Large-Scale and Multi-Structured Databases

Weather Prediction App

Rojan Shrestha

Alessio Simoncini

Lorenzo Vezzani

Application Highlights

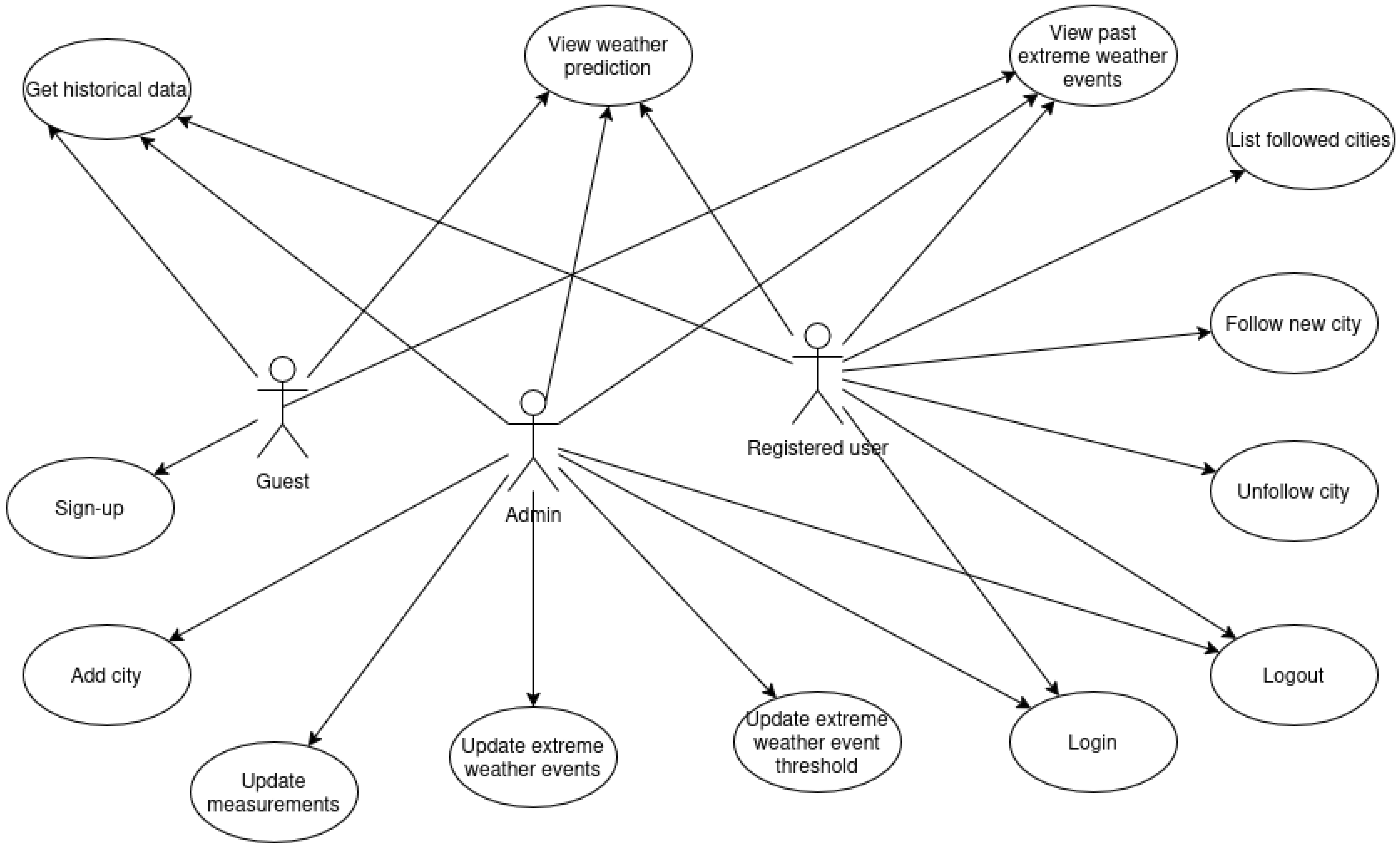
WeatherApp provides you with the weather information you need to plan your day with key features:

- **Weather Forecast** - Offers hourly, daily, or weekly weather forecasts, helping users plan ahead. Based on proximity and weather patterns the app even provides weather forecast of cities not in database.

