Simplify : An AI-powered Skincare Web App

By

Hein Zaw, Minji Kim, Dowon Kim

hein.zaw@stonybrook.edu
dowon.kim@stonybrook.edu
minji.kim.6@stonybrook.edu

Department of Computer Science, SUNY Korea

CSE 416: Software Development

Professor Yoonseok Yang

April 18, 2025

Targeted Audience: Skincare novices, beauty enthusiasts, busy professionals, individuals with sensitive skin, and tech-savvy consumers seeking personalized, AI-powered skincare solutions

TABLE OF CONTENTS

1. Introduction

1.1 Product Description

Skincare is very important at any time for everyone who wants to enhance one's lifestyle. Although it seems easy to just follow the basics of skin care, beauty products are enhanced to the point where each product is curated for different skin types which would not help if used on other skin types. However, people who are unaware of what to do or time-constrained by their busy schedules can not carefully construct and plan a skincare routine that is healthy for one's skin.

Our solution is to create an application that can assist people with their skincare-related problems. We will use AI image-recognition technologies to analyze each user's skin and recommend proper self-care routines. The user can also ask questions about any skincare-related topics and receive trustworthy information from the AI chatbot. With these features, users would be able to get fast and reliable guidance for each individual's needs.

Simplify is an AI-powered self-care assistant designed to streamline skincare routines for everyone. Our solution integrates **AI-driven product recommendations, an AI Skin Lab for personalized analysis, and gamification** to enhance engagement by encouraging consistency through competition and rewards. The project also explores a **win-win sponsorship model**, enabling brands to better connect with consumers in the self-care space.

1.2 Scope

The product will assist people in skincare that fits each user's skin type. The product would provide skin analysis, routines made from the information of their skin, and a streak system that could encourage the user's consistency. It will serve people who are new to skincare, people who want to take care of their skin better but do not know where to start, and people who do not have time to go through the details of skincare but want a better routine. This product would be generalizable as even if a user is knowledgeable in skincare, this would help the user to maintain their routine and assistance so that they do not have to worry about the minimal tasks of a routine. The product would run on both mobile and on the web for more accessibility for the users.

1.3 Users

Novices in skin care who have little to no knowledge about beauty but want to start changing their lifestyle would be our main target audience. However, the audience extends to beauty experts who want data-driven solutions to enhance their skin. They would need basic technical expertise such as how to take a selfie, how to use the internet, and how to navigate through a website. The ages could range from 15~60 and there is no constraint on gender, country, ethnicity, etc.

- **Novices in Skincare** – Little to no knowledge about beauty routines, seeking a simple and guided way to start.
- **Beauty Enthusiasts & Experts** – Familiar with skincare but looking for **data-driven insights** to refine their routine.
- **Varied Technical Expertise** – Includes both tech-savvy users and those less familiar with digital tools.
- **Accessibility Considerations** – Features like **adjustable text sizes, and clear navigation** for inclusivity.

1.4 User Feedback

User feedback is the most important when gathering requirements and designing our web app. We will gather this by using methods such as asking casually in-person or handing out surveys. As a summary of some of the feedback we were given, the most important part was to provide more personalized information to the users such as using the details of their routine and how many streaks they have, the application can give recommendations on what other things the user could do. Also, another feedback was showing information actively to the user by their previous data instead of making the user have to search for all the information. From the feedback, more personalization and active information given to the user should be taken into account as every user would deem such features convenient.

1.5 Existing Alternatives

We have found several AI dermatology services online, such as ai-derm.com, autoderm.ai, cetaphil.com, and more. These services are able to detect various skin diseases by using a sizable training dataset, but fall behind in more generalized skin care problems and solutions. They also lack personalization, something we plan to make our priority.

1.6 Definitions

Dermatology : A branch of medicine focused on diagnosing and treating skin conditions, offering expert care and guidance for skin health.

SkinLab : An AI-powered skin analysis tool within the app that assesses skin type and condition, providing personalized skincare recommendations.

Simpli : An AI-driven self-care assistant designed to simplify and optimize skincare routines, offering personalized recommendations, real-time skin analysis, and engaging features like gamification.

1.7 References

1. *Ai dermatologist: Skin Scanner*. AI dermatologist: Skin scanner. (n.d.). https://ai-derm.com/

2. Börve, A. (2025, March 15). *Skin image searchTM - dermatology artificial intelligence (AI)*. First Derm. https://www.firstderm.com/ai-dermatology/

3. *The 5 signs of skin sensitivity*. Recommended Skincare Brand for Sensitive Skin | Cetaphil US. (n.d.). https://www.cetaphil.com/us/

2. Requirements

2.1 Functional Requirements

1. Users must be able to receive a personalized visual skin analysis
2. Users must be able to keep track of the process by using the checklist
3. Users must be able to get AI-driven product recommendations
4. Users must be able to ask skin care related questions for instant guidance
5. Users should be able to get rewards with consistent usage

2.2 Use Cases

Use Case 1	Skin Lab Photo Analysis
Primary Actor	User
Priority	Essential
Scenario	1. The user logs into the application. 2. The user clicks on the skin lab tab on the sidebar. 3. The skin lab page should show default information of how the analysis will come out with a button to start the analysis. 4. The user clicks on the start button and asks the user for device camera access. 5. Skin lab page shows a web camera view for the user to take a photo along with buttons to take the photo and upload a photo. 6. The user clicks on the take photo button and the web camera view will turn to a still picture the user just took. 7. The user clicks on the submit button and the skin lab page will display the analysis of the user's photo.
Extensions	1a) Password is not correct 1a 1) Show error and back to sign up page 5a) User denies camera access 5a 1) The skin lab page blacks out the photo frame and waits for the user's permission

Use Case 2	Inquire about the AI assistant about skin care related problems
Primary Actor	User
Priority	Essential
Scenario	1. The user logs into the application. 2. The user clicks on the Chat tab in the sidebar. 3. The user types their skin care question in the input field.

