HealthMatch

Project Deliverable Two

https://github.com/ejd230001/3354-HealthMatch

Team Members:

- Elden Deguia
- Mathew Biji
- Hudson Duong
- Tanzeel Jaffery
- Timothy Mathew
- Vidur Nangia
- Dylan Tirandaz

Project Description

HealthMatch is a comprehensive, management-facing application that allows businesses to correctly choose healthcare providers for their employees using relevant data. HealthMatch will collect data on employee's needs as well as management's cost restraints to find an ideal healthcare provider match that meets all parties' needs. Through usable design and efficient software, HealthMatch will make the task of choosing healthcare providers easier than ever before.

Task Delegation/Details of what each member should be working on

- **Elden Deguia** – Organizing the report, creating architectural design diagram, creating a test commit
- **Mathew Biji** – create Sequence Diagrams, create conclusion page
- **Hudson Duong** – create use case diagram, compare work to similar designs
- **Tanzeel Jaffery** – Create the project scope document and create the Software Process Model Description
- **Timothy Mathew** – create Software Requirements Diagram estimate cost, effort, and pricing
- **Vidur Nangia** – Making the first commit to the repo, addressing feedback for the project proposal, estimating project scheduling
- **Dylan Tirandaz** – create Class Diagram create references page

Software Process Model Description

Chosen Model: Agile Software Development Model

HealthMatch will be developed using the **Agile process model**, emphasizing *iterative development*, *team collaboration*, and *continuous feedback integration* [1]. Given the evolving nature of stakeholder needs and the emphasis on usability, Agile provides the flexibility required to refine the system incrementally as feedback is gathered.

Rationale for Choosing Agile

- **Iterative Progress:**
  The project can be broken into short sprints, allowing continuous progress toward functional components such as data collection, matching algorithms, and the user interface.

- **Flexibility and Adaptation:**
  Requirements like new provider attributes or updates in cost criteria can be easily incorporated in future iterations without restarting development.

- **Team Collaboration:**
  Since each team member is responsible for distinct deliverables (e.g., diagrams, architecture, documentation), Agile encourages synchronized progress and seamless integration of work.

Implementation for HealthMatch

1. **Requirements Gathering**
   a. Define initial functional and non-functional requirements through team discussion and feedback.

2. **Design Phase**
   a. Create UML diagrams (use case, class, sequence, and architecture) that evolve with each sprint.

3. **Implementation Phase**
   a. Develop modular features, beginning with the employee input interface, provider database, and matching logic.

4. **Testing and Feedback**
   a. Conduct iterative testing after each sprint to verify usability and performance metrics.

5. **Deployment**
   a. Prepare a stable release version by the final sprint for demonstration.

6. **Maintenance and Improvement**
   a. Incorporate final feedback into post-release adjustments.

Expected Outcome

Using the Agile model ensures that **HealthMatch** remains adaptable and functional throughout its lifecycle. It enables the team to iteratively improve the application's performance, maintain usability, and align closely with stakeholder expectations. The result will be a **reliable, scalable, and meaningful healthcare matching solution**.

Software Requirements

The following section defines the functional and non-functional requirements for the HealthMatch system. These requirements are the core features, constraints, and quality attributes necessary to ensure the system operates efficiently, and securely.

Functional requirements

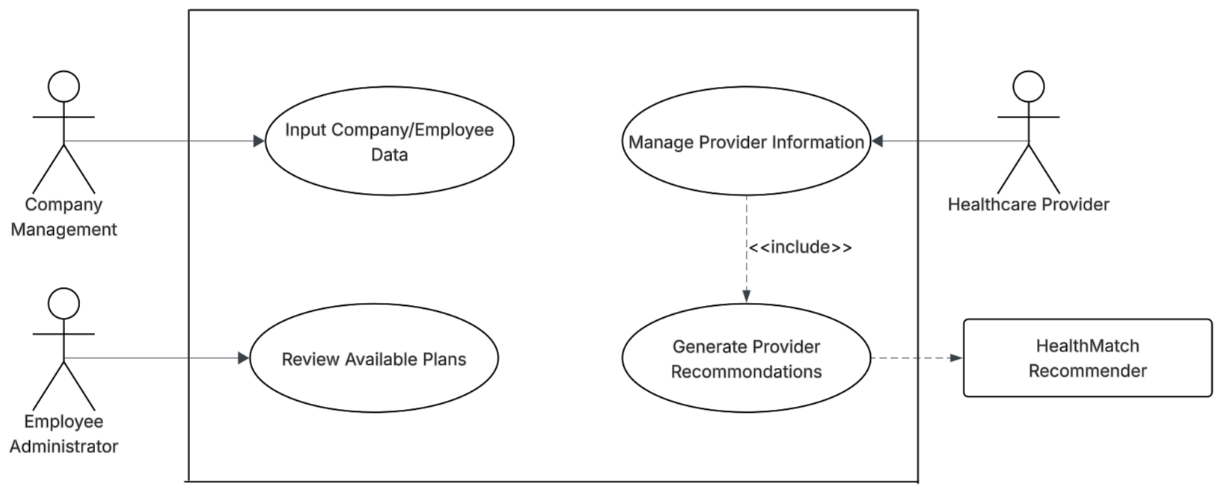
- This system shall enable a user who is privileged to the Company Management level to upload, insert and update company and employee data using the user interface.
- The system shall create a prioritized list of healthcare provider suggestions to the Company Management by running an employee needs and company cost constraint matching algorithm.
- The system shall enable an employee administrator to request and see the list of available healthcare plans in the database.
- The system shall offer a way, through which a Healthcare Provider can access a list of providers and make updates on the information in the database.
- The system shall ensure a session and permissions are verified before any request is made by a user to access or modify any data.
- The system shall enable a Manager to produce reports depending on the matches of healthcare providers generated by the system.