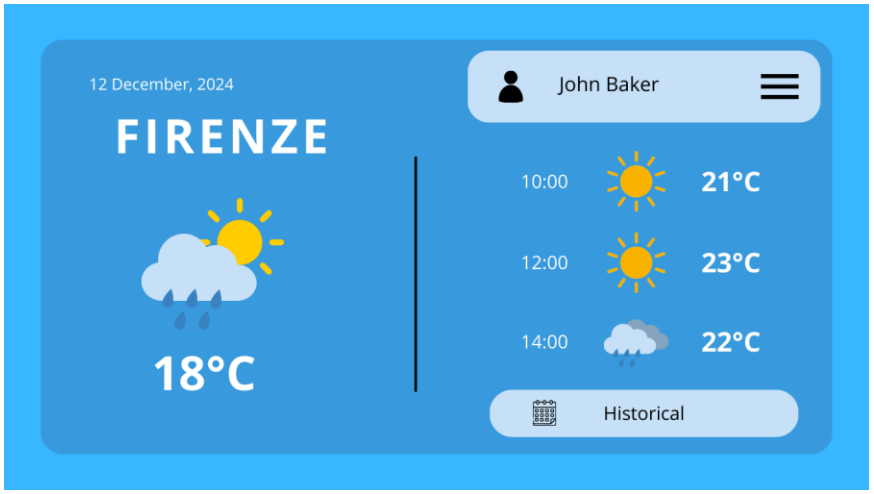
- **Weather Trends and History** - Offers historical weather data, allowing user to analyze past weather conditions for specific cities over selected time periods.

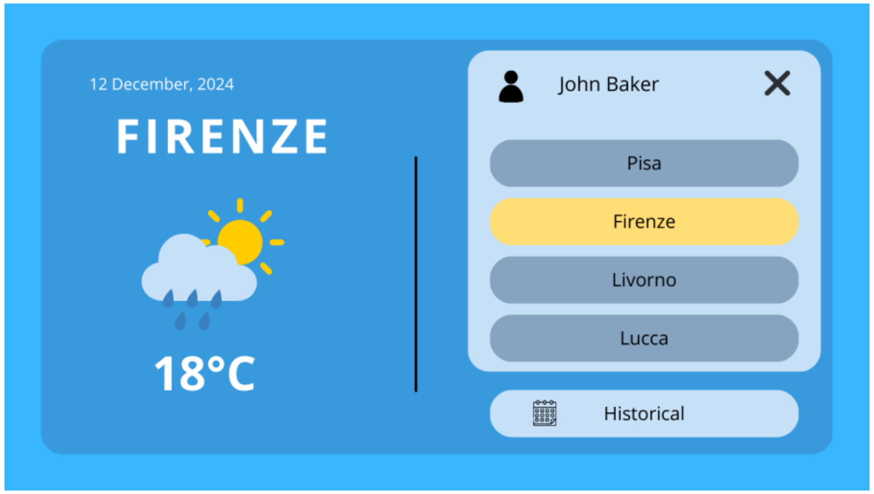
- **Extreme Weather events** – List of extreme weather conditions that users can use to get informated about storms , hurricanes or extreme temperatures.

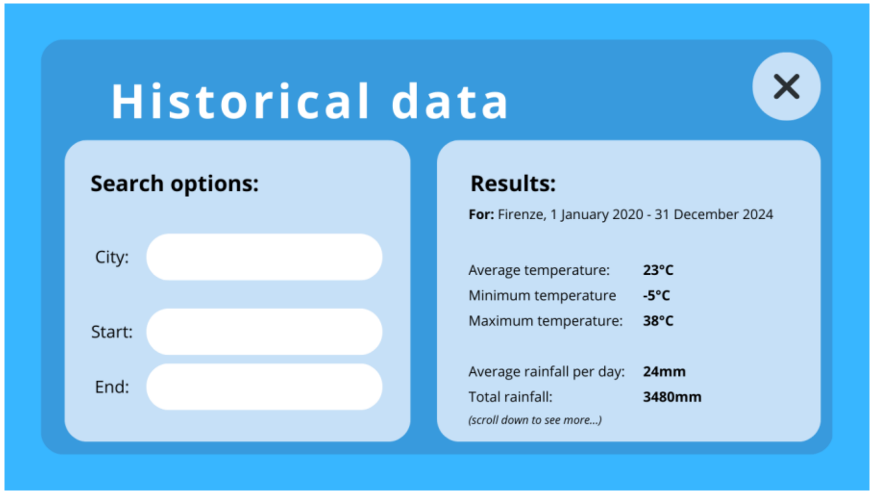
Actors


Get historical data
View weather prediction
View past extreme weather events
List followed cities
Follow new city
Unfollow city
Logout
Login
Update extreme weather event threshold
Update extreme weather events
Update measurements
Add city
Sign-up
Guest
Admin
Registered user

Main mock-ups







Results:

For: Firenze, 1 January 2020 - 31 December 2024

Average temperature:	**23°C**
Minimum temperature	**-5°C**
Maximum temperature:	**38°C**
Average rainfall per day:	**24mm**
Total rainfall:	**3480mm**

(scroll down to see more...)