	4. The user sends the message. 5. The system replies with the appropriate answer.
Extensions	1a) Password is not correct 1a 1) Show error and back to sign up page

Use Case 3	Check the rank on the leaderboard
Primary Actor	User
Priority	Essential
Scenario	1. The user logs into the application. 2. The user navigates to the leaderboard section from the sidebar. 3. The system displays the leaderboard showing ranks of all users based on their skincare routine consistency. 4. The user locates their rank, which is highlighted or listed with their username and points.
Extensions	1a) Password is not correct 1a 1) Show error and back to sign up page

Use Case 4	Finish the Routine
Primary Actor	User
Priority	High
Scenario	1. The user logs into the application and navigates to the routine page. 2. The user sees a list of tasks in their skincare routine for the day. 3. The user completes each task in the routine (e.g., applying cleanser, moisturizer, etc.). 4. The user checks the checkbox next to each completed task in the routine. 5. Once all tasks are completed, the user checks the final checkbox to mark the routine as finished.

	6. The system records the completion and adds to the user’s streak count and points. 7. The system displays a message confirming the streak and points earned.
Extensions	1a) Password is not correct 1a 1) Show error and back to sign up page 5a) User has already completed the routine but tries to mark it again 5a 1) The system will display an error message and remind them that the routine has already been finished for the day.

Use Case 5	Delete the user’s data
Primary Actor	Admin
Priority	Essential
Scenario	1. The admin logs into the system with proper credentials. 2. The admin navigates to the user management section. 3. The admin searches for the user by username, and email. 4. The admin selects the user whose data needs to be deleted. 5. The admin confirms the deletion request. 6. Upon confirmation, the system securely deletes the user's personal data, including skincare preferences, recommendations, and any stored images or analysis.
Extensions	6a. If a server error occurs during the data deletion , the system will display an error message and prompt the admin to try again.

a

2.3 Non-functional Requirements

1. Security:

- User data (e.g., skin analysis results) must be encrypted using HTTPS.
- Basic authentication (email/password) for user login.
- 2FA with Google OAuth

2. Usability:

- The app should be intuitive, with simple navigation requiring minimal guidance (less than 10 minutes for a new user to get started).
- Clear error messages for invalid input or incomplete steps.

3. Compatibility:

- The app should work on the latest versions of Chrome and Firefox.
- It should be responsive for screens from phones, tablets and desktops.

4. Maintainability:

- The code should be modular and easy to update (less than 50 lines per function).
- Use basic automated tests to check app functionality (e.g., test login and skin analysis features).

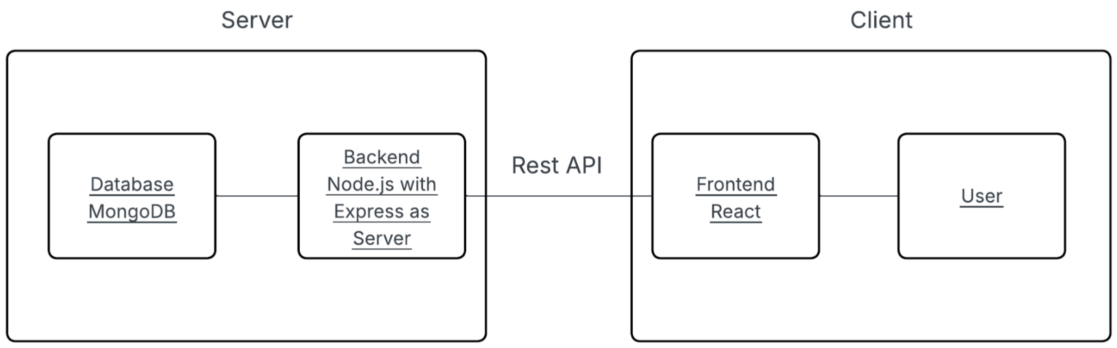
5. Performance:

- Real-time AI processing within 15 seconds

3. System Architecture

3.1 Overview

We are using a client-server architecture which is a distributed application structure that partitions tasks or workloads. With the MERN stack that uses MongoDB, Express.js, React, and Node.js. On the client side of the architecture, there will be the user and the frontend which will be React. The server side of the architecture will be the Server with Express.js and Node.js and the Database with MongoDB. For the connection of the server side and the client side, we will use REST APIs.



Since we are using the MERN stack we are primarily using Javascript-based code on both frontend and backend. We also additionally used CSS to ensure consistent styling across the application and to enhance the modularity of components, making them easier to manage and reuse.

Other than the above we are using some more third-party libraries and frameworks.
<u>Frontend</u>
For the frontend we are using Vite which is a helpful and fast development tool optimized for JavaScript frameworks like React. For consistency of design for the UI, we are using Chakra UI.

Major versions

- React: V19
- Vite: V6
- ChakraUI: V2

<u>Backend and Database</u>
Backend server is just Express.js and Node.js. Our database will be mainly MongoDB for storing user credentials and other persistent application data. In addition to MongoDB we will use Supabase which is an open-source used to store vectorized or CSV-based web scraped data.

Major versions

- Express.js: V4
- MongoDB: V8

Authentication and Others

For authentication we are also planning to use Google OAuth which allows secure and token-based sign in through Google accounts. Regarding deployment will be looked into in section 3.5 of this document.

Major versions

- Google OAuth: V2

Reason for this technology stack

We chose this technology stack as the MERN stack is widely recognized as an industry-standard stack for modern web development. It also offers a unified Javascript environment across both frontend and backend simplifying the development process. Additionally, the team has prior experience with this stack through previous courses making it easier for all our team members to easily follow on productively.

