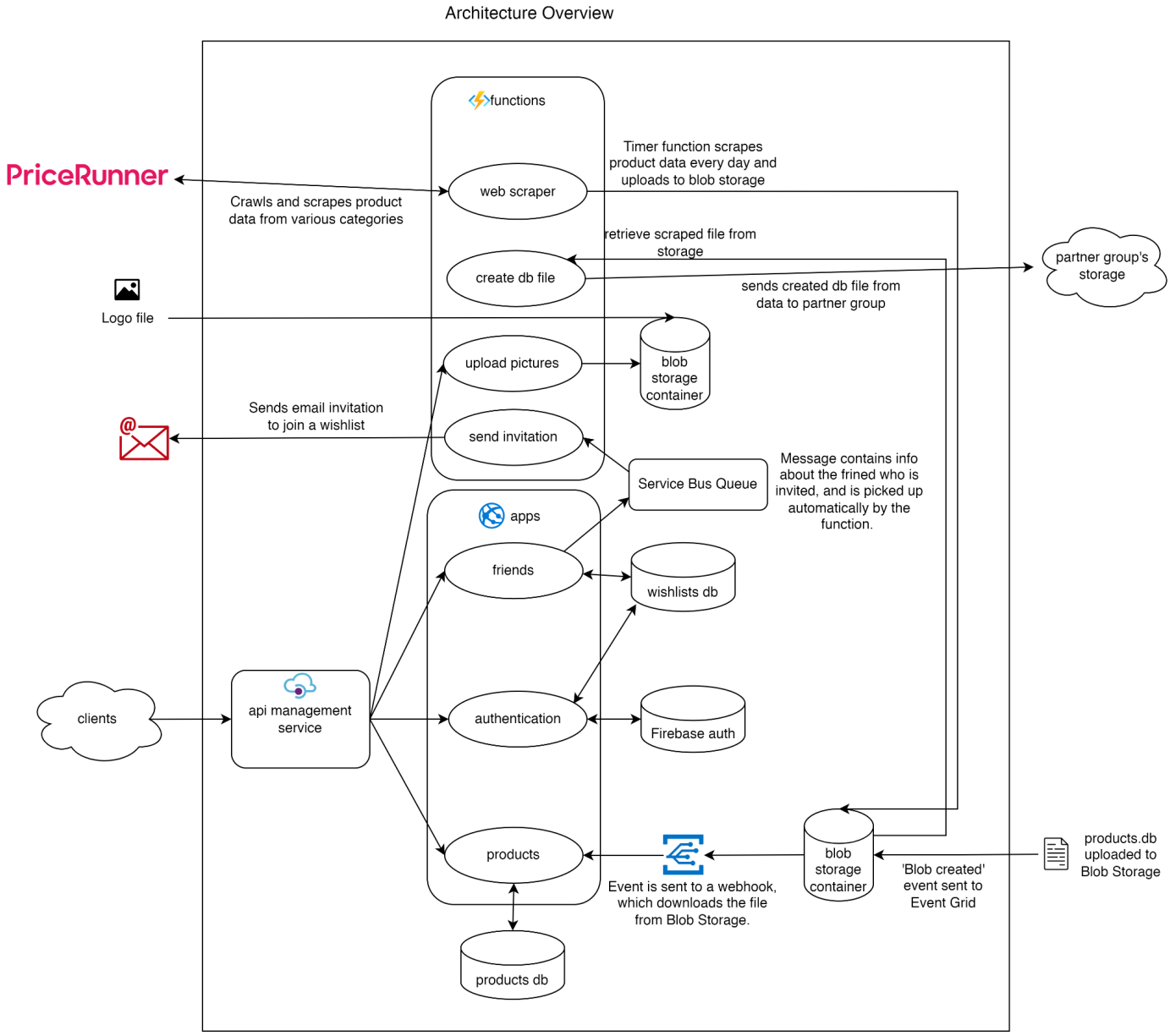
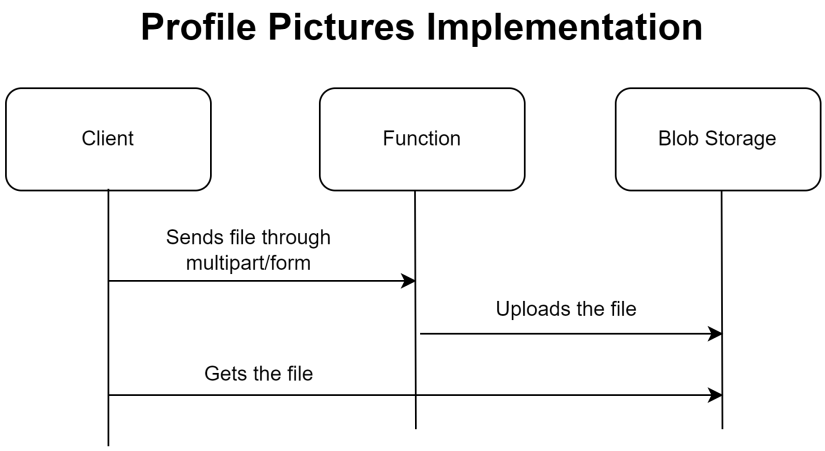
Overview of the system

