Hello, welcome to the demo of **HealthyAura**, a full-stack web application designed to help users discover affordable and healthy eateries around Singapore. It's built with **React** for the frontend and **Spring Boot** for the backend, using **JWT-based authentication**, personalized recommendation logic, and role-based access for both users and admins.

From a software engineering perspective, HealthyAura applies several key **design and architectural patterns** to ensure scalability, maintainability, and security.
On the backend, we implemented the **Model–View–Controller (MVC)** architecture to clearly separate concerns — the controllers handle API requests, the services contain business logic, and the repositories manage database access. This **separation of concerns** makes the system easier to maintain, test, and extend.

We also applied the **Dependency Injection** principle through Spring's `@Autowired` annotations, allowing loose coupling between classes and improving testability.
The **Repository pattern** is used with JPA interfaces to abstract data access from business logic, while **DTOs (Data Transfer Objects)** ensure a clean separation between backend entities and exposed API responses.
The **Strategy pattern** is used in our **recommendation algorithm**, which allows the system to dynamically switch between different recommendation strategies — for example, filtering based on user preferences, distance, or health ratings — without changing the main logic.

In terms of **security principles**, we applied **role-based authorization** so that only admins can access management endpoints, and **JWT authentication** ensures that user sessions remain stateless and secure. Passwords are encrypted using **bcrypt hashing**, and all sensitive actions — such as profile updates and admin creation — are validated server-side to prevent unauthorized access or data tampering.

On the frontend, React follows a **component-based structure** that supports modularity and reusability. Pages like *Home*, *Explore*, and *Profile* communicate with the backend via a centralized API service, enforcing a consistent separation between presentation and data handling layers.

Now, let's begin the demo. I'll start by creating a new user account. After signing up, I'll log in using the same credentials. Once logged in, we arrive on the **Home page**, which immediately displays two types of recommendations — *personalized suggestions* based on the user's stored preferences, and *nearby eateries* determined through geolocation.

The recommendation algorithm matches user preferences like *Halal* or *Vegetarian* with eateries tagged under similar categories, factoring in average health and hygiene ratings. Since this new user currently has no preferences, the recommendations shown are quite general. Let's go to the **Profile page**, edit the preferences to include *Halal* and *Healthy*, and save the changes. Returning to the Home page, we can now see that the recommendations update in real time, reflecting eateries that better match these selected preferences. This demonstrates how user data dynamically personalizes the experience.

Next, let's explore the full eatery database. On the **Explore page**, users can search for eateries by name or location and apply dietary filters such as *Vegan*, *Vegetarian*, *Halal*, or *High Protein*. After applying filters, only the matching results appear. When we click *Get Directions* on one of them, it takes us to the **Details page** for that specific eatery.

Here on the **Details page**, we can view comprehensive information about the eatery — including its address on a **OneMap Singapore** map, real-time crowd and queue data, and average health and hygiene ratings. Below, users can read existing reviews or contribute their own. I'll leave a sample review here. Once submitted, the backend automatically updates both the eatery's ratings and my personal **points balance** in the Rewards system. I'll also post another review that's intentionally inappropriate, to demonstrate the flagging and moderation process later.

Now, let's open the **Rewards page**. This page displays the total points earned by the user, which increase whenever they leave valid reviews. Once a user has enough points, they can redeem them for available rewards in the catalog. This gamification encourages users to provide useful and constructive feedback to the community.

Next, I'll log in with a **different user account** to demonstrate the moderation system. On the same eatery's page, I can see the inappropriate review I just posted. I'll flag it by providing a short reason — for example, "inappropriate content." This flag is sent to the backend and will appear in the **Admin dashboard** for moderation.

Now, switching to an **admin account**, I'll show you the administrative features. In the **Review Moderation panel**, admins can view all flagged reviews along with the reasons provided, and decide whether to approve or remove them. This ensures that the platform remains fair and trustworthy. Admins can also manage **dietary tags**, adding or editing tags like "Low Sugar" or "High Protein" to improve search accuracy. Lastly, admins have the ability to **create new admin accounts**, which ensures that only authorized personnel can manage platform operations securely.

Before wrapping up, I'll show how users can manage their accounts. From the **Profile page**, any user can update their email address or change their password. These updates are validated and encrypted on the backend to maintain security and integrity.

And that concludes my demo of **HealthyAura** — a personalized, secure, and community-driven food discovery platform that combines smart recommendations, clean architecture, and key design principles such as separation of concerns, dependency injection, and the strategy pattern. Thank you.