Non-functional requirements

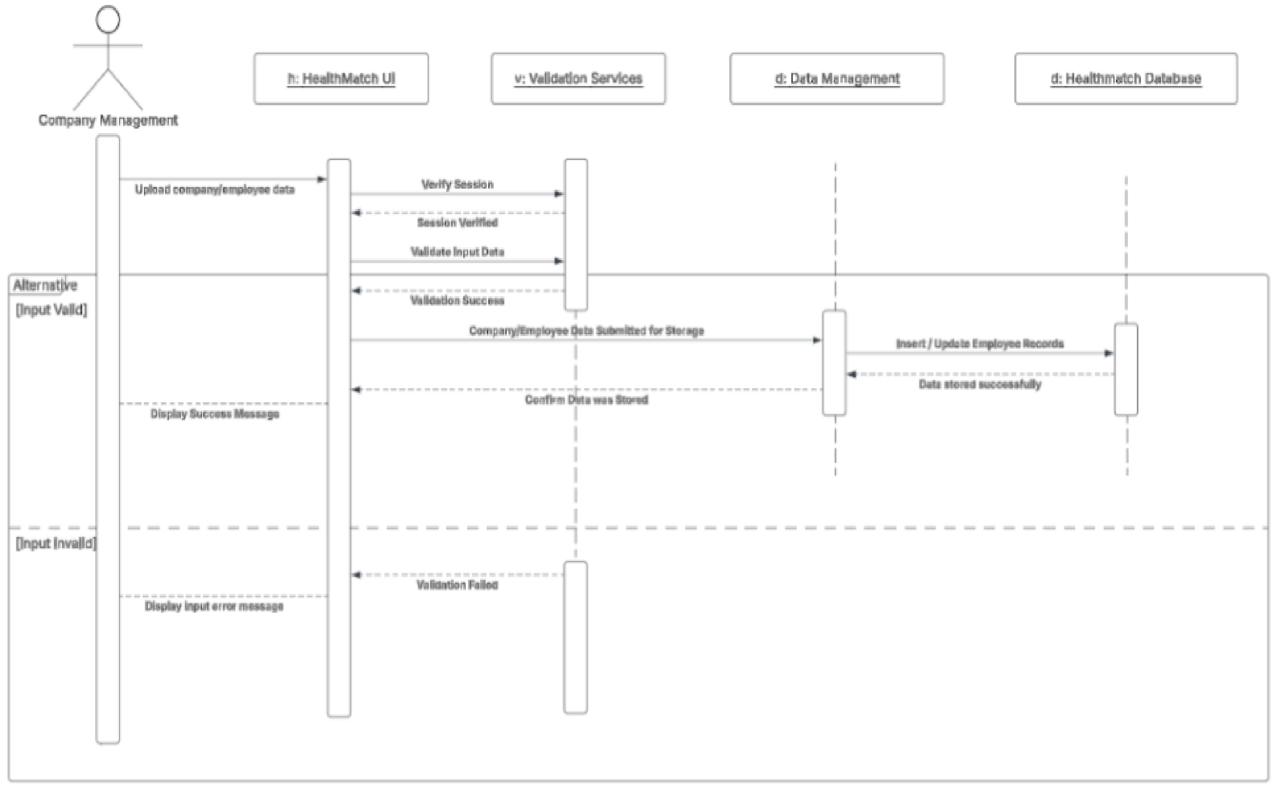
- Product requirements:
    - Usability requirements: A new Company Manager shall be able to upload employee data and produce a list of recommendations within a maximum of one hour of training.
    - Efficiency requirements:
        - Performance requirements: The recommendation engine in the system shall take the inputs and present the generated provider recommendations in less than 15 seconds upon request by the user.
        - Space requirements: The installed program along with the necessary databases shall not take over 1 GB of persistent storage.
    - Dependability requirements: The system shall be operational during regular working hours (Monday-Friday, 9:00 AM to 5:00 PM) and it shall have a 99.5 uptime.

    - Security requirements: The database shall store all user passwords in an encrypted and hashed form in order to stop unauthorized users.
- Organizational Requirements:
    - Environmental requirements: The application shall be a web based system that fully works on Google Chrome, Mozilla Firefox, Microsoft Edge, etc.
    - Operational requirements: The system database shall be backed up on a daily basis and the backup shall last at least 30 days in order to recover the data.
    - Development requirements: The system shall be built according to the agile process model, and the development of progress shall be divided into iterative sprints to meet the constantly changing requirements.
- External requirements:
    - Regulatory requirements:
        - The system shall be in accordance with all provisions of the Health Insurance Portability and Accountability Act (HIPAA) with regard to privacy and security of patient information [3].
    - Ethical requirements: The matching algorithm shall not employ any demographic information (such as age, gender) in its algorithm in order to get impartial and unbiased recommendations.
    - Legislative requirements:
        - Accounting requirements: The system shall keep an audit trail of all data additions, alterations, and recommendation creation activities including the user who is responsible and a date timestamp.
        - Safety/security requirements: Any company management roles shall have certain restricted access to sensitive employee health data based on the authorized roles in which they possess allowing access to the system according to the access control policies of the system.

