Large-Scale and Multi-Structured Databases

Project Design BarberShop

Andrea Bedini
Edoardo Geraci
Alessandro Versari

Application Highlights

Book and review barbers near you

- View a list of barbers near you
- View a barber shop page with an Appointment calendar and book one slot for yourself
- Review a barber shop
- Upvote or Downvote a review

Handle your barber shop

- See the current pending appointment list
- Set your shop description and employees availability
- Get insightful analytics on your shop

Data Sources

Data used in the Backend is obtained through 2 API sources:

- Google Maps/Places
- Yelp

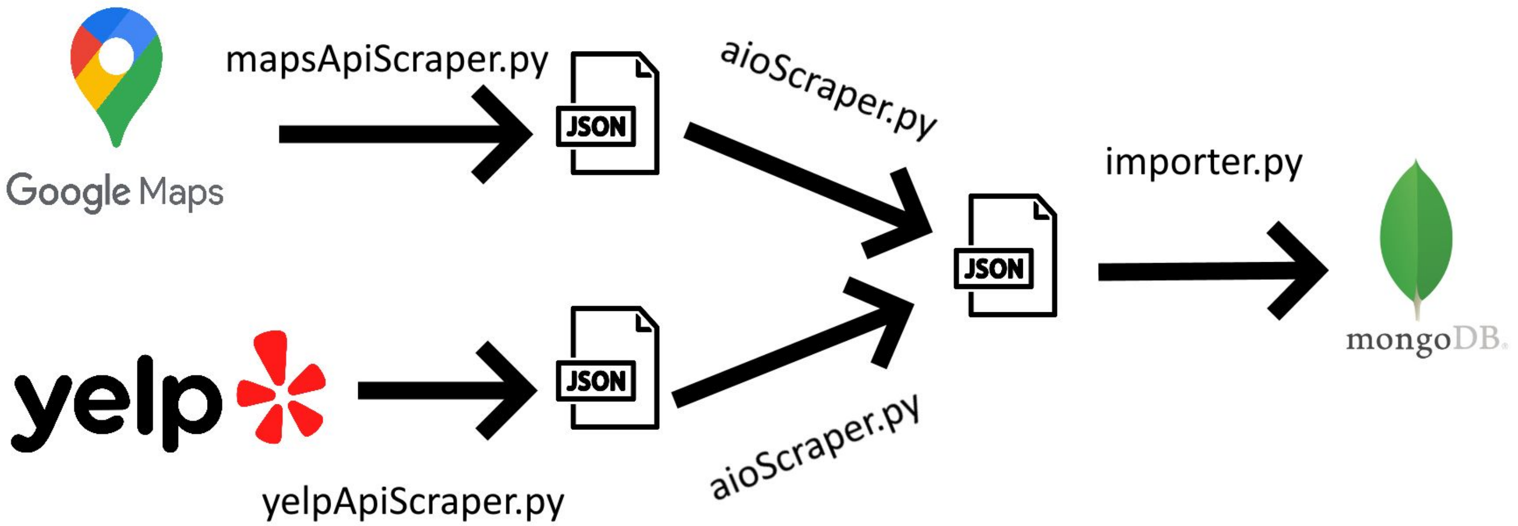
From these sources, the following data is obtained:

- Usernames
- Barbershops' data
- Reviews

All the other data in the Database is produced by Faker, a library that produces believable data. This is data that cannot be known/scraped, such as a user login details, or data that is too application specific, such as ShopViews and Appointments.

All data is obtained, processed, and then imported via the usage of Python scripts.

Data Sources Diagram



Functional and non functional requirements in a nutshell

Functional:

Any type of user

- account related operation
  - signup
  - login
  - password recovery
- find shops
- view shop profile
  - review shop
  - up/downvote review

User

- view profile info
  - view current appointment
  - delete current appointment
- delete account
- view shop profile
  - book appointment

Barber

- view profile info
- delete account
- browse owned shops
  - view booked appointments
  - delete appointments
  - view shop analytics

Admin

- browse users
- find user
- view user
  - delete user
  - edit barber shop ownership
- browse shops
  - edit shop information
- create shops
- view app analytics