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

IN SUPREMÆ DIGNITATIS 1343 UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Dataset Description

Source:
https://open-meteo.com/en/docs/historical-weather-api
Open-Meteo is an open source weather API that partners with national weather services around the globe.

Description: The hourly weather data over 80 years of any major city of Italy.

Volume: 75MB

Velocity/Variability: Weather information is important only when recent. After a certain period of time it will be relevant only for statistical purposes.

Non-functional requirements

Performance: Fast Read Times and Scalability

- Minimize read time as much as possible

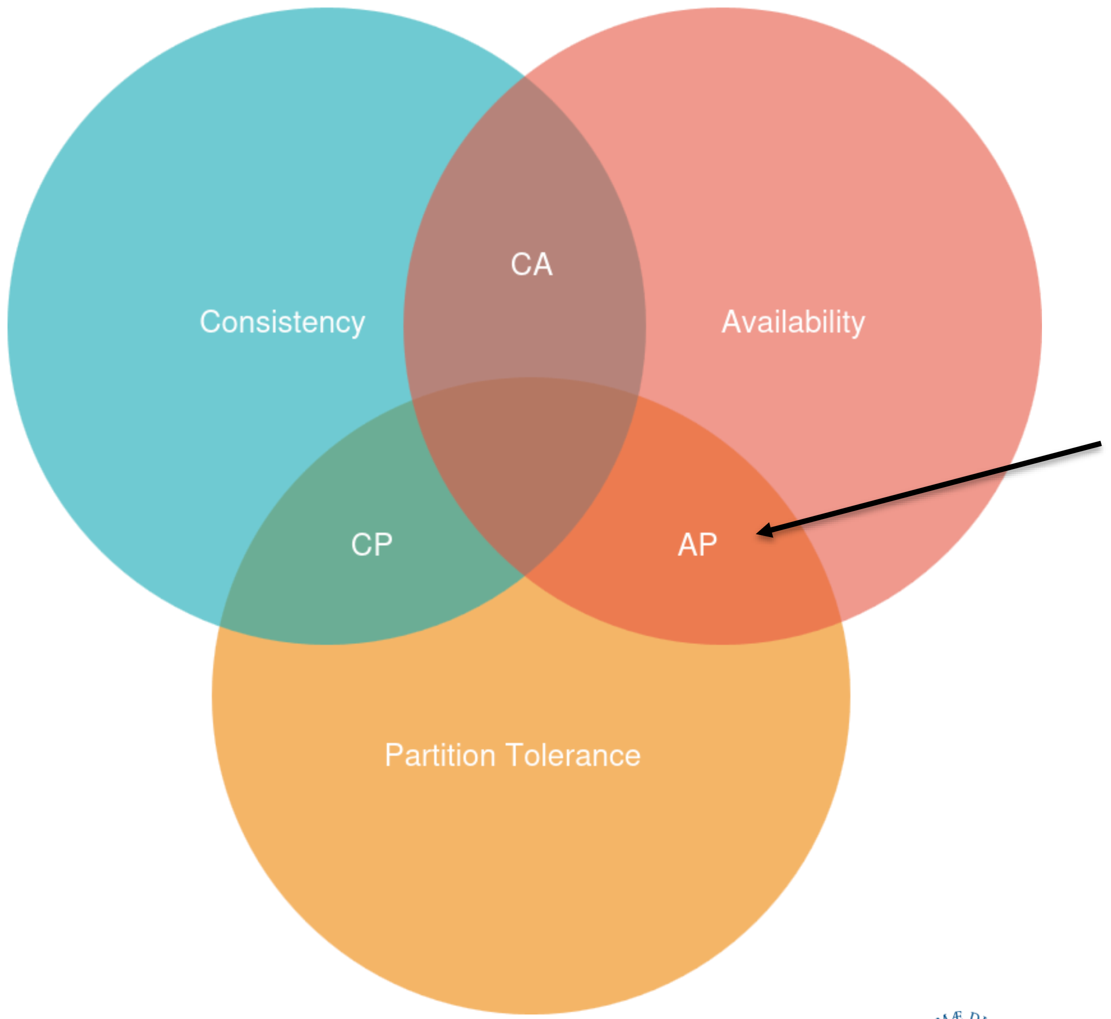
Performance: Availability

- High availability service
- Low latency even under heavy load

Security

- Authentication mechanism to ensure secure database access
- Upon login, the user receives a token. Token must be included in HTTP requests to access specific APIs