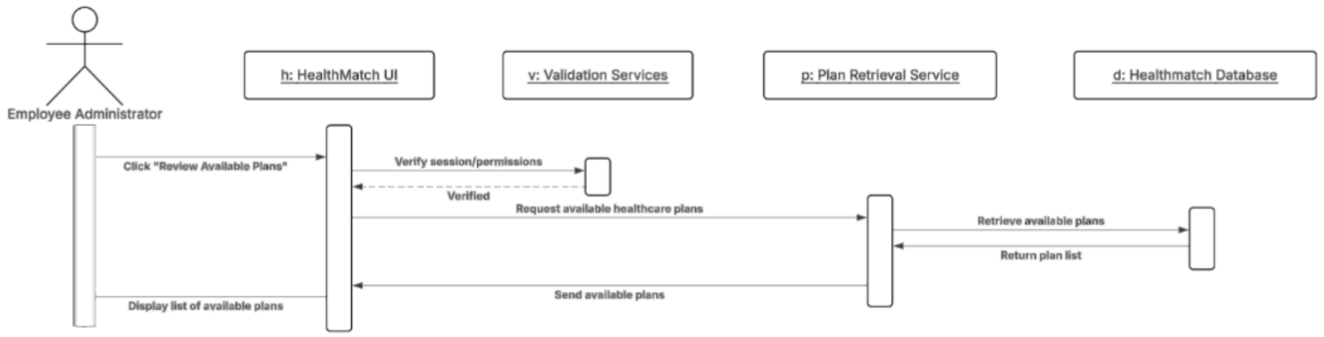
Use Case Diagram



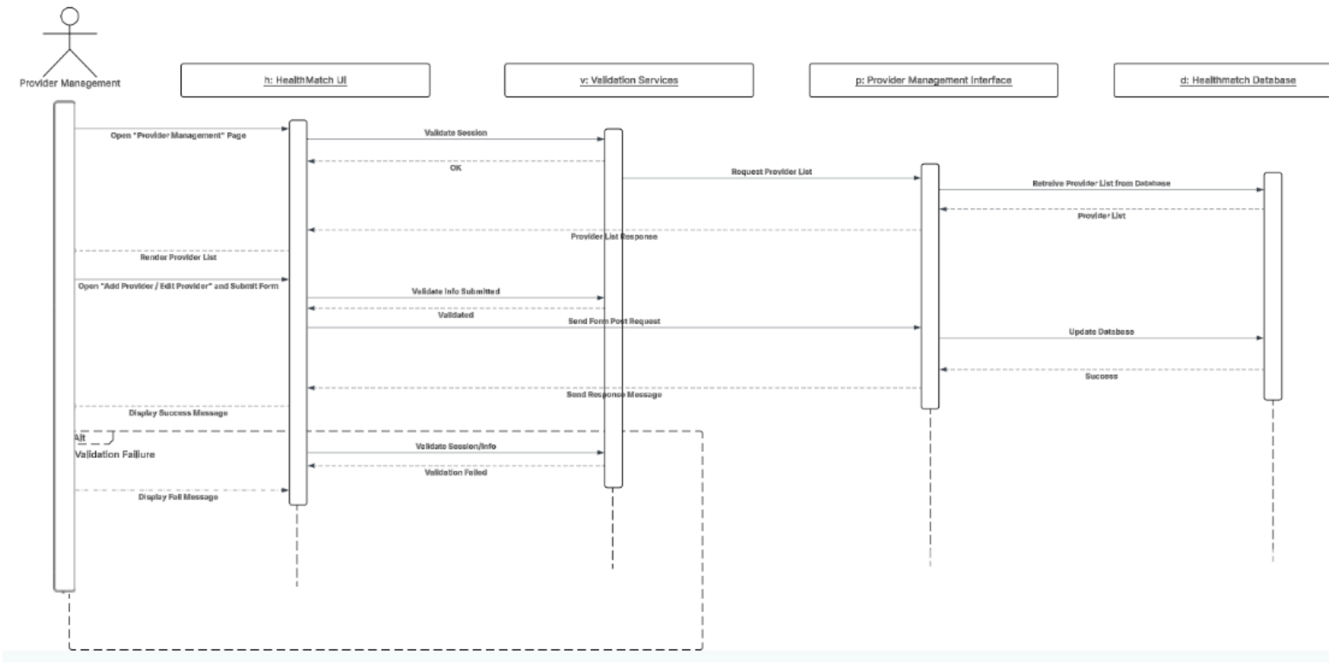
Sequence Diagram for Input Company/Employee Data Use Case



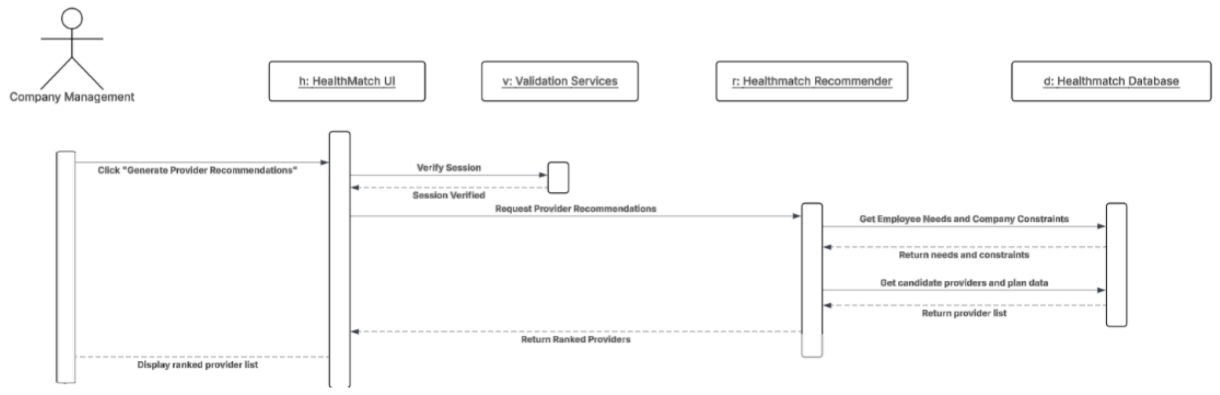
Sequence Diagram for Reviewing Available Plans