Functional and non functional requirements in a nutshell

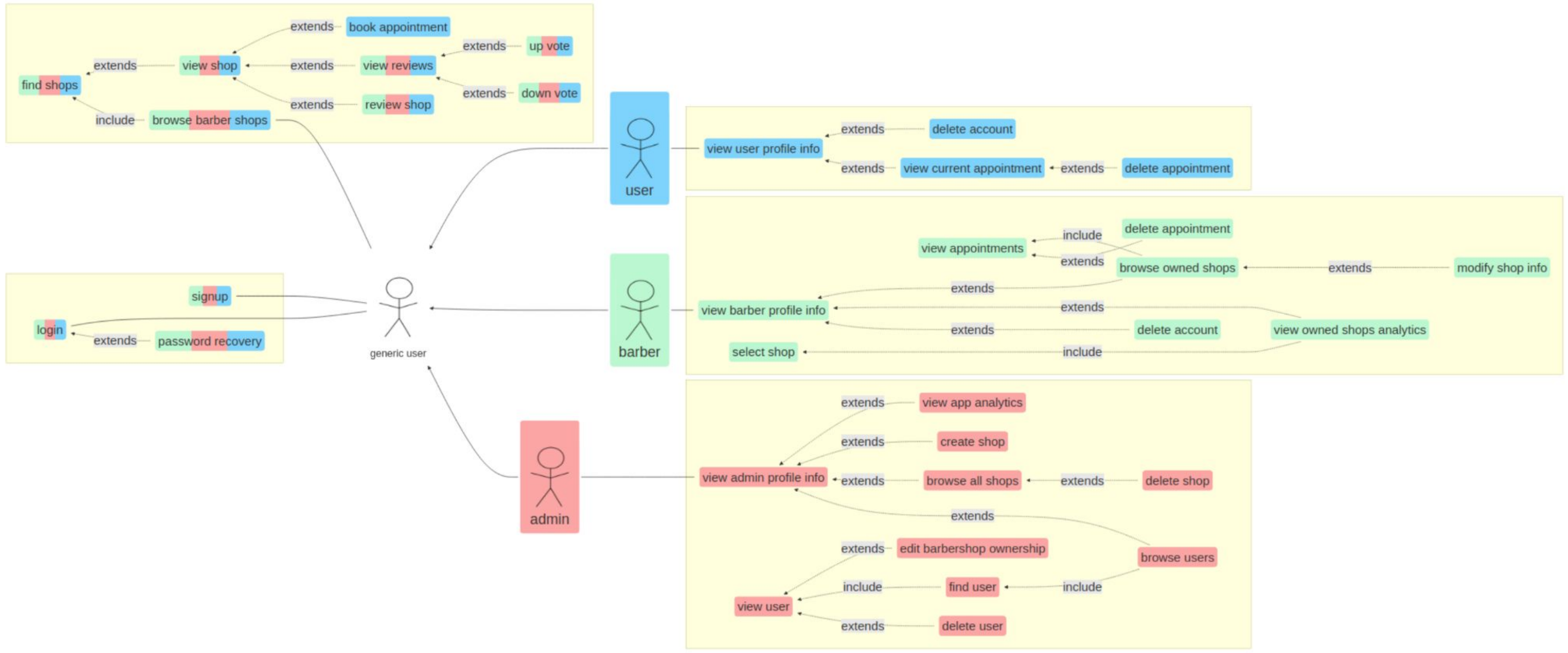
Non functional:

- Retrieve the list of the **nearest barber shops** in a **reasonable amount of time**
- Ensure **availability** of the data and **partition tolerance**
- **Cache** the appointment **calendar** of every barbershop to ensure **access in a short amount of time**
- Allow the user to search for places with a “**human-readable**” address/location
- Make meaningful **analytics** that might help identify issues in a shop’s management
- Separate data in different collections in order to **minimize** both **data loaded** from the database and **data sent** over the network
- Produce an **easy and simple to use UI** in order to allow every type of user to easily use the application

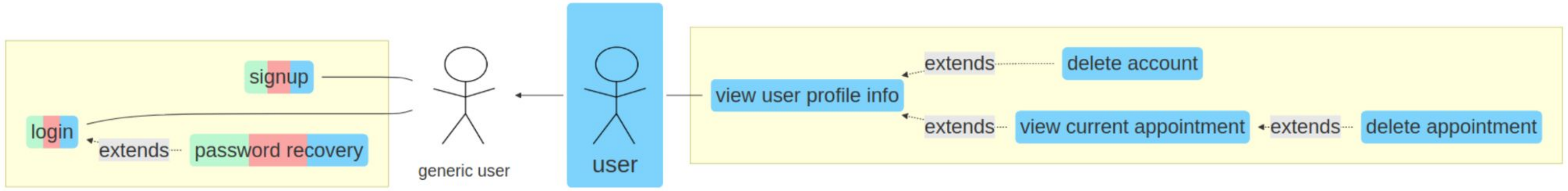
CAP Theorem Issue

- To ensure optimal performance for the expected influx of read operations, it is imperative to prioritize both high availability and low latency during the application's design;
- This application's design should prioritize **Availability (A)** and **Partition Tolerance (P)** over Consistency (C);

