standard HTTP methods such as the GET, POST, PUT and DELETE methods.

# The 6 REST design principles

I will be using these principles to create our application and use them to my advantage:

* **Uniform interface**
* All API requests for the same resource should look the same, no matter where the request comes from. The REST API should ensure that the same piece of data, such as the name or email address of a user, belongs to only one uniform resource identifier (URI). Resources shouldn’t be too large but should contain every piece of information that the client might need. (IBM, 2024)
* **Client-server decoupling**
* In REST API design, client and server applications must be completely independent of each other. The only information that the client application should know is the URI of the requested resource; it can't interact with the server application in any other ways. Similarly, a server application shouldn't modify the client application other than passing it to the requested data via HTTP. (IBM, 2024)
* **Statelessness**
* REST APIs are stateless, meaning that each request needs to include all the information necessary for processing it. In other words, REST APIs do not require any server-side sessions. Server applications aren’t allowed to store any data related to a client request. (IBM, 2024)
* **Catcheability**
* When possible, resources should be cacheable on the client or server side. Server responses also need to contain information about whether caching is allowed for the delivered resource. The goal is to improve performance on the client side, while increasing scalability on the server side. (IBM, 2024)
* **Layered system architecture**
* In REST APIs, the calls and responses go through different layers. As a rule of thumb, don’t assume that the client, and server applications connect directly to each other. There may be a number of different intermediaries in the communication loop. REST APIs need to be designed so that neither the client nor the server can tell whether it communicates with the end application or an intermediary. (IBM, 2024)
* **Code on demand (optional)**
* REST APIs usually send static resources, but in certain cases, responses can also contain executable code (such as Java applets). In these cases, the code should only run on-demand. (IBM, 2024)

# How we will be using the API to design our application

The API will manage key operations such as inventory management, stock procurement, supplier management, and invoicing. Here's an overview of how we will use the API endpoints and their purposes:

* **User Management:**
  + POST /register: Register a new user (business owner, supplier).
  + POST /login: Authenticate a user and return a token.
  + GET /profile: Retrieve the profile details of the logged-in user.
* **Inventory Management:**
  + GET /inventory: Retrieve all items in the inventory.
  + POST /inventory: Add a new item to the inventory.
  + PUT /inventory/{itemId}: Update an existing item's details (e.g., quantity, description).
  + DELETE /inventory/{itemId}: Remove an item from the inventory.
* **Supplier Management:**
  + GET: Retrieve a list of all suppliers.
  + POST: Add a new supplier.
  + PUT: Update details of an existing supplier.
  + DELETE: Remove a supplier.
  + Supplier details such as supplier\_id, name, contact\_info, inventory\_items.
  + List of suppliers with details like supplier\_id, name, contact\_info, inventory\_items.
* **Stock Procurement:**
  + POST /procurement: Procure stock from a supplier.
  + GET /procurement/{itemId}: Retrieve procurement details for a specific item.
* **Supplier Management:**
  + GET /suppliers: Retrieve a list of registered suppliers.
  + GET /suppliers/{supplierId}/inventory: View the inventory of a specific supplier.
  + POST /suppliers: Register a new supplier.
* **Warehouse Management:**
  + POST /warehouses: Add a new warehouse.
  + GET /warehouses: View all warehouses.
  + PUT /warehouses/{warehouseId}/items: Move items to a warehouse.
* **Invoicing:**
  + POST /invoices: Generate an invoice for a customer.
  + GET /invoices/{invoiceId}: Retrieve invoice details.
* **Profit & Loss:**
  + GET /profit-loss: Calculate and retrieve profit vs. loss data.
* **Data Sent and Received**
  + User Data: Usernames, emails, passwords (hashed), roles (admin, supplier).
  + Inventory Data: Item names, quantities, prices, descriptions, supplier info.
  + Procurement Data: Item IDs, quantities, supplier IDs, procurement dates.
  + Invoice Data: Customer details, itemized lists, totals, tax information.
  + Warehouse Data: Warehouse locations, items stored, capacities.