3.2 Data Design

UML Data Diagram contains classes User, Product, Feedback, Forum, Chat, and Leaderboard.
User has referenced attributes to a Feedback, an array of Products, an array of Chats, and a Forum.
LeaderBoard references an array of Users.

LucidChart Data Diagram

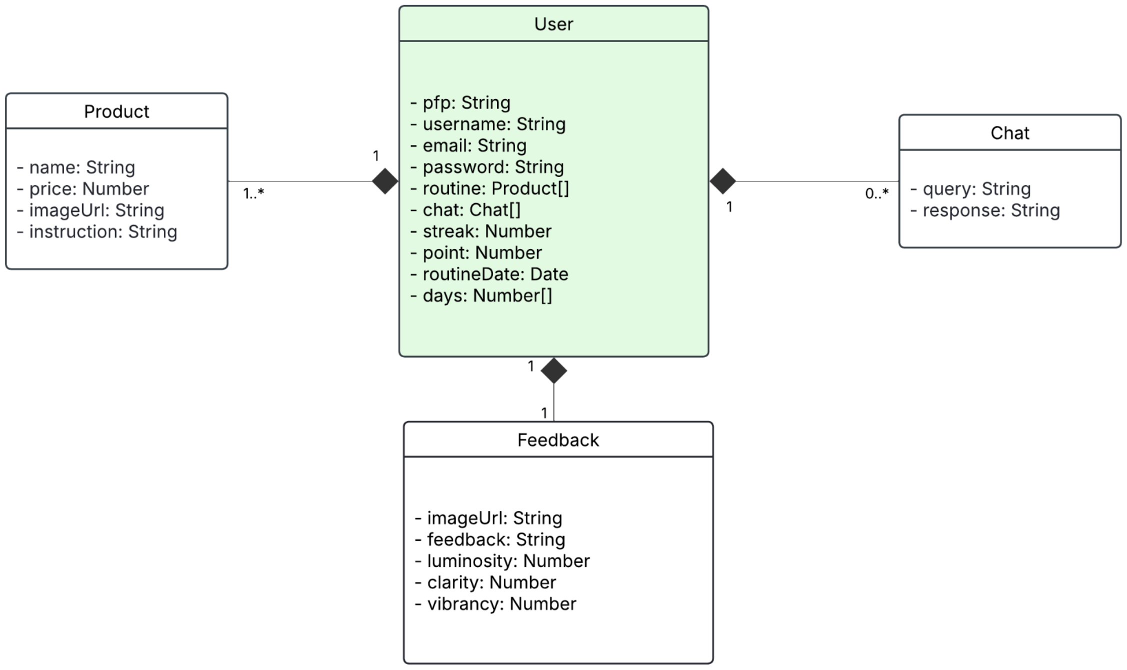
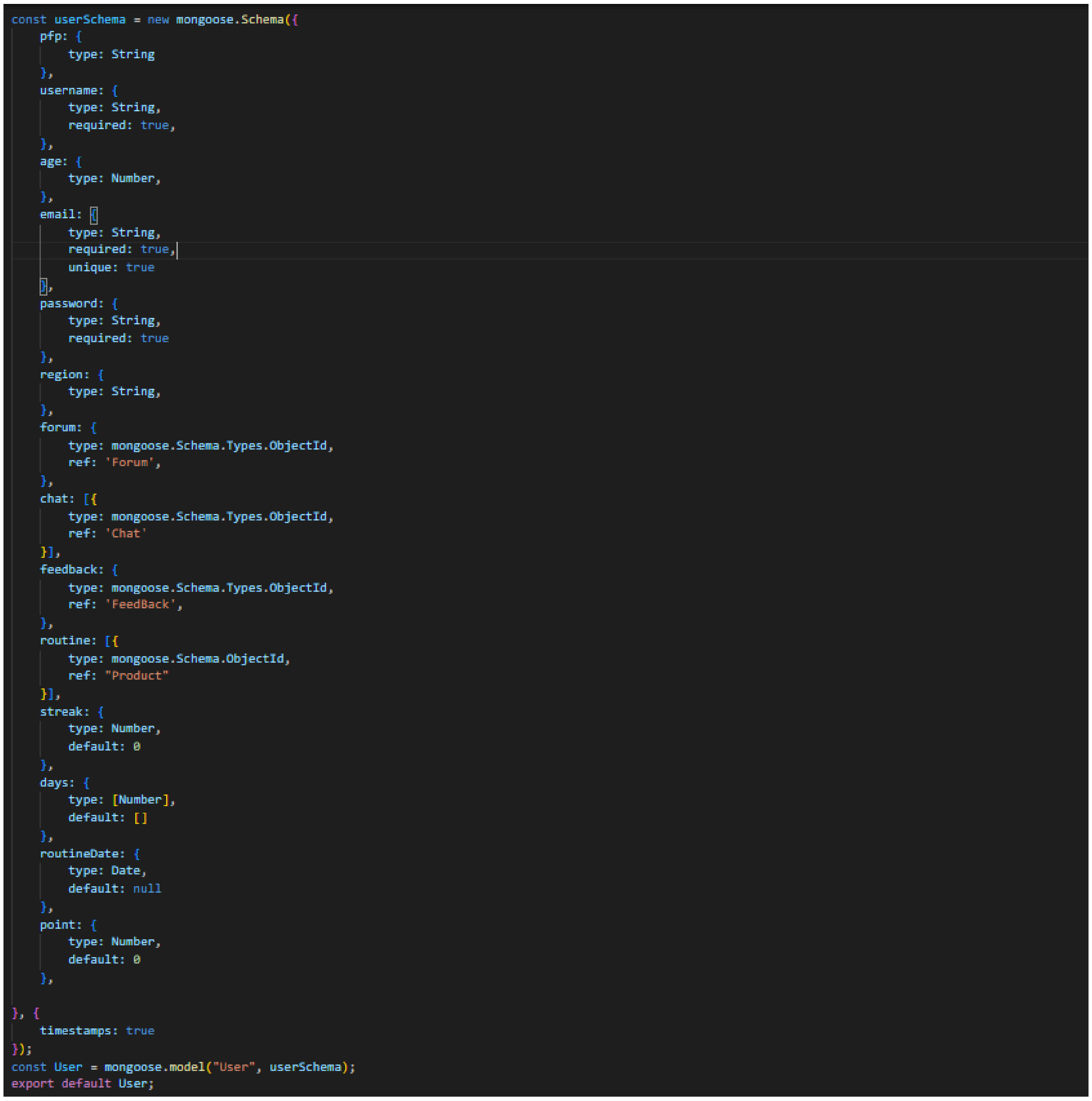