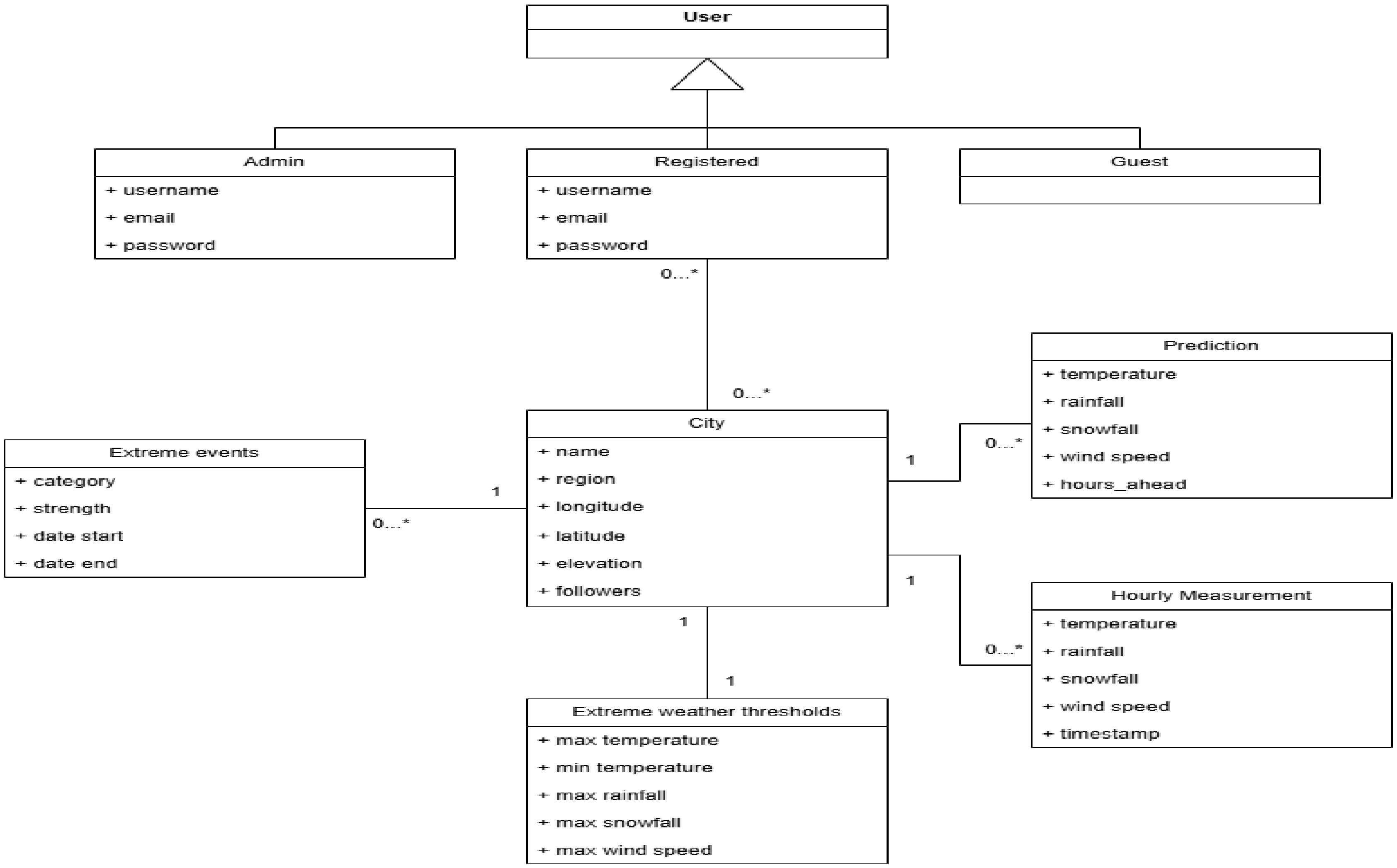
CAP theorem consideration



The application is focused on Availability and Network Partition Tolerance (AP).