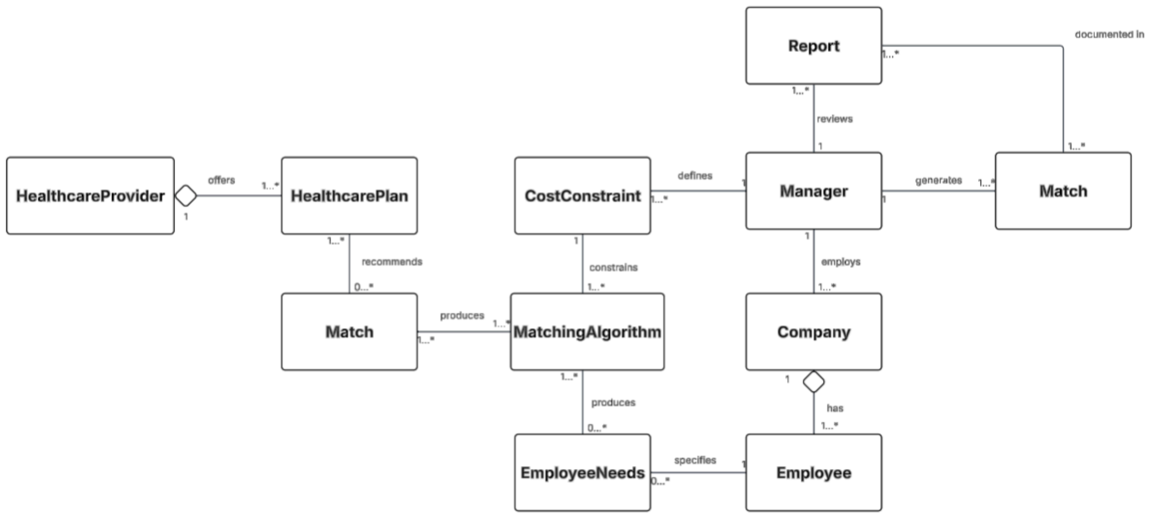
Sequence Diagram for Managing Provider Information



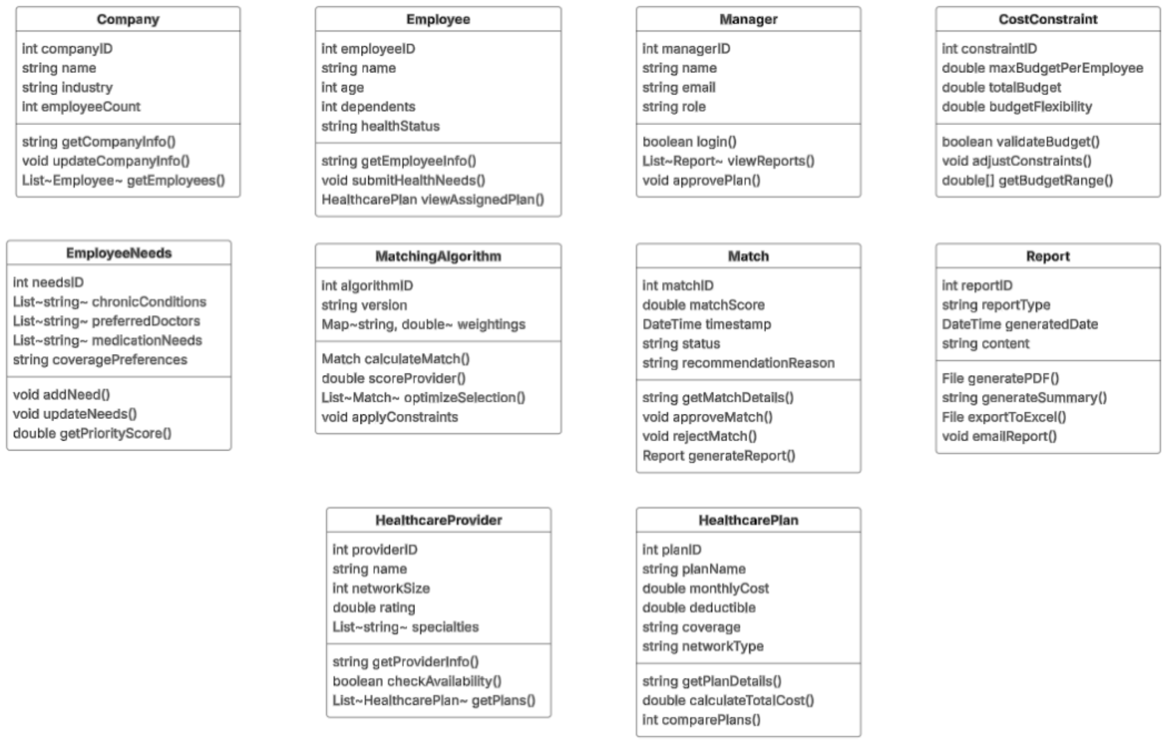
Sequence Diagram for Generating Provider Recommendations


Company Management
h: HealthMatch UI
v: Validation Services
r: Healthmatch Recommender
d: Healthmatch Database
Click "Generate Provider Recommendations"
Verify Session
Session Verified
Request Provider Recommendations
Get Employee Needs and Company Constraints
Return needs and constraints
Get candidate providers and plan data
Return provider list
Return Ranked Providers
Display ranked provider list