# Creation and Hosting of the API

**Development**:

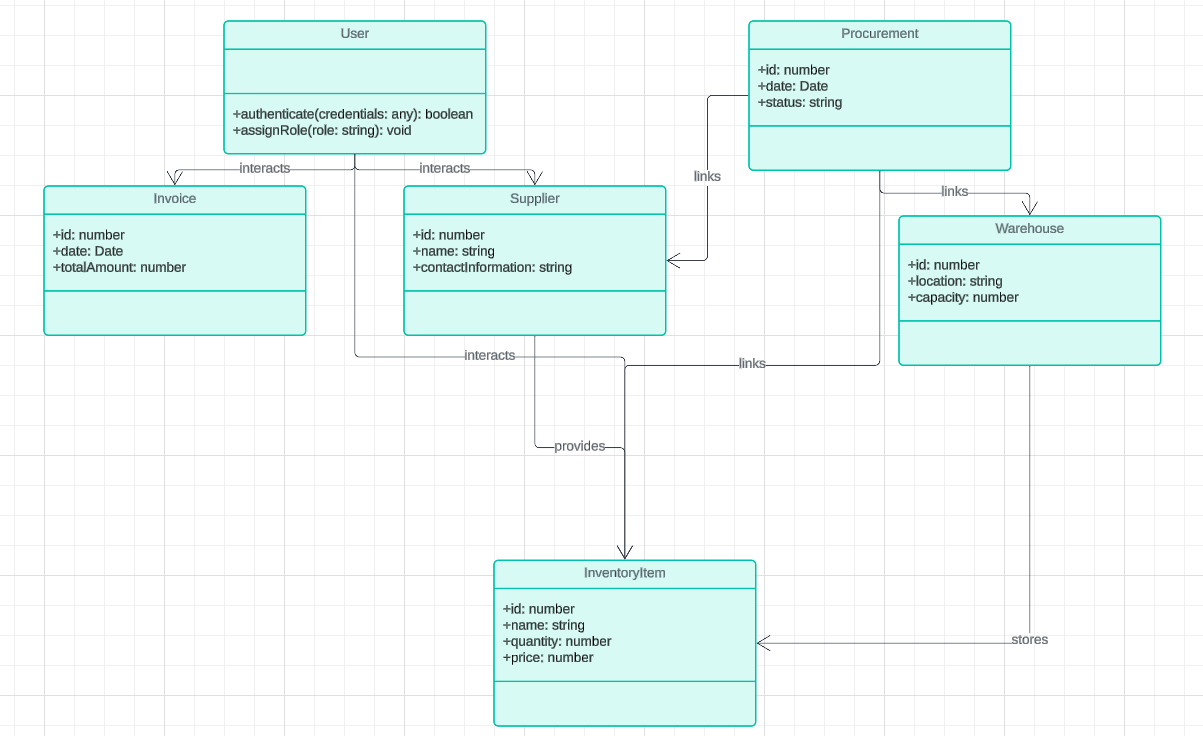
* The API will be developed using Node.js with Express.js framework in Visual Studio Code, which provides a robust environment for server-side development.
* Firebase can be used for the backend database and as a real-time communication service for the Android app. Firebase provides seamless integration with Android Studio.
* Android Studio will be used to develop the Android client, which will consume the API.

**Hosting**:

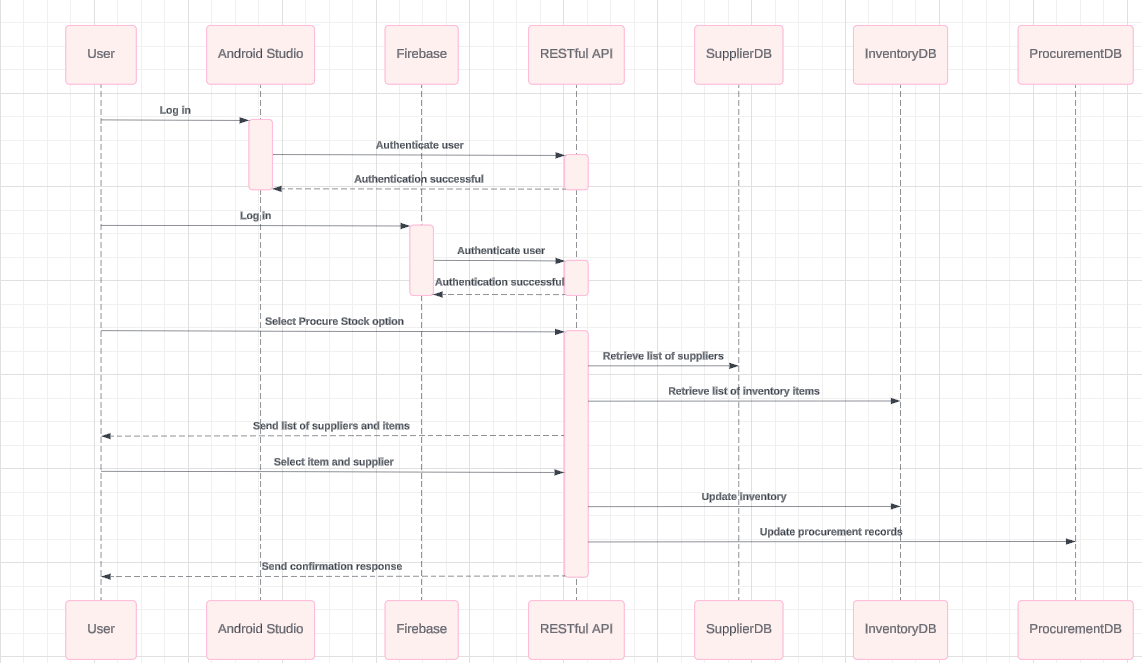
* The API can be hosted on **Firebase** or **Google Cloud Platform (GCP)**, which offers seamless integration with Android Studio. Firebase Hosting is particularly well-suited for deploying static and dynamic web content, and it provides SSL certificates and custom domain support.
* Heroku can also be used for hosting if Firebase is not selected, as it is easy to set up and supports Node.js applications out of the box.

# UML Diagrams for our application

* Class Diagram



* Sequence Diagram



* Deployment Diagram