- High availability to ensure fast reads, since the application is read-based
- The application is distributed, partition tolerance is not a given

UML Class Diagram



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Document DB: collections and indexes

- **Users**
  - `username`, `password`, `email`, `listCityId`

  📌 *Data linking* strategy: stores only city IDs

  Index on `username` for efficient login

- **Cities**
  - `name`, `region`, `coordinates`, `threshold`, `followers`, `city_id`, `eweList`

  📌 *Data embedding* used for extreme weather events (rare, read-heavy)

  Default _id index (sufficient for `city_id` access)

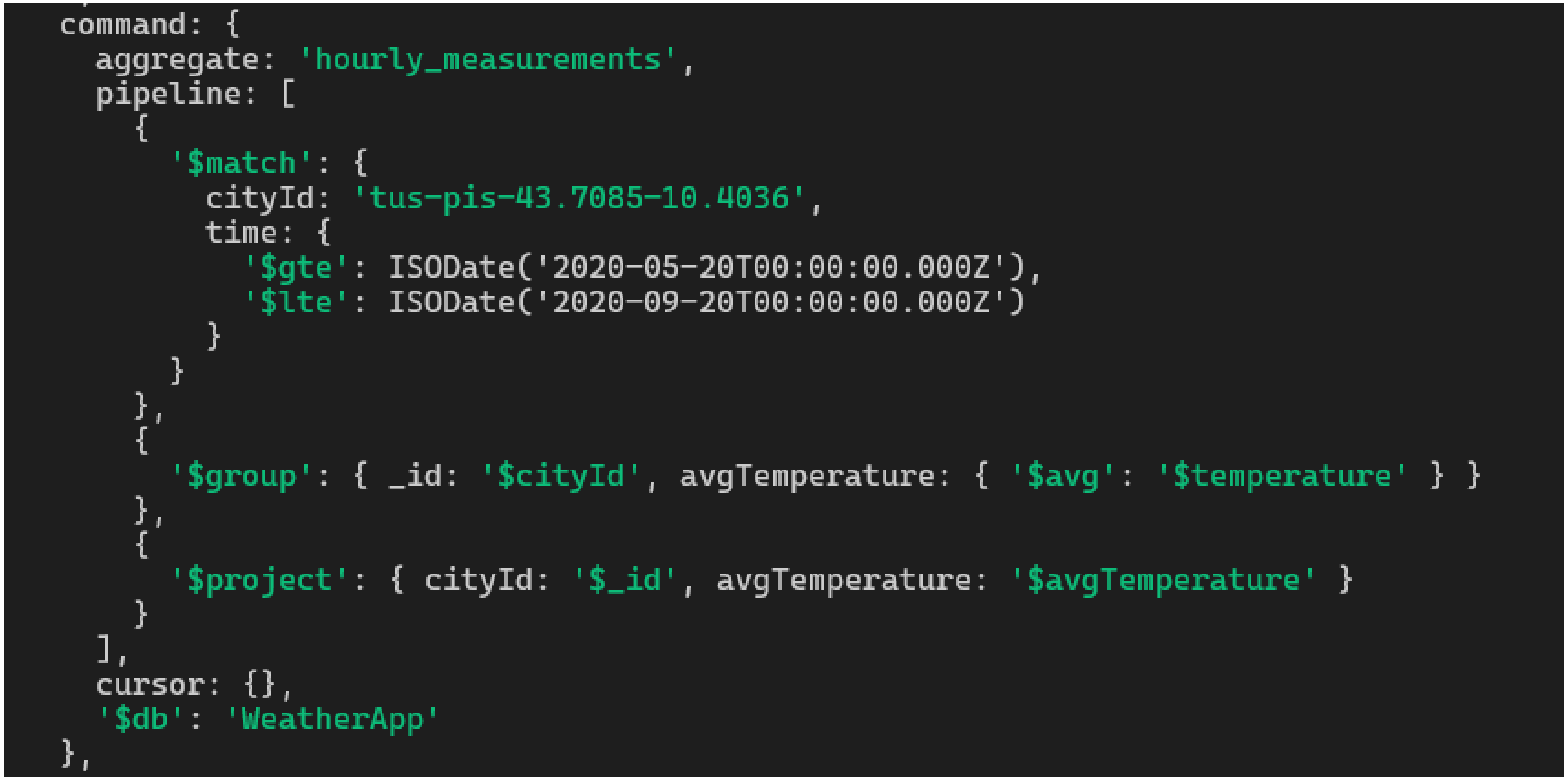
- **Measurements**
  - `temperature`, `rainfall`, `snowfall`, `windspeed`, `timestamp`, `city_id`