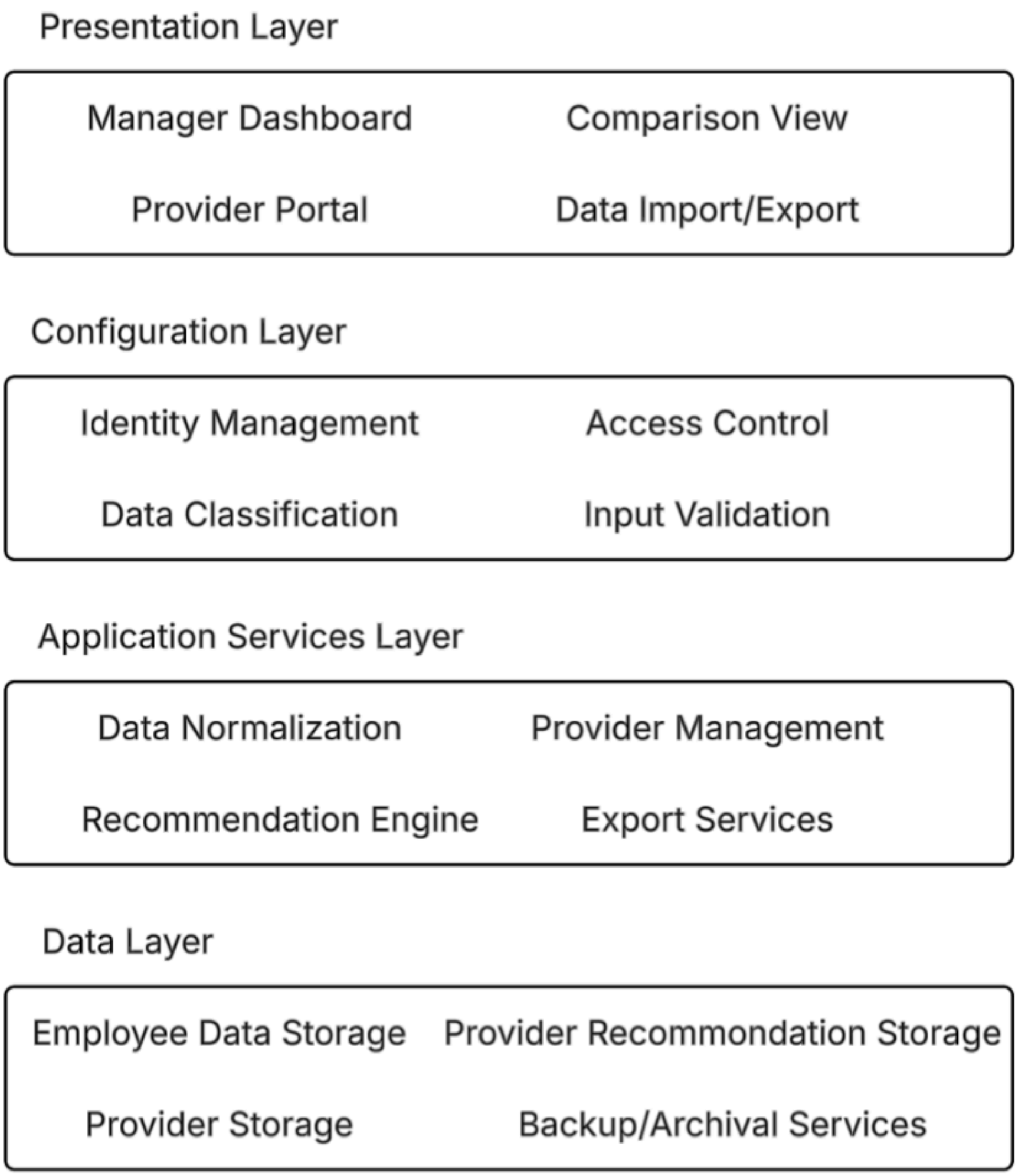
Class Diagram



Class Definitions



Architectural Design Diagram (Layered)



Project Scheduling

6-month Project Timeline (following the layered architecture)

The system shall be built according to an agile process model

January 15, 2026 - July 30th, 2026

- **Data Layer (6 weeks | Jan 15 - Feb 26): design schemas, databases, and data modeling**
- **App Services Layer (12 weeks | Feb 12 - May 7): Partially concurrent, data normalization, build CRUD operations, developing and testing.**
- **Configuration Layer (10 weeks | Mar 12 - May 21): Partially concurrent, authentication and security**
- **Presentation Layer (10 weeks | April 9 - Jun 18): Partially concurrent, front-end layers, dashboard, manager/comparison view**
- **Product Readiness (6 weeks | Jun 18 - Jul 30): Readying and integrating all aspects of the product, final discussions with stakeholders, building necessary advertisers etc.**
- **Length: January 16 - July 30th**
  - Standard timeline for complete and dedicated cross-functional group
  - Enough time to meet MVP
  - Concurrent development allows for fast approach
- **Work details:**
  - No weekends
  - Standard 8 hours/day 5 days/week

Cost Estimation

Estimation Method Used

- Used the **Function Point (FP)** method.

- FP measures software size by counting:
  Inputs (data entering the system)
  Outputs (reports, results)
  Data files (actual database management)
  User interactions (inputting employee/company data)

  These metrics fit better for our data-driven, interaction-based system as opposed to application composition, which is more UI centric [2].

Inputs

Function	Complexity	FP
Create EmployeeNeeds	Simple	3
Update EmployeeNeeds	Simple	3
Submit Provider Data	Medium	4
Input Company Info	Simple	3
Filter Providers	Simple	3

3 + 3 + 3 + 3 + 4 = 16 FP from Inputs

Outputs

Function	Complexity	FP
Provide Match Results	Medium	4
Generate Employee Summary	Simple	3
Generate Recommendation Rationale	Simple	3
Generate Admin Report	Simple	3

4 + 3 + 3 + 3 = 13 FP from Output

User Interaction

Function	Complexity	FP
View Employee Needs	Simple	3
View Provider Details	Simple	3

3 + 3 = 6 FP from User Interaction

File Interfacing

Function	Complexity	FP
Employee Needs Files	Medium	7
Provider Files	Medium	7
Match Results Files	Simple	4
Medical Specialty Files	Simple	3
Insurance Coverage Files	Simple	3

7 + 7 + 4 + 3 + 3 = 23 FP from File Interfacing

- HealthMatch totals **≈ 60 Function Points** (based on its inputs, outputs, and data complexity).

