## 1. Introduction

**Slide 1**

Good morning professor, TA and classmates. Today we are presenting HawkerStalk that allows users to discover the hawker gems in Singapore! With me today, we have Ming Hao, Zhen Ming, Zhi Wei, Weng Shi, Xin Yee, Navitraa and Hoang.

**Slide 2**

This is a brief outline of what we will cover in today’s presentation.

**Slide 4**

Hawkers are struggling to keep up with rising costs. Core inflation in Singapore is at 5.5% and continues to rise, as shown by the increasing consumer price index on the graph. Additionally, GST hikes add further pressure on hawkers. Hawkers face higher rents and operating costs, yet consumers still expect low prices.

**Slide 5**

Furthermore, preserving hawker culture is crucial to Singapore’s heritage and pride, especially since it has gained global recognition. We should continue to protect this unique culture. To support hawkers, we need to boost demand for their food by making it more accessible to everyone.

**Slide 6**

In that light, we introduce to you - HawkerStalk, a one-stop platform that connects tourists, residents and hawkers to preserve Singapore’s unique hawker culture!

**Slide 7**

\*\* Use Case Diagram \*\*

**Slide 8**

Here are a list of features we include in our app: Firstly, authentication through sign-ups for users, hawkers and admins, a verification email sent to inbox upon sign-up and login for any type of user. Secondly, the admin user is able to approve or reject a newly-created hawker account, suspend any account and view the account details of all users and hawkers.

**Slide 9**

Thirdly, the hawker is able to initialize his profile or stall during sign-up, update his stall’s menu, update his stall’s opening hours and days, view his stall reviews, view fault reports directed at his stall, update opening status outside formal opening hours and delete his account when he wants to close his stall permanently.

**Slide 10**

Lastly, the user is able to view the hawker centre on the map, search for hawker centre on search bar, see the stalls present at a hawker centre, view the menu, opening hours, ratings and reviews of each hawker stall, submit a stall review and submit fault reports.

**Slide 11**

Our key non-functional requirements include a user-friendly interface that is simple and intuitive to use, security and data privacy to protect sensitive user and hawker information, and a scalable design to handle more users without compromising performance.

**Slide 12**

In building our app, we used 2 APIs - namely Google Map API and EmailJS API. Our dataset is obtained from the list of government markets hawker centres in data.gov.sg.

**Slide 13**

For our frontend, we primarily used html and css. For backend, we used PHP, Javascript and the Microsoft Azure SQL database.

**Slide 14**

Now, without further ado, here’s a live demo of our app!

\*\* Insert live demo \*\*

## 2. Good SWE Principles

**Slide 15:  
Title:** Introduction to Good Software Engineering Practices  
**Script:**"Thank you! Now, I'd like to introduce some key software engineering practices that will help us work more efficiently and effectively as a team."

**Slide 16:  
Title:** Implementation of README  
**Script:**"We start with the README. This simple yet essential document gives every developer a clear overview of our project, helping us stay aligned and maintain consistency across the board."

**Slide 17:  
Title:** Writing Code with Comments  
**Script:**"Next, we’re focusing on writing code that includes comments. Our goal is to make our code readable and understandable, which makes collaboration easier and reduces misunderstandings among team members."

**Slide 18:  
Title:** Introducing an Online SQL Database  
**Script:**"Now, let me introduce an online SQL database. Using this remote database enables smooth, real-time collaboration across locations, allowing us to work together seamlessly on data management."

**Slide 19:  
Title:** Consistent File Naming Conventions  
**Script:**"Consistent file naming conventions are essential for maintaining a clean, organized project structure. This practice not only prevents confusion but also improves readability, making it easier for anyone to understand our work at a glance."

**Slide 20:  
Title:** Reusability and Refactoring  
**Script:**"Finally, reusability and refactoring are vital practices. By avoiding redundancy and structuring code for reuse, we ensure that our project remains maintainable and flexible for future development needs."

**End Note:**"These best practices help us create a more organized, readable, and collaborative codebase that supports both current and future development."

## 3. System Design

**Slide 23**

Now, I will explain our system design.

**Slide 24**

The architecture diagram for HawkerStalk illustrates a high-level view of our system's structure, showing how different layers interact. We designed the system with four distinct layers. The first layer, known as the presentation layer, includes the UI pages of the app, where users interact directly with the system. Below this, the app logic layer contains the controller classes that manage the core operations of the app. The next layer, the data access layer, holds the APIs that connect the app logic to data sources. Finally, the persistent data layer stores user data in a database. Communication flows sequentially from the top layer down to the lower layers, ensuring a clear and organized system architecture.