  📌 *Data linking* adopted: embedding would exceed MongoDB size limits

  Compound index on `city_id` and `timestamp`, expensive but essential for fast analytics over time

Experimental indexes validation

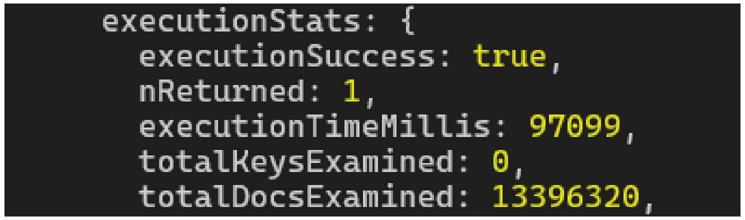
Chosen query:



```
  command: {
    aggregate: 'hourly_measurements',
    pipeline: [
      {
        '$match': {
          cityId: 'tus-pis-43.7085-10.4036',
          time: {
            '$gte': ISODate('2020-05-20T00:00:00.000Z'),
            '$lte': ISODate('2020-09-20T00:00:00.000Z')
          }
        }
      },
      {
        '$group': { _id: '$cityId', avgTemperature: { '$avg': '$temperature' } }
      },
      {
        '$project': { cityId: '$_id', avgTemperature: '$avgTemperature' }
      }
    ],
    cursor: {},
    '$db': 'WeatherApp'
  },
```

Experimental indexes validation

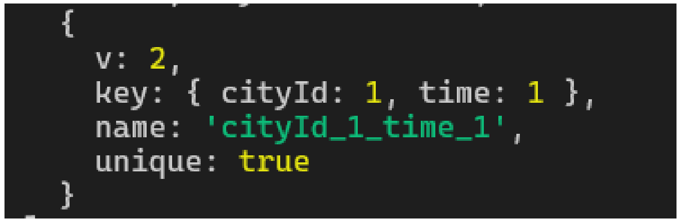
Without index:



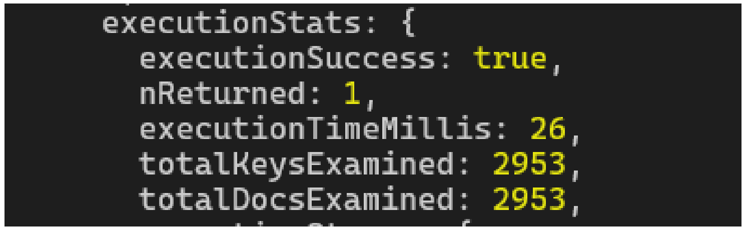
```
   executionStats: {
     executionSuccess: true,
     nReturned: 1,
     executionTimeMillis: 97099,
     totalKeysExamined: 0,
     totalDocsExamined: 13396320,
```

(The entire collection is examined)

With index:



```
  {
    v: 2,
    key: { cityId: 1, time: 1 },
    name: 'cityId_1_time_1',
    unique: true
  }
```



```
   executionStats: {
     executionSuccess: true,
     nReturned: 1,
     executionTimeMillis: 26,
     totalKeysExamined: 2953,
     totalDocsExamined: 2953,
```

Key-Value DB

Cities Storage

- **Key Format:**
  `city:{regioncode}-citycode-latitude-longitude`
  `e.g., city:{tus}-pis-43.7085-10.4036`
- **Value Type:** HASH
  ➤ Stores useful metadata (e.g., region, elevation)

Forecast Storage

- **Key Format:**
  `forecast:{regioncode}-citycode-latitude-longitude:date`
  `e.g., forecast:{tus}-pis-43.7085-10.4036:2025-06-08`
- **Value Type:** STRING
  ➤ Daily forecast including:
  temperature, rainfall, snowfall, windspeed, timestamp

Redis is also used for storing user **token**: created at login and deleted at logout.

Handling intra-DB consistency

The only shared entity between **MongoDB** and **Redis** is `city.`
The critical operation is saveCity(·):

- Adding the city to MongoDB.
- Storing city metadata in Redis.

Rollback Mechanism

- In case of any failure:
  - The inserted city is **deleted from MongoDB** (if added).
  - The corresponding Redis hash is **deleted** (if added).

Result

- The `saveCity(·)` operation ensures **strong intra-database consistency**.
- No partial insertions of `city` remain if the process is interrupted.