- Organization Rate: **10 FP ≈ 1 person-month**

- Required effort: **60 / 10 ≈ 6 person-months**

Labor Cost (FP Method)

- Estimated Size: **60 FP**
- Productivity Rate: **10 FP ≈ 1 PM**
- Required Effort: **6 person-months**
- Developer Rate: **$9,000 / month**
- **Labor Cost = $54,000**

Personnel Cost Summary

- **Total Personnel Cost: $54,000**

Hardware Cost: ~$450

- Covers cloud server + storage needed to host and develop HealthMatch
- No physical machines required since everything can run online

Software Cost: ~$270

- Mainly from managed database + analytics tools
- The software itself is planned to be developed using open-source, free software
- Most development tools (GitHub, VS Code, IntelliJ CE) are free

Labor Cost: $54,000

- Based on the FP estimate of 6 person-months
- Includes development, testing, and iteration time

Total Estimated Project Cost:

- **$54,730**

Quoted Price (+25% markup):

> **≈ $68,412 (Represents a realistic client-facing price including profit plus the overhead)**

Test Plan

For the purposes of this project, we are testing the validateEmployeeNeeds method. The EmployeeNeeds object contains several attributes:

- needsID: a unique identifier for each employee
- chronicConditions: a list of the employee's chronic conditions
- preferredDoctors: a list of preferred healthcare providers
- medicationNeeds: a list of medications the employee requires
- coveragePreferences: a string describing the employee's desired type of healthcare coverage

Each of these attributes must meet specific validity requirements:

- needsID must be an integer between 1000 and 9999.
- chronicConditions, preferredDoctors, and medicationNeeds must be non-null lists and must contain only string values.
- coveragePreferences must be a non-null, non-empty string and must contain alphabetical characters only (A–Z, a–z).

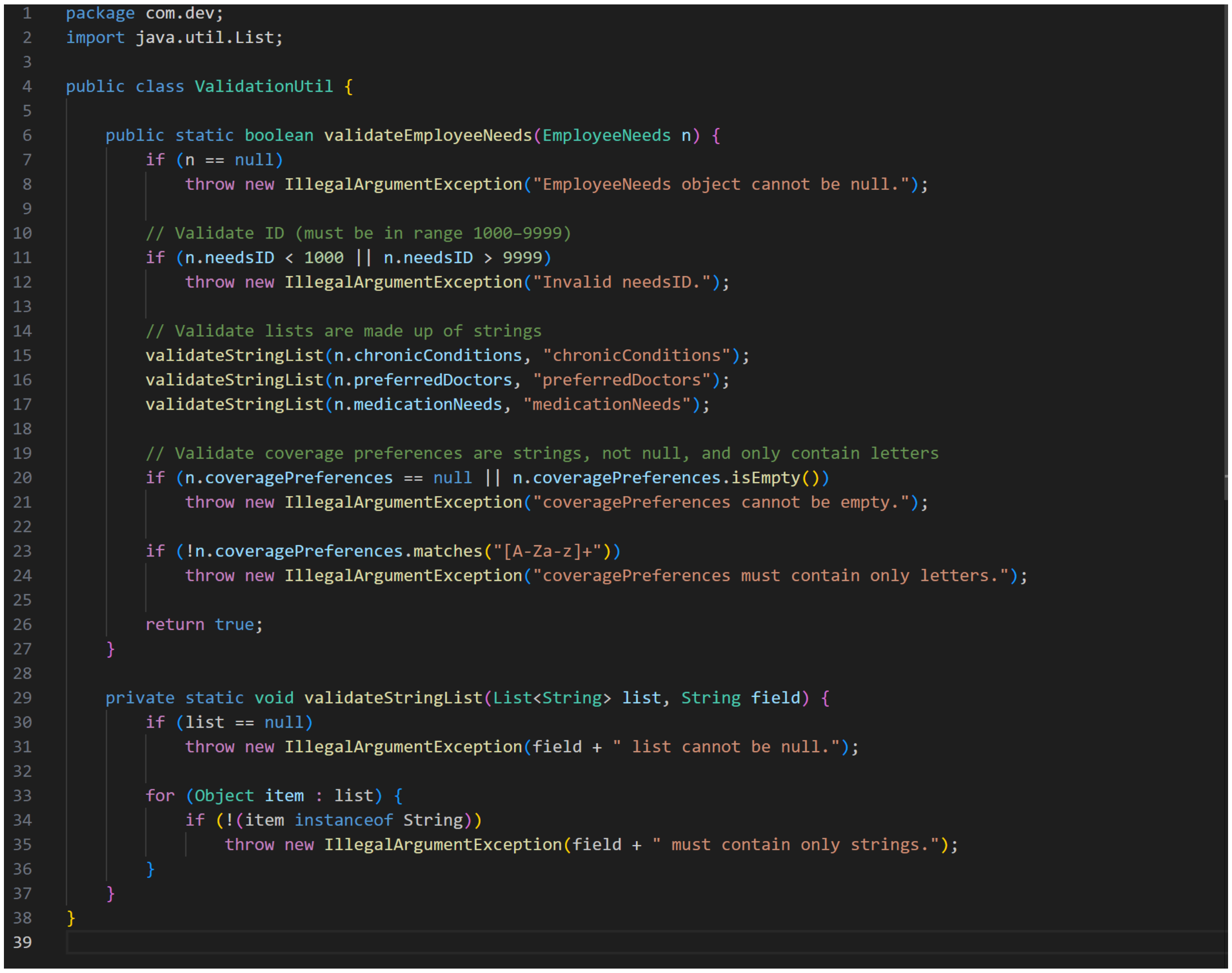
The purpose of the validateEmployeeNeeds method is to ensure that all attributes of an EmployeeNeeds object conform to these rules. If any attribute violates the expected constraints, the method should throw an appropriate IllegalArgumentException.

