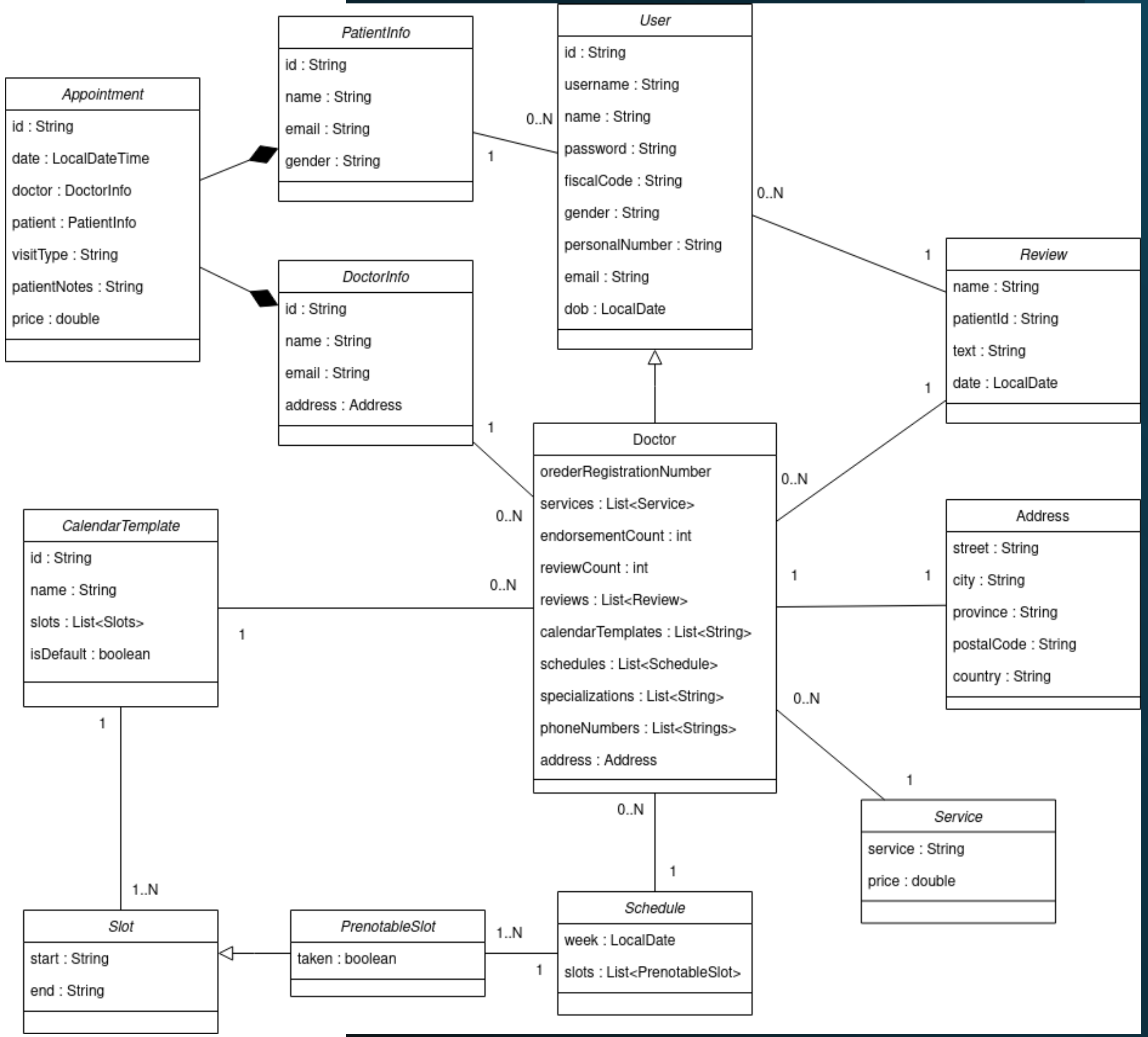
HealthHub

A Healthcare Data Management System

Authors:
Paolo Palumbo
Francesco Panattoni
Nedal Elezaby

UML Class Diagram

- At its core, we model **Users**, which can be either **Patients** or **Doctors**. Doctors inherit from Users and are associated with **Services**, **Reviews**, and **CalendarTemplates**.
- **Appointments** contain denormalized information from both the **Doctor** and the **Patient**, ensuring data stability even if users are later deleted or modified.
- We also introduced entities like **Schedule** and **Slot** to handle time availability, and **Address** is embedded in doctors for spatial filtering.
- By using **composition**, we express ownership and life-cycle dependency. For example a Doctor owns a list of Reviews and Services, which do not exist independently.