System Architecture Diagram



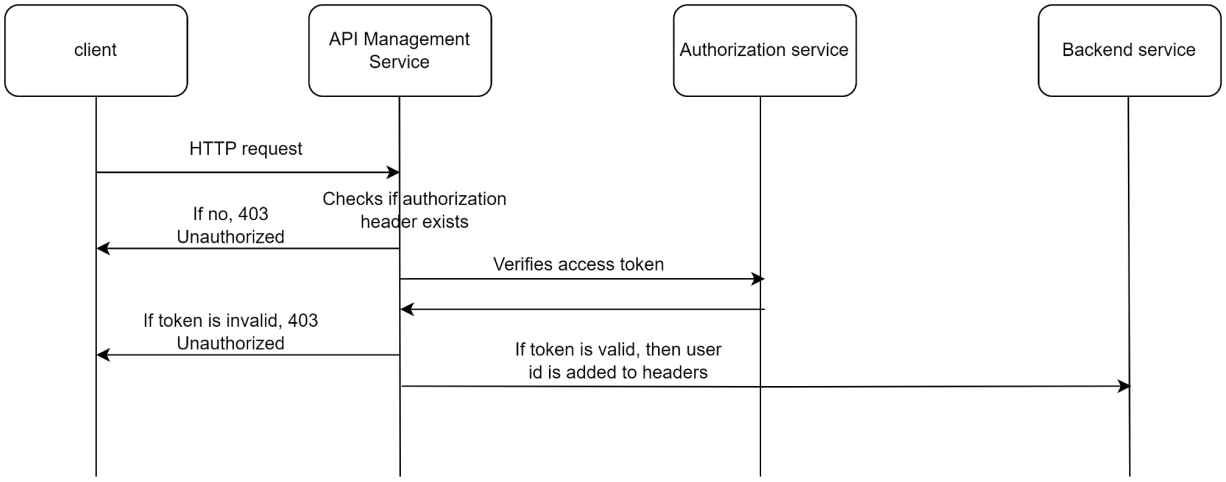
System Sequence Diagrams

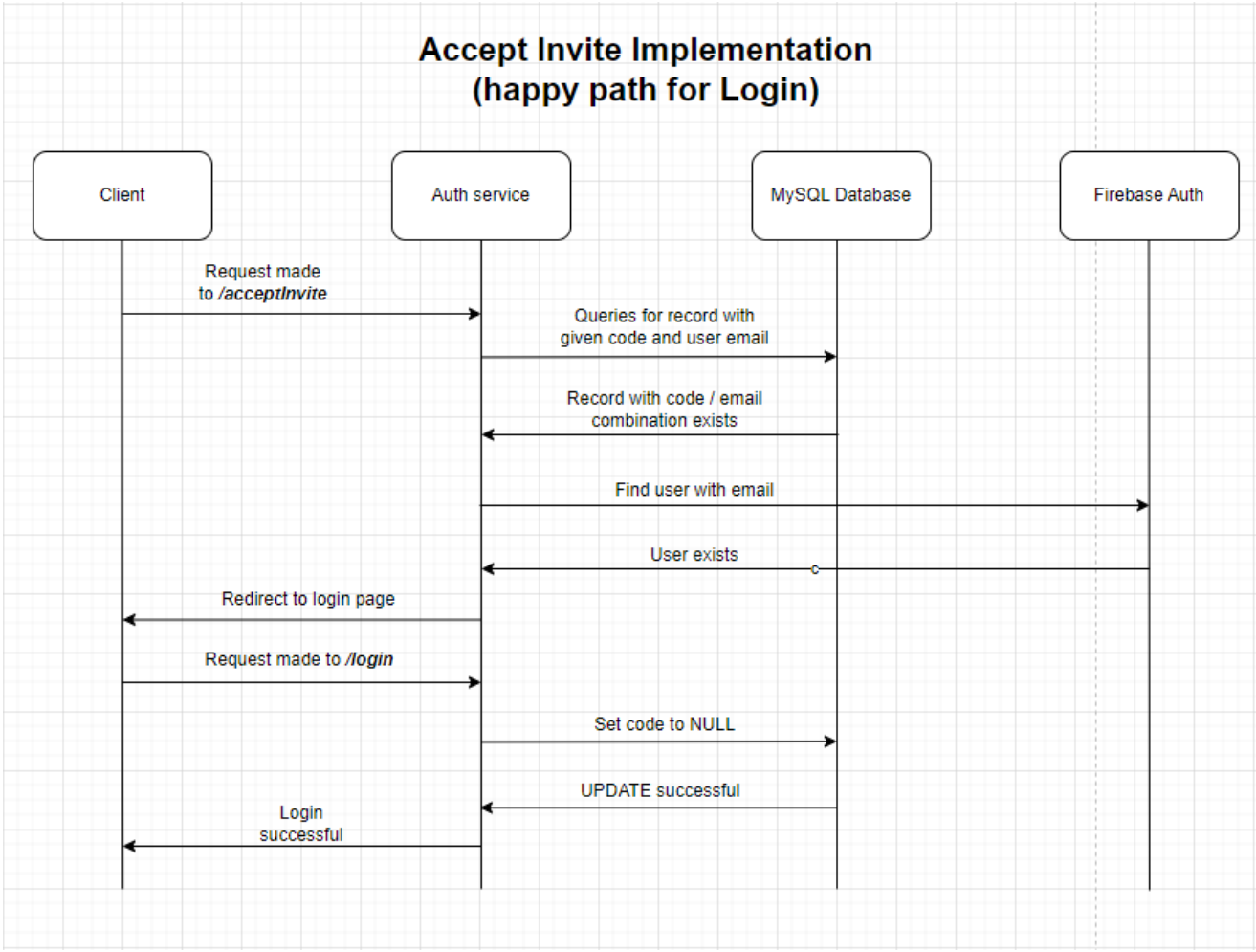


Advantages of the design:
- Decouples storage from the client.
- Does not expose any API key to the client, hence it is secure.

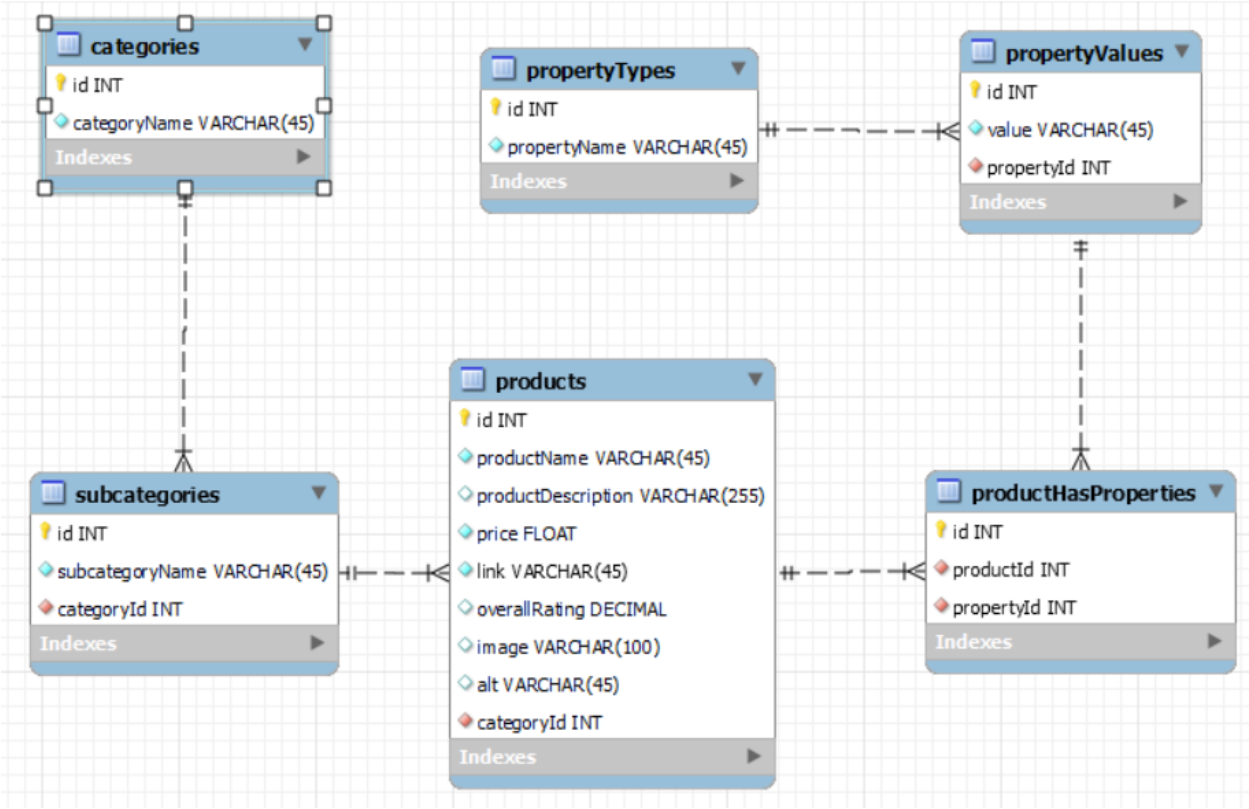
API Management Service and Authentication

We used the API Management Service (APIMS) to unify our backend services and route requests. Furthermore, APIMS can cache the authorization bearer token, therefore making the system processing requests faster. We added an inbound policy rule, so APIMS will check if incoming requests contain authorization header with valid token. If yes, then it will forward the request to the backend service. If not, it will make a request to our own Authorization service to refresh the token and save it to the cache. The following sequence diagram illustrates the flow.