To verify the correctness of this method, five test cases were developed:

1. A valid EmployeeNeeds object
    - Ensures that the method returns true and does not throw an exception.
2. An invalid needsID below the required range
    - Verifies that the method throws an exception for improperly formatted IDs.
3. A null chronicConditions list
    - Confirms that null lists are detected and rejected.
4. A chronicConditions list containing a null value
    - Ensures the validator rejects lists that contain non-string elements.
5. A coveragePreferences value containing numbers
    - Confirms that non-alphabetical strings are considered invalid.

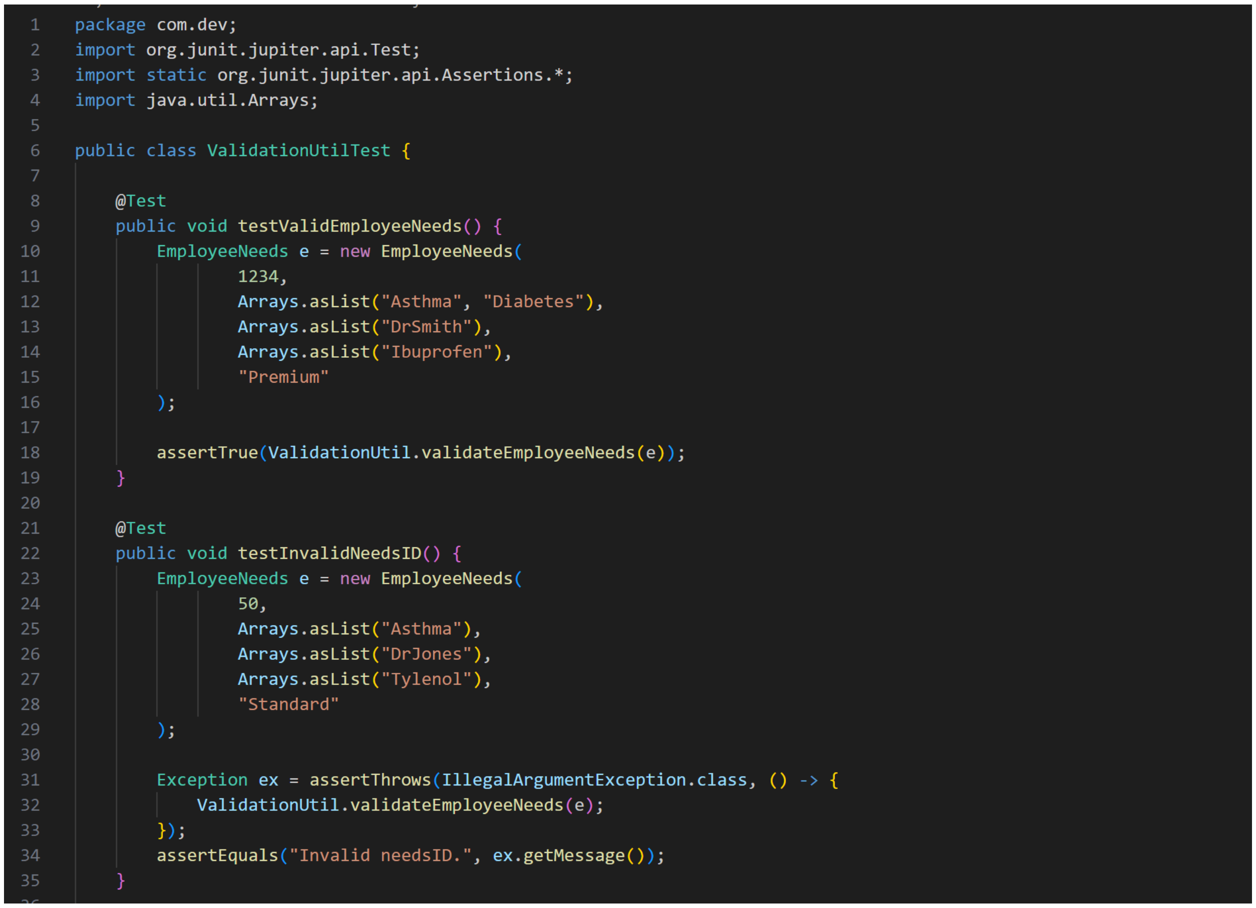
The first test case is intended to pass successfully through the validateEmployeeNeeds method without errors. The remaining four test cases are expected to trigger exceptions, demonstrating that the validation logic correctly identifies and rejects invalid inputs. Upon running the test cases, they pass.
The employeeNeeds class definition, validateEmployeeNeeds method definition, and unit test code are in the Github linked at the top.