Dataset Source and Composition

- **Web Scraped** from MioDottore.it: ~700 K reviews, 215 K unique users, 88 K doctors.
- **Synthetic generation**: appointment history, "likes" network, demographic profiles.
- **Format**: JSON documents (~960MB total), split into doctors.json, users.json, appointments.json, templates.json, user_likes.json.

Velocity & Variety

- **Velocity**: simulates > 100 new reviews/day, ~450 new doctors/year
- **Variety**: structured profiles, unstructured text reviews, time-series appointments, graph interactions (endorsments, reviews)

Non-Functional Requirements

- **Performance & Scalability**
  - Acceptable response times for common operations
  - Efficient handling of load peaks

- **Availability & Reliability**
  - 24/7 availability with failover mechanisms
  - Backup and recovery procedures
  - Tolerance to data inconsistency in non-critical views

- **Security & Privacy**
  - Secure and authenticated access for all users
  - Encryption of data in transit and at rest
  - Protection against attacks (e.g. injection)

Non-Functional Requirements

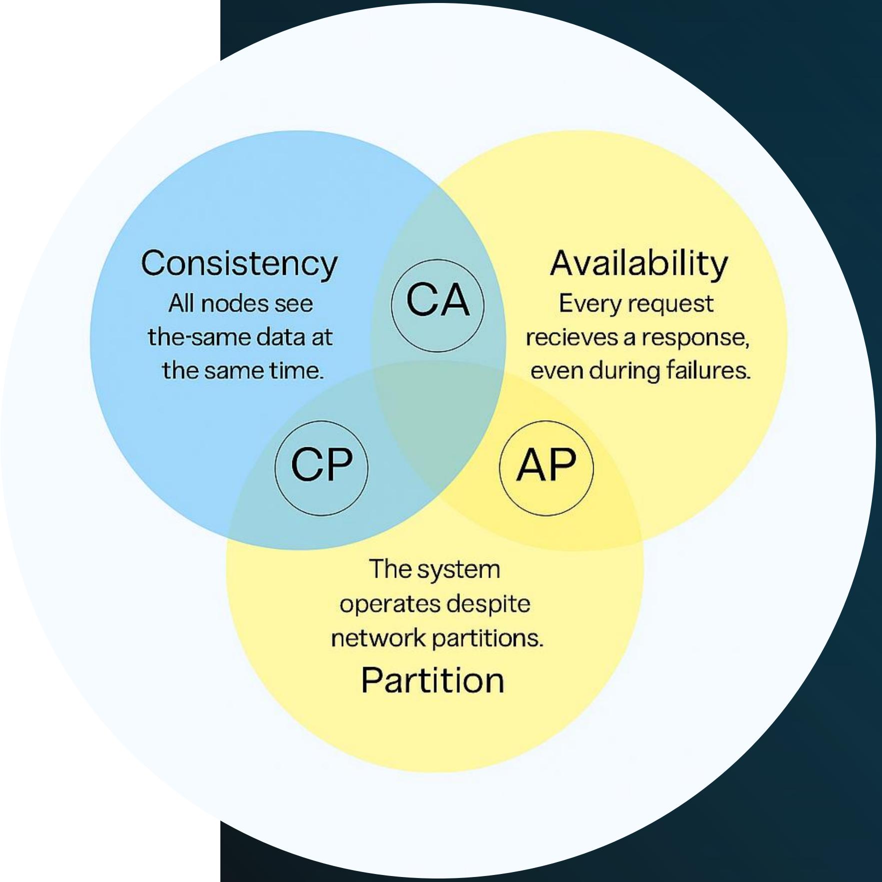
- **Usability**
  - Intuitive and user-friendly interface
  - Low latency in user interactions

- **Portability & Flexibility**
  - Deployment on multiple operating systems (Windows, macOS, Linux)
  - Modular and easily extensible architecture

- **Maintainability**
  - Code based on OOP principles and modularity
  - Reduction of single points of failure
  - Complete documentation and comments for future maintenance

CAP Theorem Handling

- **AP-oriented**: prioritize Availability & Partition-Tolerance.
- **Write concern**: w:1 for low latency.
- **Read concern**: local for general, majority for schedules.
- **Eventual consistency** for social graphs and **strong consistency** only for booking flows.