Fig 1: UML Data Diagram

MongoDB Data Diagram

We used ODM(Object Document Mapping) to map our classes to the database using MongoDB schemas

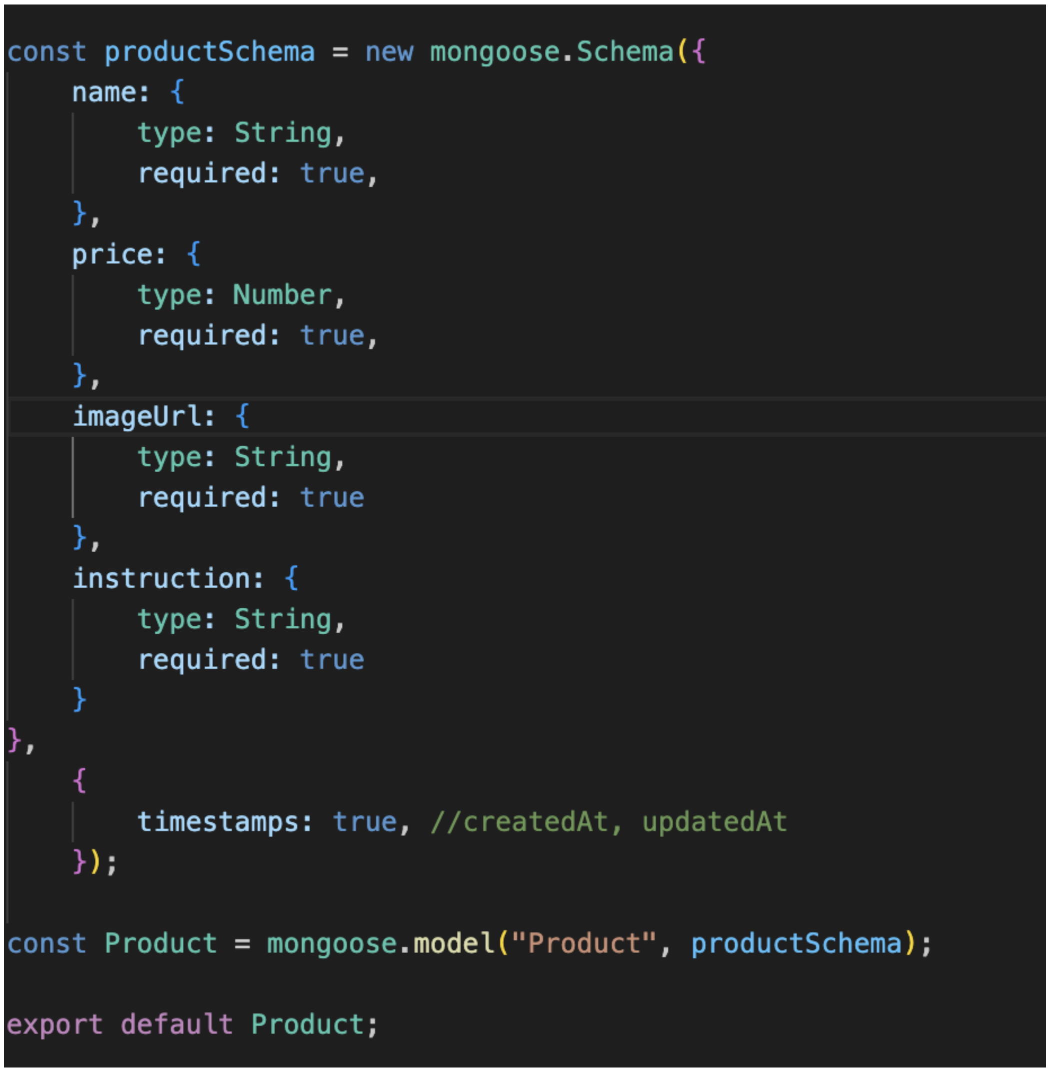


```
const userSchema = new mongoose.Schema({
    pfp: {
        type: String
    },
    username: {
        type: String,
        required: true,
    },
    age: {
        type: Number,
    },
    email: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String,
        required: true
    },
    region: {
        type: String,
    },
    forum: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Forum',
    },
    chat: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Chat'
    }],
    feedback: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'FeedBack',
    },
    routine: [{
        type: mongoose.Schema.ObjectId,
        ref: "Product"
    }],
    streak: {
        type: Number,
        default: 0
    },
    days: {
        type: [Number],
        default: []
    },
    routineDate: {
        type: Date,
        default: null
    },
    point: {
        type: Number,
        default: 0
    },

}, {
    timestamps: true
});
const User = mongoose.model("User", userSchema);
export default User;
```

Fig 2: user Schema

The user schema contains information about the user such as pfp(profile picture), username, email, and streak. As MongoDB is not designed to store large binary files we decided to use URLs for images to keep our database lightweight and easier to manage. Point and streak will be used for our leaderboard(gamification). Email and password are related to user authentication. User schema also references chat, feedback, and product classes.