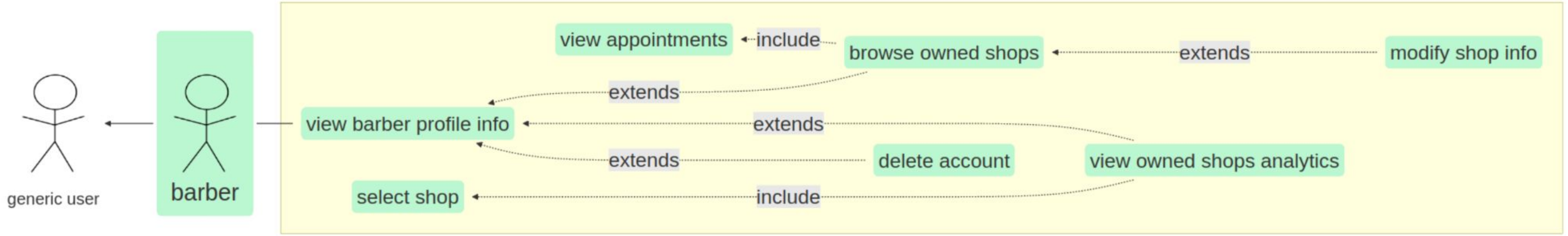
Actors and main supported functionalities



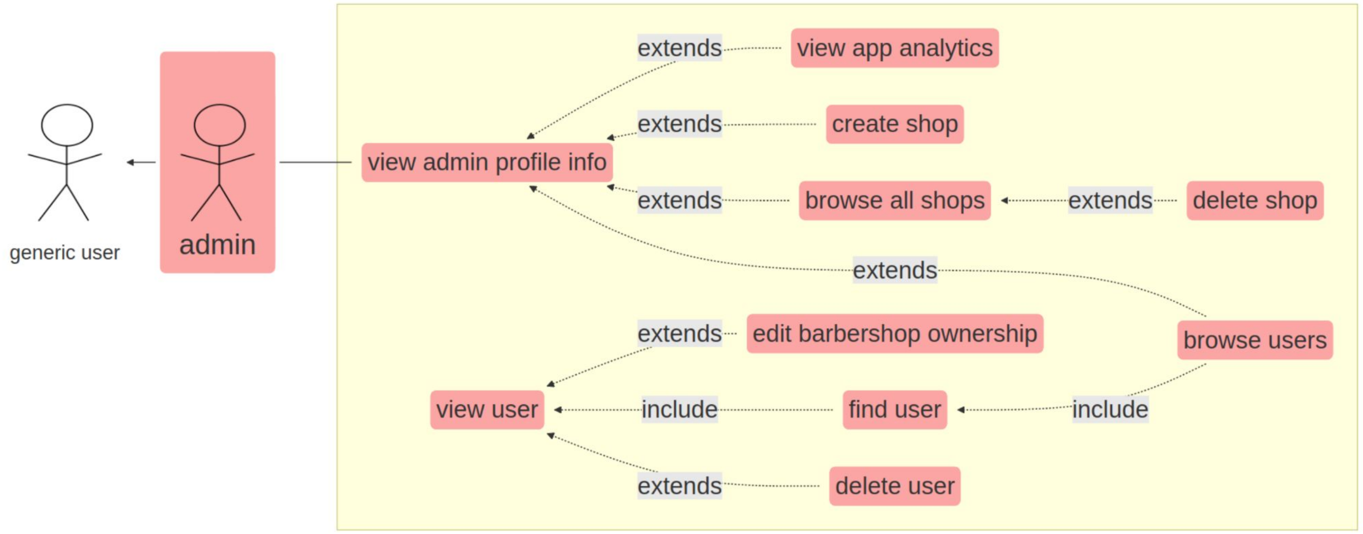
Actors and main supported functionalities