Accept Invite Implementation
(happy path for Login)
Client
Auth service
MySQL Database
Firebase Auth
Request made to /acceptInvite
Queries for record with given code and user email
Record with code / email combination exists
Find user with email
User exists
Redirect to login page
Request made to /login
Set code to NULL
UPDATE successful
Login successful

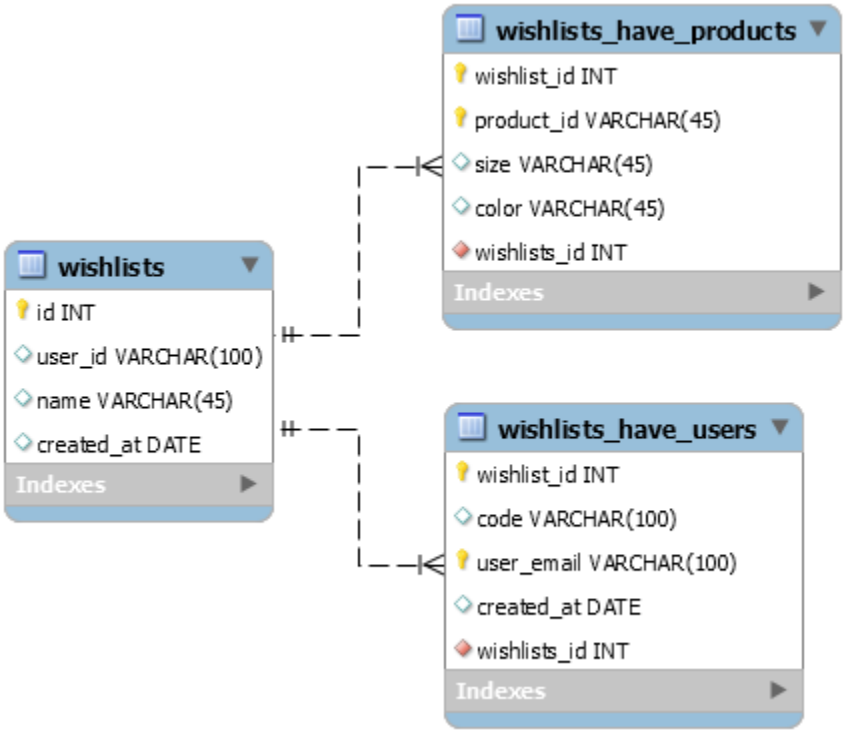
Products database schema



Our products database has 6 tables and revolves around the main products table. Each product has several unique properties that were found on the pricerunner.dk stored in the products table. Additionally, there were multiple properties that were the same for multiple products, therefore they have been split into multiple tables to reduce duplication.
Each product has its own category, and each category has its own subcategories. This was represented by the categories and subcategories tables. Products table only references subcategories, because each subcategory belongs to a specific category, and the information can be retrieved by joins.

Each product can also have different properties with different options. While doing web scraping we have found that these properties are size and color, however we haven't restricted our database to these in case more options will be available in the future. This type is stored in the propertyTypes table. Each one can have multiple values, which are stored in the propertyValues table, each referencing which propertyType it belongs to by id. At the end, the value is joined with the product with a many to many relationship using joining table productHasProperties, because a single product can have many different values assigned to it.

Wishlists database schema



Our wishlists database schema has three tables and is the main database for the “friends” backend service. It has three tables and is used for storing data about wishlists being created by the users, along with associations with products and invited users. The wishlists_have_users table helps us know when a user has been invited to join a wishlist and has not yet accepted the invitation (new record with user email and a code), and when a user has been invited and has accepted the invitation (record with user email but the code is set to NULL).

Who worked on what: (just add your names to whatever you can talk about during exam)

1. Expose
    a. **The "nice logo, bro" path:** George
    b. **The data path:** Dimitris, Marianna
    c. **The product search path:** George
    d. **The friend path:** Dimitris, Marianna
    e. **The authentication path:** David
    f. **The profile picture path:** Dimitris, Marianna

2. Integrate
    a. **The website path:** Group
    b. **The "nice logo, bro" path:** George
    c. **The data path:** Marianna
    d. **The product search path:** George
    e. **The friend path:** Dimitris, Marianna
    f. **The authentication path:** David
    g. **The profile picture path:** Dimitris, Marianna