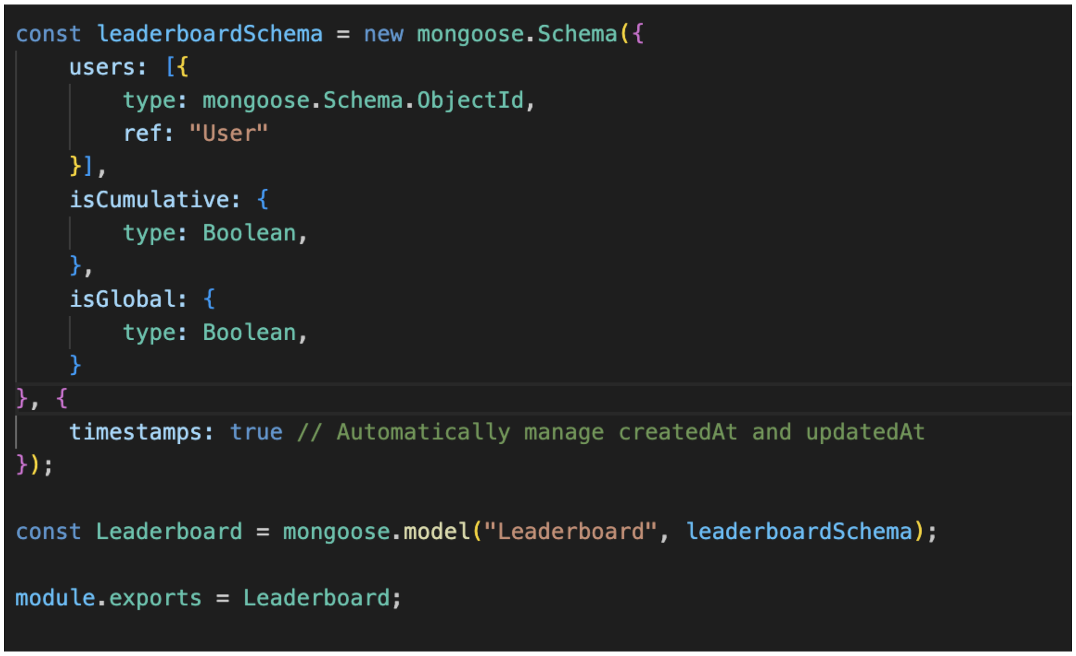
```
const productSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
    },
    price: {
        type: Number,
        required: true,
    },
    imageUrl: {
        type: String,
        required: true
    },
    instruction: {
        type: String,
        required: true
    }
},
    {
        timestamps: true, //createdAt, updatedAt
    });

const Product = mongoose.model("Product", productSchema);

export default Product;
```

Fig 3: Product Schema

Product schema contains the product name, price, and imageURL for the information of products. Also each product has an instruction on how to use it for a user’s routine.
The timestamps option is enabled to automatically store createdAt and updatedAtvalues for each leaderboard document.



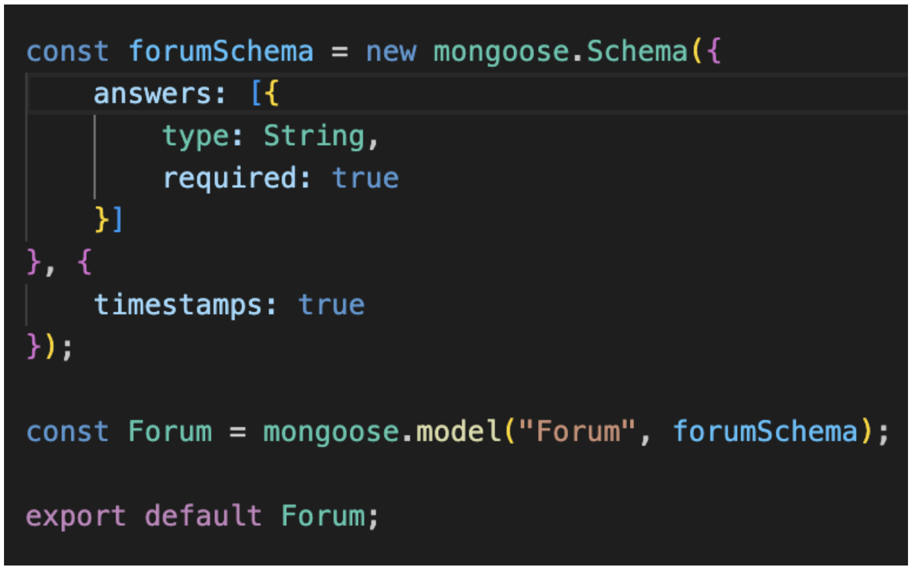
```
const leaderboardSchema = new mongoose.Schema({
    users: [{
        type: mongoose.Schema.ObjectId,
        ref: "User"
    }],
    isCumulative: {
        type: Boolean,
    },
    isGlobal: {
        type: Boolean,
    }
}, {
    timestamps: true // Automatically manage createdAt and updatedAt
});

const Leaderboard = mongoose.model("Leaderboard", leaderboardSchema);

module.exports = Leaderboard;
```

Fig 4: Leaderboard Schema

Leaderboard schema contains a list of referenced users and two boolean type attributes being isCumulative and isGlobal. The two boolean values are for the filtering of the leaderboard. Depending on the filters, it will show different users within the leaderboard.