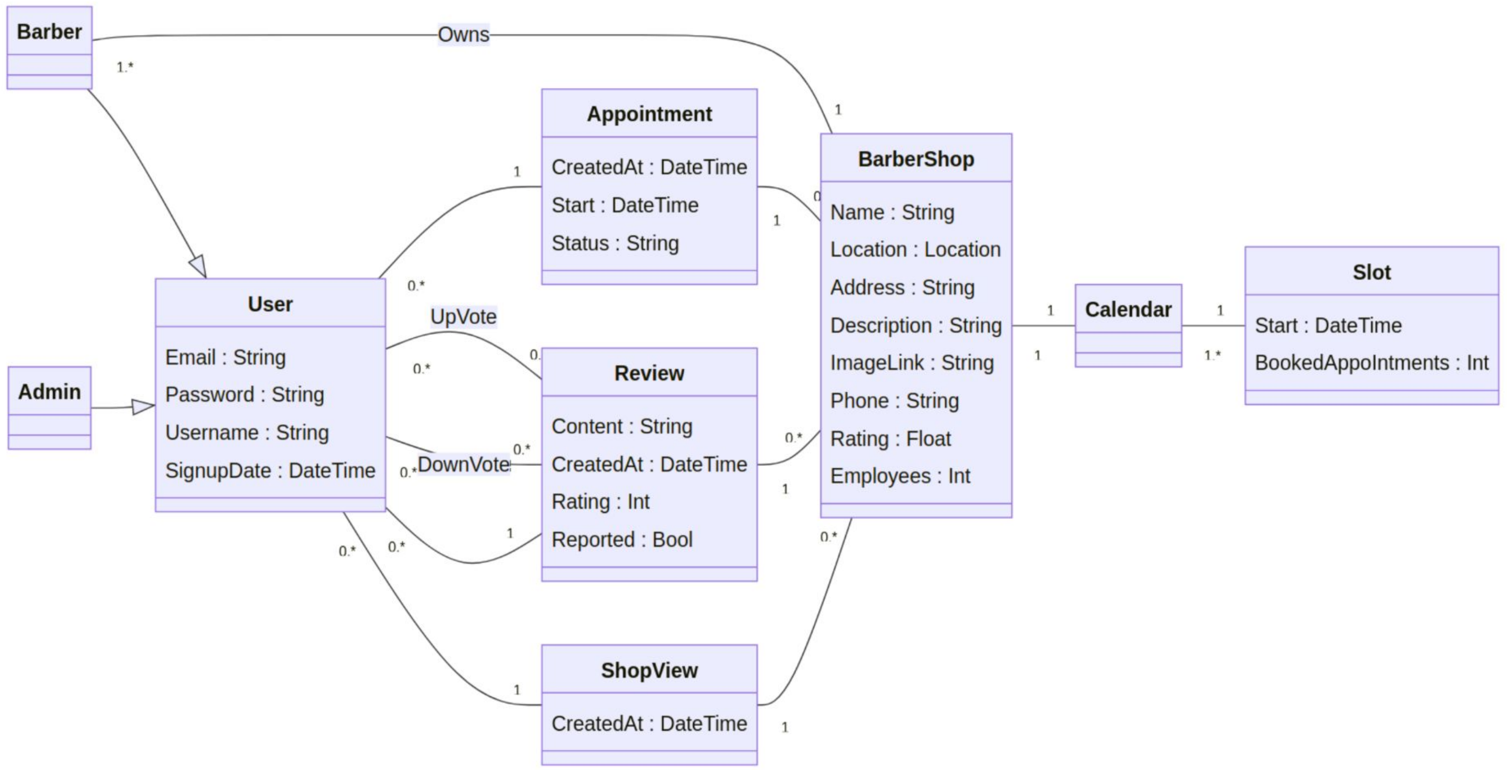
Actors and main supported functionalities



Actors and main supported functionalities



UML Class Diagram



Data Model

MongoDB:

MongoDB uses 5 different collections in order to **reduce** the amount of **unneeded fetched data**, and to better apply the necessary aggregations:

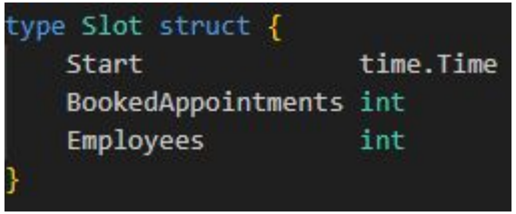
- Users
- Barbershops
- Shopviews
- Reviews
- Appointments

Data Model

Redis:

Redis is used as a cache for the **Appointments**. In order to correctly display the **Slots** in the **Appointments'** calendar, it is necessary to have stored some data that can be quickly accessed. Each Redis entry is set to expire a day after the **Datetime** associated with the **Slot**, as it is no longer useful.