**Slide 25**

This class diagram for HawkerStalk represents the structure of our system, including its classes and their relationships. The boundary classes—such as loginPage and removeAccountPage—represent the UI pages in our app. These boundary classes interact with control classes like userSignUpValidator, searchValidator, and resetPasswordValidator, which handle core validation and processing tasks. These control classes then access the central database, our main entity class that stores user and app data. This layered approach enables a structured and modular system architecture, making our app easier to manage and extend.

**Slide 26**

This dialog map for HawkerStalk illustrates how different actors, such as users and hawkers, interact with various pages in the app. Each box represents a specific state or page, while arrows indicate the actions that allow transitions between these pages. What sets this dialog map apart is its categorization into two main domains: user and hawker. Each domain has its own unique set of pages, tailored to the specific interactions and needs of each actor.

## 4. Design Pattern

**Slide 27**

Moving on to the design pattern,

**Slide 28**

We chose the Model-View-Controller (MVC) architecture for HawkerStalk for three main reasons. Firstly, HawkerStalk serves three distinct user groups—hawkers, users, and admins—each requiring different interactions with the app. MVC enables these groups to engage with the app in unique, tailored ways. Secondly, the app needs to manage real-time data efficiently, such as providing live updates on stall operating status. Finally, because the MVC structure allows independent modification of each component, new features can be added with less risk of bugs, making the app more scalable and maintainable. Therefore, MVC is well-suited for building a flexible, efficient system for HawkerStalk.

**Slide 29**

In HawkerStalk’s MVC pattern, the model component manages the core logic and data. It handles tasks like verifying accounts, storing reviews and managing stall data. The view component displays this data to users, providing user-friendly interfaces and showing real-time content such as reviews and ratings. The controller links the model and view, by processing forms, validating data, and responding to user actions. This setup keeps data management, display, and interaction-handling separate, making the app easier to maintain and scale.

**Slide 30**

The MVC architecture in HawkerStalk incorporates both the observer and strategy design patterns. The observer pattern is applied between the model and view layers, allowing the view to automatically update whenever the model’s data changes. This ensures that users see real-time information without direct coupling between the layers. Meanwhile, the strategy pattern is used within the controller layer to manage different business logic and behaviors. This approach allows the controller to dynamically select appropriate actions based on specific interactions or conditions, making the system more adaptable and modular.

## 5. Traceability in Project Deliverables

**Slide 31**

*Thank you Navitraa. Next I’ll be walking you through the traceability in our project deliverables, using the login feature as an example.*

**Slide 32**

**First, let’s look at the use case description.**Our login use case describes how users enter their credentials to access the system. It specifies preconditions, such as the need for valid credentials, and postconditions, like successfully granting or denying access. This use case ensures that we clearly trace the purpose and expected outcomes for login.

**Slide 33**

**Next, in our class diagram**we’ve divided the login functionality into three main classes:

* The **UI boundary class**, which manages user interactions,
* The **LoginValidator control class**, which verifies credentials, and
* The **Database entity class**, holds the user data.

Each class plays a role that aligns with our design principles, helping us trace functionality back to specific components in the code.

**Slide 34**

**Moving on to the sequence diagram,**It demonstrates how these classes interact during the login process. The UI class sends user input to the LoginValidator, which then checks with the Database to validate credentials. This illustrates the flow of control and data, allowing us to trace each step in the login process.

**Slide 35**

**Now, let’s look at good design principles applied in the code.**Our code adheres to **single responsibility and separation of concerns** principles. Each class is designed with a specific role, making it easy to modify or debug any issues within a single component.

**Slide 36**

**Next, we have black and white box testing, starting with the control flow graph.**The control flow graph visualizes paths through our login code, showing possible paths a user might take. This helps us ensure that all critical paths are covered in testing.

**Slide 37**

**Finally, our test cases and results for black and white box testing.**We’ve defined test cases covering both typical scenarios, like correct credentials, and edge cases, like invalid input. Our test results confirm that the login process behaves as expected, with any failed cases documented for improvement.

*This structured approach shows how traceability keeps our project aligned with requirements, from design to testing.*

*Now I will pass on to Minghao who will share with you the future prospects of HawkerStalks*

# 6. Future Prospects

**Slide 39**

In the future, we plan to expand our app to include new features. Firstly, live crowd monitoring to inform users of peak and off-peak hours so they can plan their trips accordingly. Secondly, personalised recommendations using AI to recommend stalls or dishes to users. Thirdly, mobile payment integration via digital wallets, credit cards or QR codes.