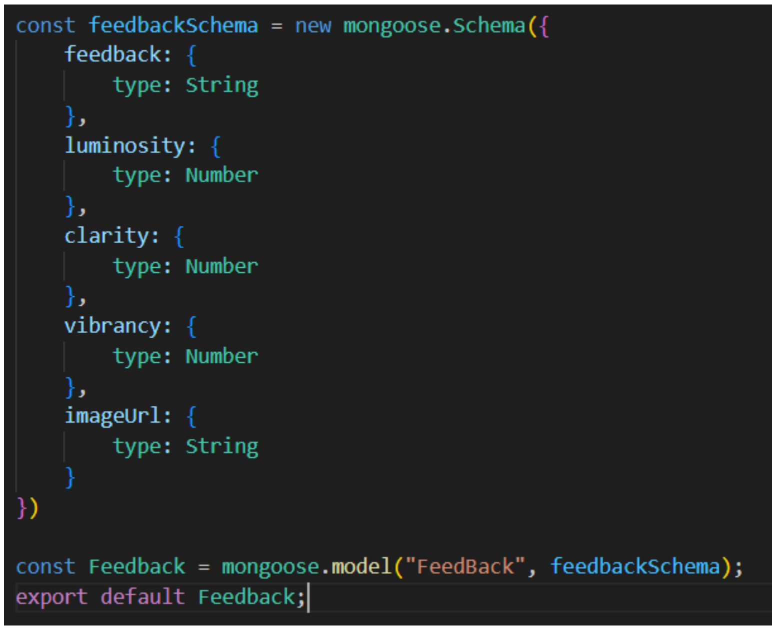
```
const forumSchema = new mongoose.Schema({
    answers: [{
        type: String,
        required: true
    }]
}, {
    timestamps: true
});

const Forum = mongoose.model("Forum", forumSchema);

export default Forum;
```

Fig 5: Forum Schema

Forum schema has an array of answers which would be the user's selected answers from the survey the user goes through while signing up.

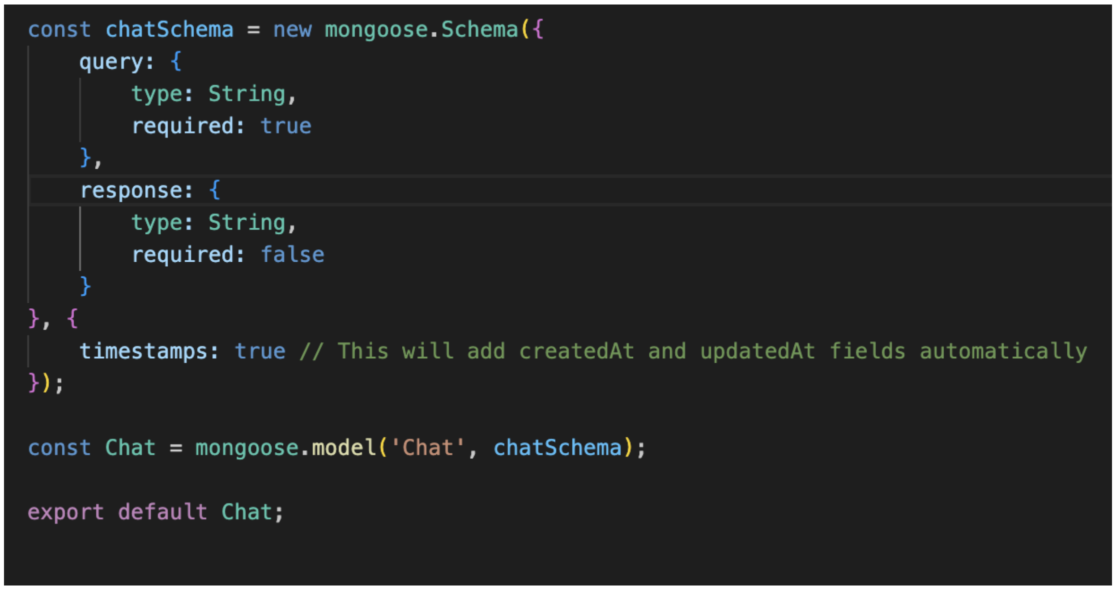


```
const feedbackSchema = new mongoose.Schema({
    feedback: {
        type: String
    },
    luminosity: {
        type: Number
    },
    clarity: {
        type: Number
    },
    vibrancy: {
        type: Number
    },
    imageUrl: {
        type: String
    }
})

const Feedback = mongoose.model("FeedBack", feedbackSchema);
export default Feedback;
```

Fig 4: Feedback Schema

Feedback schema is for storing the information of the Skin Labs output. It stores the text feedback luminosity, clarity, vibrancy, and skinUrl. Luminosity, clarity, and vibrancy are the metrics for the skin analysis and skinUrl is for the user's input image for skin analysis.



```
const chatSchema = new mongoose.Schema({
    query: {
        type: String,
        required: true
    },
    response: {
        type: String,
        required: false
    }
}, {
    timestamps: true // This will add createdAt and updatedAt fields automatically
});

const Chat = mongoose.model('Chat', chatSchema);

export default Chat;
```

Fig 5: Chat Schema

Chat schema is for storing each query and response for each user's query. By saving chats in an array, we will save the user chat history.