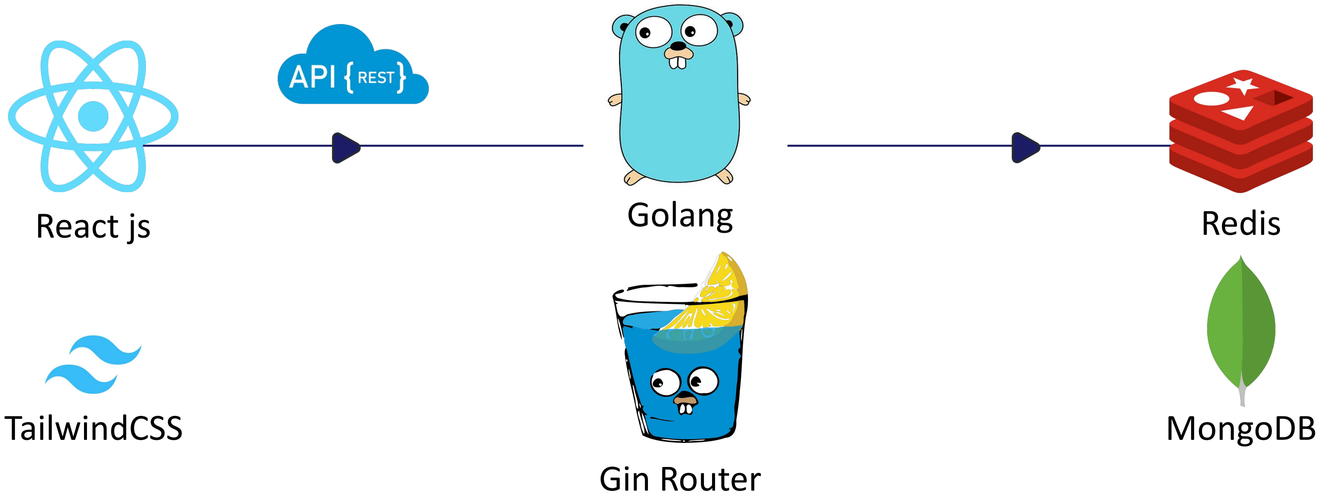
Each Redis key has the format "**barbershop:<shopID>:slots:<time>**", and it refers to a small document each with the following structure:



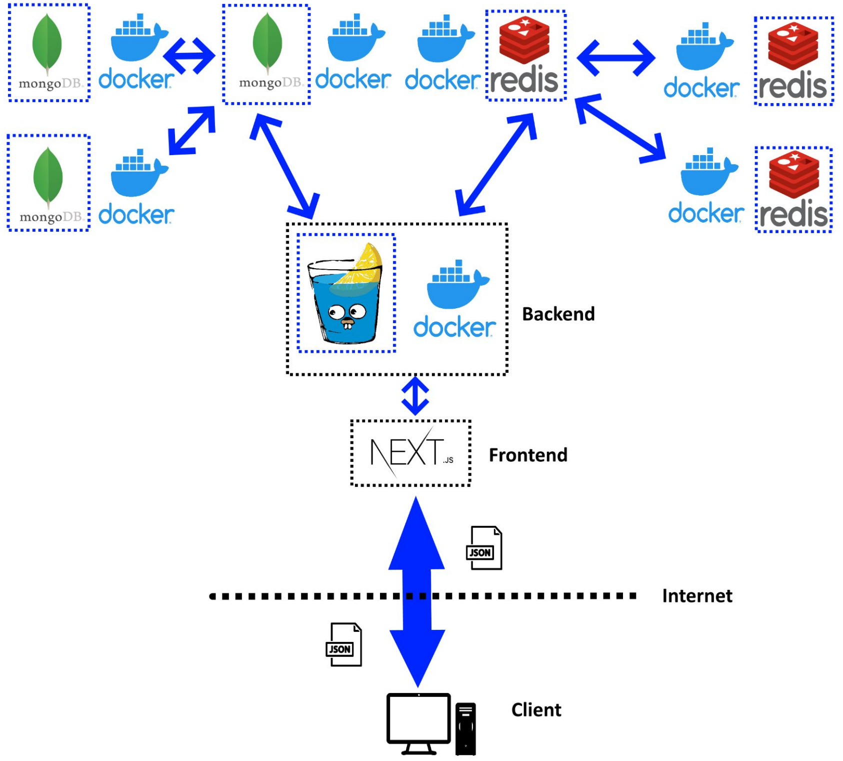
```
type Slot struct {
    Start             time.Time
    BookedAppointments int
    Employees         int
}
```

Software Architecture

This service will be implemented as a web app using the following ***tech-stack***



Software Architecture Diagram


mongoDB
docker
redis
Backend
NEXT.JS
Frontend
JSON
Internet
Client