MongoDB Collections

```
{
    _id: ObjectId('684adad337804916ca657645'),
    name: 'Silvia Minisini',
    email: 'silvia.minisini@yahoo.com',
    username: 'silvia_minisini',
    password: '5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8',
    address: {
      street: 'Via Michelangelo Buonarroti 10,',
      city: 'Grado',
      province: 'GO',
      country: 'IT',
      postalCode: '34073'
    },
    phoneNumbers: [],
    specializations: [ 'Medico di Base' ],
    services: [],
    endorsementCount: 9,
    reviews: [
      {
        patientId: ObjectId('684ada4537804916ca636428'),
        name: 'Dott. Giampaolo Draghi',
        text: 'Gentilissima, sa mettere i bambini Nella tranquillità. Molto precisa è competente nel
suo lavoro !!Mia bambina La adora !!Per dare la fiducia a bambina Prima visitava la sua bambola .!!Poco
dire una dottoressa PERFETTA!!!',
        date: ISODate('2018-07-08T07:24:25.424Z')
      },
      {
        patientId: ObjectId('684ada4637804916ca64e5d8'),
        name: 'Raffaello Pederiva',
        text: 'Competente e cortese, ha dato indicazioni precise e dedicato tutto il tempo necessario
alla visita. Ottima prestazione.',
        date: ISODate('2017-10-10T18:57:18.424Z')
      }
    ],
    reviewsCount: 2,
    dob: ISODate('2006-05-02T00:00:00.000Z'),
    fiscal_code: 'MINSIL060502FYSF',
    orderRegistrationNumber: 'GO-420665',
    calendarTemplates: [ ObjectId('684ad9f537804916ca60d95b') ]
  }
```

•**User**: includes ID, first name, last name, email, password, date of birth, and role.
•**Doctor**: extends User with specialization, reviews, clinic address, and weekly availability.

```
{
    _id: ObjectId('684ada4537804916ca639e42'),
    fiscalCode: 'PINSIG860130F930',
    name: 'Sig.ra Pina Cerutti',
    password: '5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8',
    dob: ISODate('1986-01-30T00:00:00.000Z'),
    gender: 'female',
    personalNumber: '0782946312',
    email: 'zginese@gmail.com',
    username: 'Fasica'
  }
```

MongoDB Collections

- **Templates**: name, daily time slots, default flag.
- **Appointments**: date, doctor & patient info, visit type, notes, price.

```
{
  _id: ObjectId('684ad9f537804916ca60e5d5'),
  name: 'Standard',
  slots: {
    monday: [
      { start: '08:30', end: '09:00' },
      { start: '09:00', end: '09:30' },
      { start: '09:30', end: '10:00' },
      { start: '10:00', end: '10:30' },
      { start: '10:30', end: '11:00' },
      { start: '11:00', end: '11:30' },
      { start: '11:30', end: '12:00' },
      { start: '12:00', end: '12:30' }
    ],
    wednesday: [
      { start: '14:30', end: '15:00' },
      { start: '15:00', end: '15:30' },
      { start: '15:30', end: '16:00' },
      { start: '16:00', end: '16:30' },
      { start: '16:30', end: '17:00' },
      { start: '17:00', end: '17:30' },
      { start: '17:30', end: '18:00' },
      { start: '18:00', end: '18:30' }
    ],
    friday: [
      { start: '10:00', end: '10:30' },
      { start: '10:30', end: '11:00' },
      { start: '11:00', end: '11:30' },
      { start: '11:30', end: '12:00' },
      { start: '16:00', end: '16:30' },
      { start: '16:30', end: '17:00' },
      { start: '17:00', end: '17:30' },
      { start: '17:30', end: '18:00' }
    ]
  },
  isDefault: true
}
```

```
{
  _id: ObjectId('684adc6637804916ca6d123e'),
  date: ISODate('2025-05-22T13:31:59.000Z'),
  doctor: {
    _id: ObjectId('684adad437804916ca65ed9a'),
    name: 'Saverio Fania',
    address: {
      street: 'Piazza Madre Teresa di Calcutta, 5/10,',
      city: 'Cerignola',
      province: 'FG',
      country: 'IT',
      postalCode: '71042'
    },
    email: 'saverio.fania@live.com'
  },
  patient: {
    _id: ObjectId('684ada4637804916ca651c3f'),
    name: 'Ottone Curatoli',
    fiscalCode: 'CUROTT361018MSOE',
    email: 'bellodonato@tin.it',
    gender: 'male'
  },
  visitType: 'Visita otorinolaringoiatrica di controllo',
  patientNotes: '',
  price: 90
}
```