Clustering and Sharding: mongo-conf

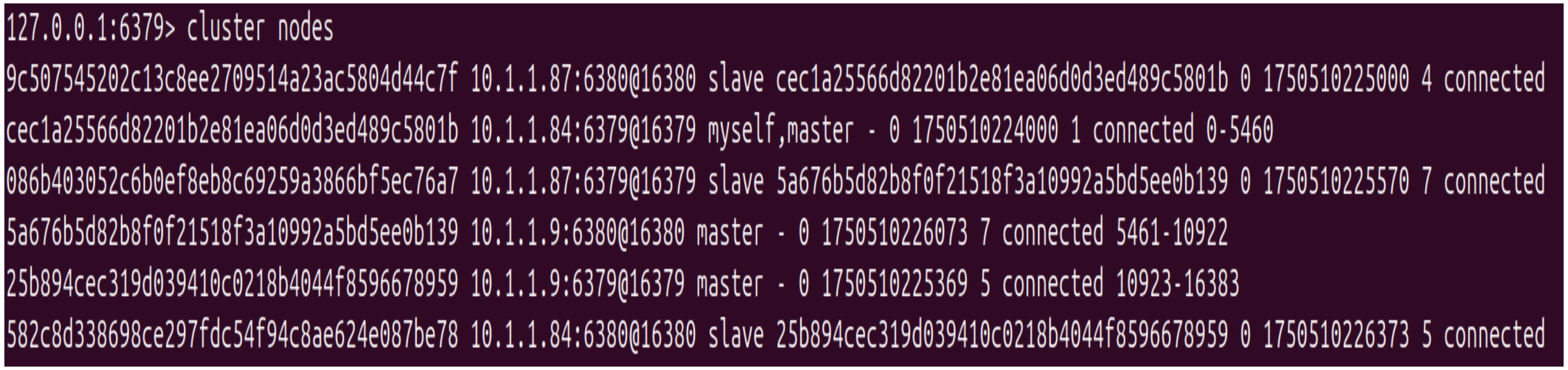
```
mongod --replSet lsmdb --dbpath ~/data --port 27020 --bind_ip localhost,10.1.1.9 --oplogSize 200

mongod --replSet lsmdb --dbpath ~/data --port 27020 --bind_ip localhost,10.1.1.84 --oplogSize 200

mongod --replSet lsmdb --dbpath ~/data --port 27020 --bind_ip localhost,10.1.1.87 --oplogSize 200
```

```
rsconf = { _id: "lsmdb",
    members: [
        {_id: 0, host: "10.1.1.9:27020", priority:1},
        {_id: 1, host: "10.1.1.84:27020", priority:2},
        {_id: 2, host: "10.1.1.87:27020", priority:5}
]};
```

Clustering and Sharding: redis-conf

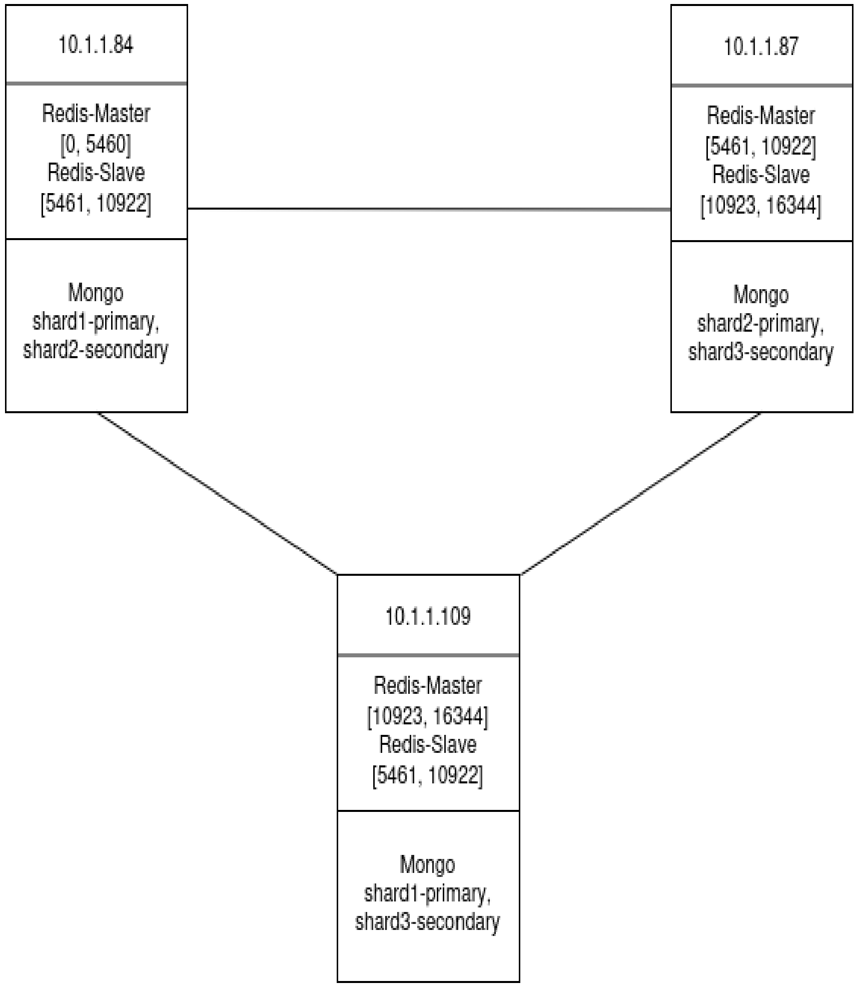
```
port 6379 # 6380 for the slaves
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 5000
appendonly yes
bind 0.0.0.0
maxmemory 512mb
maxmemory-policy allkeys-lfu
```