3.3 UML Sequence Diagram

Use Case #1:
User Signup with Survey Questions

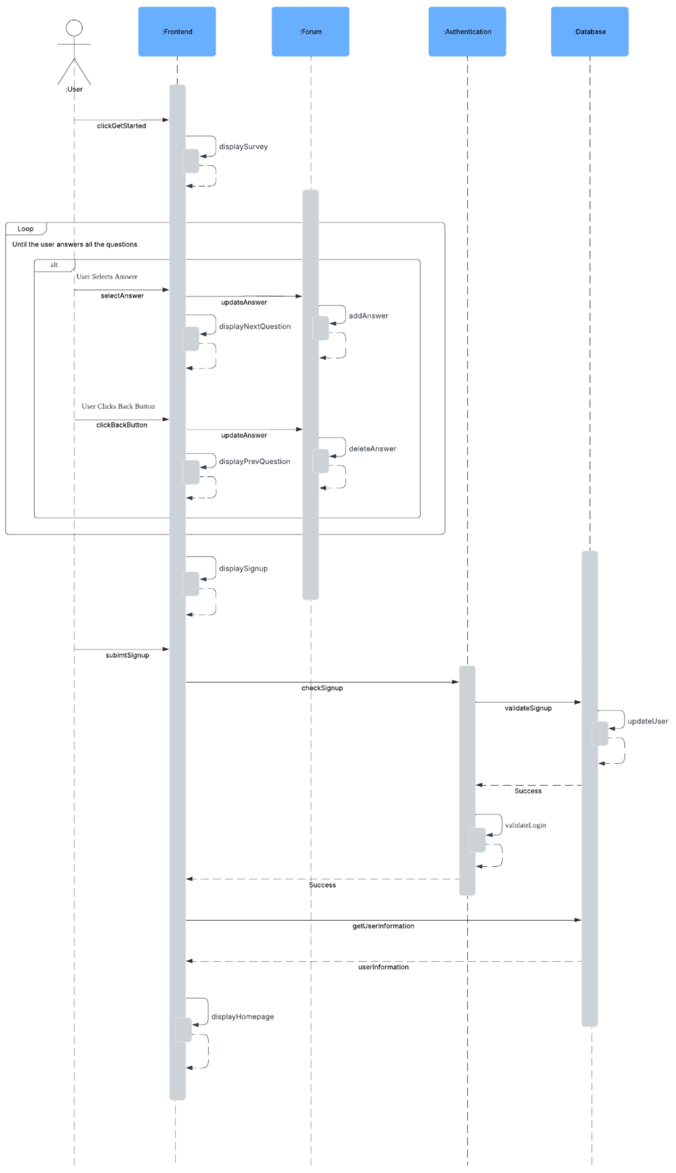


Fig 6: Sequence Diagram for Signup

Use Case #2:
Finishing a routine

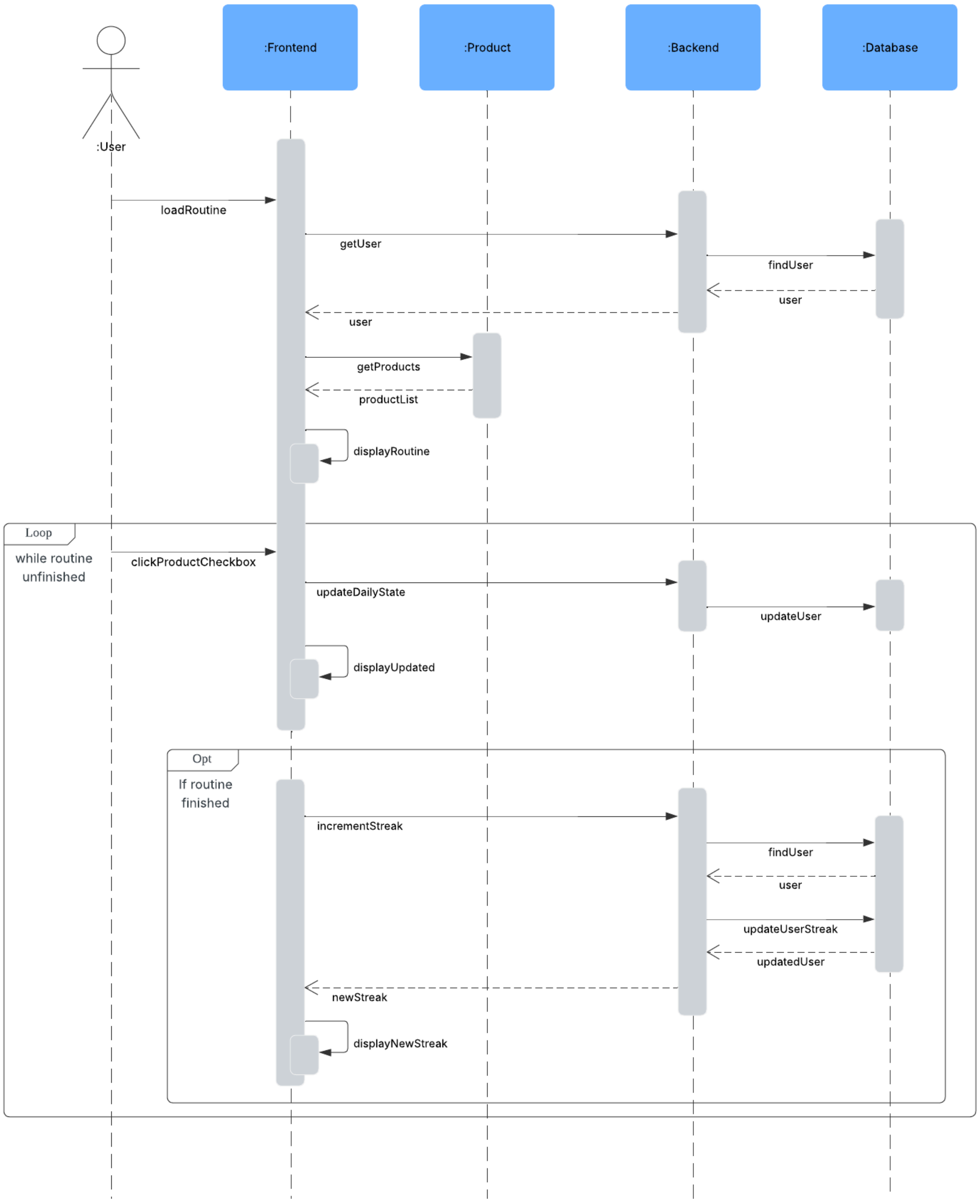


Fig 7: Sequence Diagram for finishing a routine

3.4 API Design

We've put all our API details in the spreadsheet linked below. We designed RESTful APIs by following the API design practices from Microsoft.