validateEmployeeNeeds() Method Definition



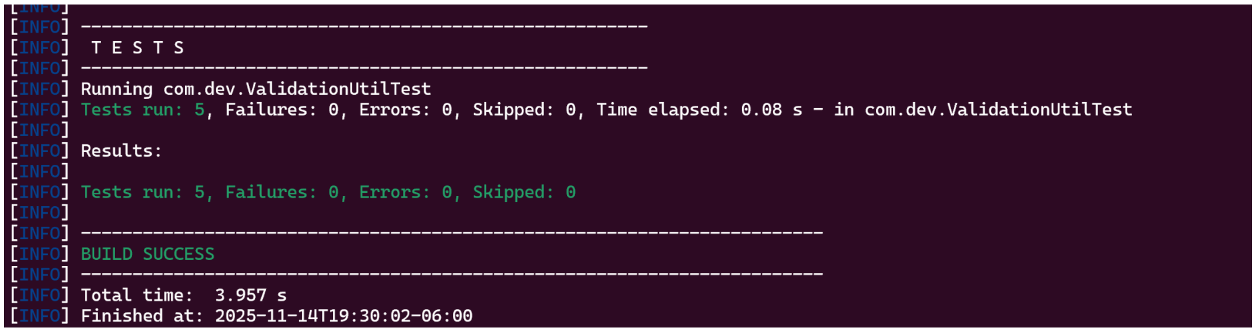
```java
package com.dev;
import java.util.List;

public class ValidationUtil {

    public static boolean validateEmployeeNeeds(EmployeeNeeds n) {
        if (n == null)
            throw new IllegalArgumentException("EmployeeNeeds object cannot be null.");

        // Validate ID (must be in range 1000-9999)
        if (n.needsID < 1000 || n.needsID > 9999)
            throw new IllegalArgumentException("Invalid needsID.");

        // Validate lists are made up of strings
        validateStringList(n.chronicConditions, "chronicConditions");
        validateStringList(n.preferredDoctors, "preferredDoctors");
        validateStringList(n.medicationNeeds, "medicationNeeds");

        // Validate coverage preferences are strings, not null, and only contain letters
        if (n.coveragePreferences == null || n.coveragePreferences.isEmpty())
            throw new IllegalArgumentException("coveragePreferences cannot be empty.");

        if (!n.coveragePreferences.matches("[A-Za-z]+"))
            throw new IllegalArgumentException("coveragePreferences must contain only letters.");

        return true;
    }

    private static void validateStringList(List<String> list, String field) {
        if (list == null)
            throw new IllegalArgumentException(field + " list cannot be null.");

        for (Object item : list) {
            if (!(item instanceof String))
                throw new IllegalArgumentException(field + " must contain only strings.");
        }
    }
}

```

Two Example Test Cases (The rest are in Github/Submission)



```java
package com.dev;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import java.util.Arrays;

public class ValidationUtilTest {

    @Test
    public void testValidEmployeeNeeds() {
        EmployeeNeeds e = new EmployeeNeeds(
                1234,
                Arrays.asList("Asthma", "Diabetes"),
                Arrays.asList("DrSmith"),
                Arrays.asList("Ibuprofen"),
                "Premium"
        );

        assertTrue(ValidationUtil.validateEmployeeNeeds(e));
    }

    @Test
    public void testInvalidNeedsID() {
        EmployeeNeeds e = new EmployeeNeeds(
                50,
                Arrays.asList("Asthma"),
                Arrays.asList("DrJones"),
                Arrays.asList("Tylenol"),
                "Standard"
        );

        Exception ex = assertThrows(IllegalArgumentException.class, () -> {
            ValidationUtil.validateEmployeeNeeds(e);
        });
        assertEquals("Invalid needsID.", ex.getMessage());
    }
```

Testing Results



```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.dev.ValidationUtilTest
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.08 s - in com.dev.ValidationUtilTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.957 s
[INFO] Finished at: 2025-11-14T19:30:02-06:00
```

Comparison with Similar Designs

We will be comparing HealthMatch to two widely used healthcare platforms, Zocdoc and Doctolib.

Both platforms provide access to healthcare providers, but both focus on primary or individual patient scheduling. Where as for our system HealthMatch is designed for company management and HR teams to help make data driven decisions when choosing a healthcare provider for employees.

Zocdoc is a US digital based platform that allows patients to search for doctors, view ratings, and book medical appointments online. Through reviews, insurance filters, and real time scheduling [4].

Doctolib is a European health-tech platform that offers similar services. It uses online booking, patient management, and telemedicine which is a remove care over video chat for visits that don’t need to be hands on [5].

Feature / Aspect	**Zocdoc**	**Doctolib**	**HealthMatch**
Target Users	Individual patients	Individual patients & clinics	Company management / HR teams
Primary Function	Doctor search & appointment booking	Appointment booking & telemedicine	Provider matching & recommendation engine
Scope	Patient-facing	Patient- and provider-facing	Management-facing (enterprise level)
Data Inputs	Patient location, insurance, specialty	Patient info, appointment type	Employee preferences, budget, provider data
Decision Support	Limited (user choice)	Limited	Automated ranking algorithm
Goal	Simplify booking for individuals	Improve healthcare scheduling	Optimize provider selection for companies

HealthMatch differs from both Zocdoc and Doctolib as it shifts the focus from individual patient scheduling to organizational healthcare decision-making. Instead of connecting patients to doctors HealthMatch would allow companies to evaluate different healthcare providers based on employee preferences. Its intelligent matching algorithm would rank providers objectively which would help HR teams in making the best choice

Conclusion

HealthMatch's design phase has resulted in a detailed and verifiable blueprint for development. HealthMatch is a software system that connects companies with the most suitable healthcare providers by analyzing employee needs, preferences, and coverage requirements**.** Because the system processes a wide range of structured inputs and relies on numerous internal data files, we narrowed our estimation approach to the function point method, which best reflected the scope and data-centric nature of the project.

We also progressively tested our UML artifacts—specifically our sequence diagrams—to ensure they accurately modeled object lifetimes, responsibilities, and activation states across the system's core interactions. This iterative validation helped confirm that our design met the strict technical standards necessary before beginning implementation.

The outcome of this process was a comprehensive 6-month project plan that is both technically realistic and financially viable, laying a strong foundation for the successful development of HealthMatch.

References

[1] Agile Alliance, “Manifesto for Agile Software Development,” 2001. [Online]. Available: https://agilemanifesto.org/

[2] International Function Point Users Group (IFPUG), “Function Point Analysis (FPA),” IFPUG Standards. [Online]. Available: https://ifpug.org/ifpug-standards/fpa

[3] U.S. Department of Health and Human Services, “Summary of the HIPAA Privacy Rule,” Mar. 14, 2025. [Online]. Available: https://www.hhs.gov/hipaa/for-professionals/privacy/laws-regulations/

[4] Zocdoc, “Find a Doctor Near You | Book Doctors Online,” Zocdoc, 2025. [Online]. Available: https://www.zocdoc.com/

[5] Doctolib, “Reinventing Healthcare,” Doctolib, 2025. [Online]. Available: https://about.doctolib.com/