Runned on 10.1.1.84:6379



```
127.0.0.1:6379> cluster nodes
9c507545202c13c8ee2709514a23ac5804d44c7f 10.1.1.87:6380@16380 slave cec1a25566d82201b2e81ea06d0d3ed489c5801b 0 1750510225000 4 connected
cec1a25566d82201b2e81ea06d0d3ed489c5801b 10.1.1.84:6379@16379 myself,master - 0 1750510224000 1 connected 0-5460
086b403052c6b0ef8eb8c69259a3866bf5ec76a7 10.1.1.87:6379@16379 slave 5a676b5d82b8f0f21518f3a10992a5bd5ee0b139 0 1750510225570 7 connected
5a676b5d82b8f0f21518f3a10992a5bd5ee0b139 10.1.1.9:6380@16380 master - 0 1750510226073 7 connected 5461-10922
25b894cec319d039410c0218b4044f8596678959 10.1.1.9:6379@16379 master - 0 1750510225369 5 connected 10923-16383
582c8d338698ce297fdc54f94c8ae624e087be78 10.1.1.84:6380@16380 slave 25b894cec319d039410c0218b4044f8596678959 0 1750510226373 5 connected
```

Clustering and Sharding


10.1.1.84
Redis-Master [0, 5460] Redis-Slave [5461, 10922]
Mongo shard1-primary, shard2-secondary
10.1.1.87
Redis-Master [5461, 10922] Redis-Slave [10923, 16344]
Mongo shard2-primary, shard3-secondary
10.1.1.109
Redis-Master [10923, 16344] Redis-Slave [5461, 10922]
Mongo shard1-primary, shard3-secondary

VMs Available

- Deployed on 3 UniPi VMs:
  - `10.1.1.9`, `10.1.1.84`, `10.1.1.87`
- Read-Optimized Architecture

MongoDB Clustering

- Write concern: `w=1` → availability
- Read preference: `local` → fast reads

Mongo Sharding Strategy (planned)

- `user`: shard by `_id`
- `city`: shard by `_id`
- `hourly_measurement`: shard by `cityId`

Redis Sharding

- Keys structured with region code inside `{}`:
  - `city:{tus}-pis-…`
  - `forecast:{tus}-pis-…-2025-01-01`

Swagger UI REST APIs documentation

User

- `POST /user/register`
- `POST /user/login`
- `POST /user/logout`

Forecasts

- `GET /forecast/today`
- `GET /forecast/today/arbitrary-city`
- `GET /forecast/day`
- `GET /forecast/day/arbitrary-city`

City

- `POST /city/add`
- `PUT /city/update-thresholds`
- `GET /city/by-name`
- `GET /city/all`
- `GET /favorites`
- `PUT /favorites`
- `DELETE /favorites`

Swagger UI REST APIs documentation

Extreme Weather Event (Admin)
- `DELETE /ewe/duplicates/range`
- `DELETE /ewe/duplicates/all`

Admin – Data Update
- `PUT /data-manager/update/forecasts`
- `PUT /data-manager/update/measurements`
- `PUT /data-manager/update/ewes`

(Some) Analytics
- `GET /analytics/measurement/city/average-per-month`
- `GET /analytics/measurement/recent/city/total-per-day` / `average-per-day`
- `GET /analytics/ewe/strength/maximum` / `average`
- `GET /analytics/ewe/duration/longest` / `average`
- `GET /analytics/ewe/count` / `count-per-month` / `count-of-at-least-strength`

Live Demo with Postman