Google Sheet link: Simplify - API Design

3.5 Deployment

For deploying our web application we will use,

- **Render** (for Backend/API hosting):
  Render has a **free tier** that allows you to host web services with **auto-sleep after 15 minutes of inactivity**. It is very accessible and with its free tier and auto-sleep feature after 15 minutes of inactivity, it provides a cost-effective solution for long-term use.

- **Github Pages** (for Frontend hosting):
  We will use **GitHub Pages** to host our frontend application. It is a free and easy way to deploy static websites directly from a GitHub repository. Since our frontend is built using React, GitHub Pages offers seamless integration with GitHub, allowing us to deploy and update the website simply by pushing changes to the main branch. It also provides free HTTPS support and allows us to use a custom domain, making it a reliable and straightforward choice for hosting the frontend of our web application.

- **Supabase** (for Retrieval Augmented Generation data):
  We are using Supabase to store and manage data needed for Retrieval Augmented Generation (RAG). It provides an easy-to-use database and API system, so we can quickly retrieve the necessary data for our AI tasks. It's fast, open-source, and offers real-time syncing, making it a solid choice for our data needs.

- **MongoDB** (for User data)
  MongoDB is a flexible NoSQL database that stores user data in a way that's easy to manage and scale. It allows us to store user information, like profiles or activities, in a document-based format. It's simple to work with and integrates well with our backend, so we can easily store and access user data.

*We will use **AWS free credits from Github student developer pack** only for the final project submission until the grading period to be cost-effective!*

3.6 Code Conventions

For code conventions, since we are primarily using Javascript on both frontend and backend, we decided to follow the Google JavaScript Style Guide. Also as we have css to support our styling, we decided to follow the Google HTML/CSS Style Guide.

For API design conventions, we used Microsoft's API design practices.

4. Schedule

Milestone 1 4/23
- API design + Early implementation (Dowon)
- Google OAuth login (Dowon)
- Web Scraping (Henry)
- Frontend Pages (Minji)
- Leaderboard Scoring Logic (Minji)

Milestone 2 4/30
- Langchain API testing (Henry)
- Chatbot implementation (Henry & Dowon)
- User daily streak system (Dowon)
- Product recommendation (Henry)
- Admin page (Minji)
- Skincare-related API prompt design (Minji)
- Frontend State management / Integration with Backend (Minji)

Milestone 3 5/12
- Dall-E vision API for Skin Lab (Henry & Minji)
- Frontend Skinlab page (Henry & Dowon)
- Leaderboard system (Minji)
- Deployment (Henry)
- User feedback (Everyone)
- Testing (Everyone)

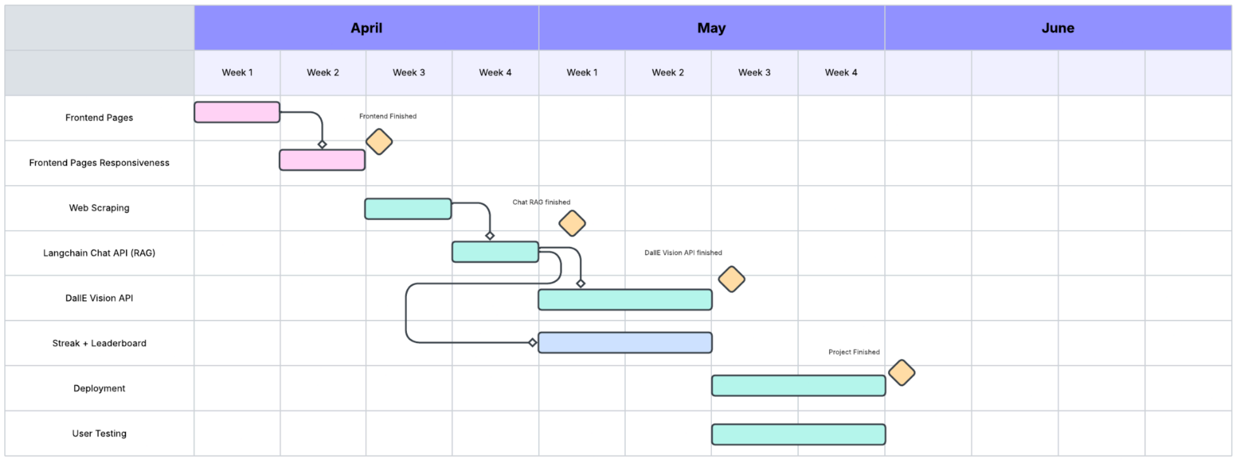


Fig 8: Gantt Chart

5. Contributions

<u>Hein Zaw / Henry</u>

- Gantt Chart
- Web Scraping
- Website Responsiveness
- Welcome Page Setup
- Supabase vector embedding storage

<u>Minji Kim</u>

- Sequence Diagram for Signup
- Software Architecture Chart
- Presentation Preparation
- Colors and Font Utils
- User Page Setup

<u>Dowon Kim</u>

- Sequence Diagram for Routine Completion
- Authentication and error logs
- Frontend responsiveness
- Frontend CSS styling
- API Design