MongoDB - Doctor Search Query (Full-Text Search)

- Search doctors by name, specialization, city, province or combination
- Use MongoDB text index with weighted fields for relevance scoring
- Pipeline steps:
  - `$match`: filter documents matching search text
  - `$project`: add text relevance score (`textScore`)
  - `$sort`: order by descending relevance
  - `$limit`: return top 250 results
- Returns most relevant doctors dynamically as user types

MongoDB - Earnings & Visit Type Analytics

- **Monthly Earnings**:
  - Filter appointments by doctor and year
  - Extract month from date and sum prices by month
  - Return map month → total revenue for dashboard visualization
- **Visit Type Summary**:
  - Filter appointments by doctor
  - Group by visit type and count occurrences
  - Return distribution of visit types for pie chart display

MongoDB - New Patients & Weekly Visits Analytics

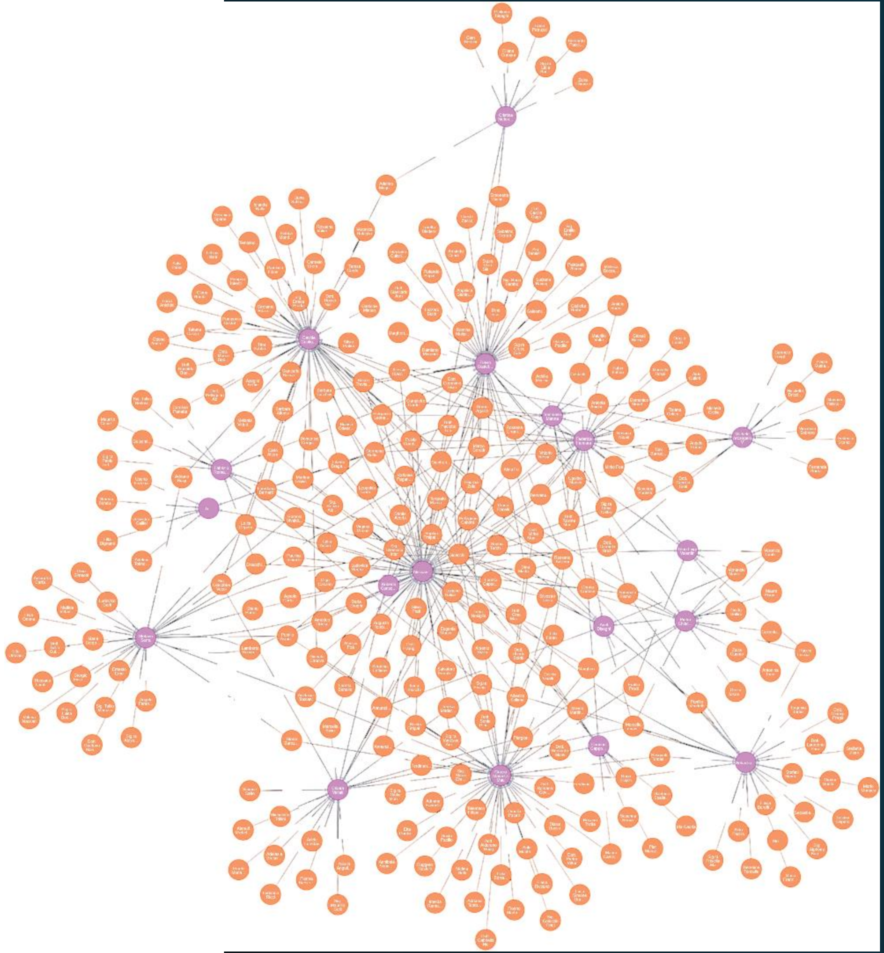
- **New Patients of the Month**:
  - Filter appointments by doctor
  - Group by patient and find earliest visit date
  - Count patients whose first visit is in the target month
- **Visits per Day in Week**:
  - Filter appointments by doctor and week range
  - Extract day of week and count visits per day
  - Return map day → visits to analyze weekly workload

