Group 45 Script

INTRO (Yuxuan, 1.5 mins)

Good morning, everyone. Today, we're excited to present our web app, ShopBlock. So, what exactly is ShopBlock?

Imagine this: there are countless e-commerce websites offering goods for sale. However, some low-income families may struggle to afford these items, especially the more expensive ones. At the same time, there are things we rarely use at home, like a tent, for example. What if instead of buying, we could borrow these items for a short period?

To address this need, we developed ShopBlock, a web application that allows users to rent items from one another. Our app has four main features, as shown on the slide: browsing available rental items, searching, sorting, and filtering items, making offers, and leaving reviews after borrowing.

This is our use case diagram, which shows the five main systems connected to the user. Users start by logging into the User Account Management System, where they can register and manage their profiles on ShopBlock. All user information is securely stored in our database.

Once logged in, users can browse through available listings in the Browsing System, find the items they're interested in, and make offers. The Listing System lets users view all available items and manage them. It's also

connected to a map API, allowing users to choose meetup locations conveniently.

The Review System lets users rate both other users and items they've rented, promoting trust and transparency within the platform. Finally, we have the Payment System, where users complete their transactions securely via PayPal.

DEMO (15 mins)

Welcome to Shop Block! This is Group 45's live demo. The first functionality we want to showcase is account creation. The system handles basic validation errors; for example, your username must be at least 4 characters long, and your email and phone number must be unique.

Next, let's log in. The login process ensures that all fields are filled in correctly, with security measures in place. For instance, if you enter the wrong password five times, you'll need to wait five minutes before trying again. Additionally, if you've forgotten your password, you can reset it. Simply confirm your email and phone number, and upon a successful reset, a green confirmation message will appear.

Once logged in, you can access your profile and change your password. The password update process is similar to resetting it, ensuring that your new password meets format requirements and that the fields match. Congratulations, you've successfully changed your password!

In the profile section, you can also update your profile photo, biography, username, and mobile number. Now, let's explore the browsing experience. There's a hero page, an explore page, and a support page accessible at the bottom. The navigation bar allows you to quickly browse through various

categories, and there's a filter option to sort items by rating or price. There's also a search bar to help you find specific items quickly.

Now, let's create a listing together. For example, let's say I want to rent out my road bike. I'll add a title, a short description, and set the price, unit, category, and location. The OneMap API makes it easy to set the location. After adding a photo, we submit the listing, and voila! Our new listing is live.

From a renter's perspective, let's look at this bike listing. It's exactly what I'm looking for! Since no one has rented this bike before, all the slots are available. I'll select a time, set the end date, and submit my availability, then make an offer.

Let's try booking another item, a tent. I'll check for available slots. The system highlights slots that are unavailable, so I'll pick a different time. I set my amount and submit it. Awesome!

Now, let's manage our offers. Here we can see the offers we've received. Someone wants to rent my bike, but I'm not interested in that offer. However, I'll accept the offer for my camping tent.

From the renter's perspective, let's go to our offers. Once we're ready to proceed, we can pay securely using the PayPal API. After entering our payment information, we confirm and process the payment. A message appears confirming the transaction was successful.

We can view our purchases, seeing those that are still in progress. In the purchases section, completed transactions are also listed. We can rate the user based on our experience, perhaps noting that they were an amazing seller. After submitting the review, the rating is displayed.

Good Coding Principles (Ze Ming, 2 mins)

Hi, I'm Ze Ming and I will be talking about Software Engineering Practices and Design Patterns.

The tech stack that we use, we separated out the front-end and back-end. The front-end uses javascript with react, and material ui for some ui components, as well as some handwritten css. The backend is a CRUD API written in Python with Django.

For good coding practices, we wrote documentation for both the front-end and back-end. This README is included in the repository itself. This also serves as a setup guide for any incoming developer. It contains instructions on how to set the project up locally so that they can develop.

We also wrote consistent and clear comments in our code. This is so that functionality is clearly documented. We followed the industry standards for this. In this case, we followed the PEP convention for python docstrings.

These docstrings will then also get picked up when we generate our API documentation. This API documentation is generated from the code, so when the code changes, the documentation also changes, hence it is always up to date. With this documentation, it allows the front-end developers to work on the program independently.

It is a friendly interface to play around with the different query parameters and request bodies. There are also examples that are included, these examples are also generated from the code itself. So that there is reference for the developers that are using the API.

While developing, we also had an auto formatter. This ensures that all code is familiar to anyone that is coming onto the project new. In our case, we used Python's Black, which is industry standard and follows all PEP standards.

We also had automated tests that is ran on the CI. These tests run on identical databases to what we have on production. On every commit it will run the test. If it fails, we will know immediately. As can be seen on the picture, we will then fix it.

We used the Facade Pattern as taught in the lecture here. We have a unified interface through an API endpoint. The frontend does not have to know the changes the backend makes. This decouples the frontend from the backend entirely. The backend is then free to make changes as is necessary.

We also used the Factory Pattern here. The code does not depend on any specific database, and instead accesses the databases through the ORM interface. We can easily swap out the databases without any changes to the code if we need to.

Traceability (Li Min, 1.5 mins)

Hi, I am Li Min and I will be talking about the traceability of two use cases in ShopBlock.

Starting with use case description for the login functionality, which is the foundation of traceability in our system. The flow of events provides a step-by-step sequence of actions that helped us work toward our class diagram.

Each component here—the LoginView, LoginController, and User corresponds directly to the actions and validations outlined in the use case description. For example, the User class attributes, such as username and password, trace directly to the preconditions that the user must have the correct credentials.

The sequence diagram shows the flow of events defined in the use case description. Each step from ClickLoginButton to SubmitCredentials and VerifyCredentials matches the events and validations specified in our use case.

Finally, we have the test cases, which complete the traceability chain by verifying that the login functionality performs as designed.

Each test case is tied back to a specific scenario described in the use case and sequence diagram. For example, Test Case 1 confirms a successful login when valid credentials are provided, directly tracing back to the successful path in the flow of events.

Similarly "Create Listing"'s traceability begins with a clear Use Case Description that outlines the process for users to create a new listing to rent out items or services. This includes essential preconditions, such as the user needing to be logged in, and specifies the necessary information required to submit a listing.

Alternative flows are provided to handle situations where required fields are missing.

The Class Diagram then breaks down the structure, showing how components like ListingView, ListingController, and Listing interact to enable listing creation. Each attribute and method in the Listing class, such as title, description, and update_title, supports the fields and actions detailed in the use case.

The Sequence Diagram visualizes the interaction between the user and the system, tracing each action in the listing creation process, from clicking the create button to confirming the listing in the database.

Finally, the Test Cases validate this feature through scenarios like filling in all fields, leaving fields blank, and canceling before submission.

This comprehensive flow guarantees that both these features are fully traceable from requirements through implementation and validation, reducing the risk of missing functionality or misalignment with user needs.

Thank you for listening to our presentation!