Neo4j

- **Node Types:**
  - User {id,name}
  - Doctor {id,name,specializations[]}

- **Relationships:**
  - (U)-[:REVIEWED]->(D)
  - (U)-[:ENDORSED]->(D)

- **Use:** Recommendation engine and search-optimization



Neo4j – Recommendation System

- **1. Collaborative Filtering**
  - Identify “similar” users who share ≥3 endorsed/reviewed doctors with the target user
  - Discover candidate doctors from those users via endorsement/review paths (≤3 hops) not yet seen by the target user
  - Score candidates by cumulative similarity strength and endorsement count; return top-N personalized recommendations
  - Complexity: O(E_shared + E_rec)
- **2. Popularity-Based Fallback**
  - Select all doctors with non-null specializations
  - Compute a popularity score = total number of endorsements/reviews from all users (zero-review doctors included)
  - Return top-N doctors ranked by descending popularity
  - Ideal for cold-start or sparse personalized results
- **3. Random Sampling Integration**
  - Fetch up to 10×limit personalized recommendations
  - If insufficient, augment with 2×remaining slots of popular doctors (avoiding duplicates)
  - Shuffle combined list and take first N to ensure diversity and novelty

Neo4j – Search Query Sypport

- **Logged-in Users Only**
- **Leverages patient's social and behavioral context for personalized results.**
- **Dual-Database Strategy**
- **MongoDB:** Fast text-based retrieval on doctor name and specializations
- **Neo4j:** Graph traversal for personalization within patient's social network
- **Personalization via Neo4j**
- **Social Neighborhood:** Match paths (1–3 hops) from user to doctors via REVIEWED/ENDORSED
- **Text Filter:** Case-insensitive search on `d.name` or any `d.specializations`
- **Proximity Scoring:**
  - Compute `steps = min(length(path))`
  - Assign `score = 5 - steps` (closer = higher score)
- **Ranking & Limit:** Order by ascending steps, return top 250

Consistency Management

- **Two-phase updates**: MongoDB → Neo4j with *@Transactional* and *@Async*.
- **Rollback** on Neo4j failure to maintain atomicity.
- MongoDB was deployed as a **three-node replica** set with primary-based writes and reads for consistency; Neo4j runs in **standalone mode**.
- **Tolerate eventual consistency** on social interactions for performance.

Sharding Strategy (Design)

- **Doctors**: shard key = address.province (hashed) → even geo distribution.
- **Appointments**: shard key = appointmentDateTime (range) → efficient time-range queries.
- **Neo4j**: no sharding (enterprise pricing, traversal overhead).

System Architecture

- **Front-end**: Thymeleaf + JS + AJAX in browser.
- **Back-end**: Spring Boot + embedded Tomcat, REST APIs.
- **Databases**: MongoDB replica set, standalone Neo4j.
- **Infrastructure**: 3-node cluster (VMs or containers)

Future Work

- Implement **real sharding** for Neo4j (enterprise).
- Add **geospatial indexing** (MongoDB Atlas).
- Enhance **ML-driven** recommendations.
- Expand **microservices** for better maintainability.

Conclusions

- Successfully combined document-based (MongoDB) and graph-based (Neo4j) stores to satisfy diverse functional needs (flexible user profiles, high-throughput bookings, and real-time recommendations).
- Prioritized availability and partition-tolerance for social features, while enforcing strong consistency on critical booking workflows to prevent data anomalies.
- Developed a working, user-friendly and community-friendly application.