Slide 1:(Yong Khang)

Speaker 1 :

Good afternoon, everyone! We are Team 4, and today, we will be introducing you to our project PropBidder.

Our team consists of [names]: Qi En, Spencer, Yong Khang, Atul, Shreya and Naveen,

We've developed a platform to bring greater transparency and efficiency to the property agent selection process in Singapore.

Let's begin by understanding the real-world problem we're trying to solve.

Slide 2: Problem Statement (Yong Khang)

Speaker 2 :

In Singapore's highly competitive property market, over 33,000 agents are registered with the Council for Estate Agencies as of 2023 — and that number is still growing to about 36,000 today.

Yet despite this large supply, sellers often rely on referrals or word-of-mouth to find agents. This leads to non-transparent commission structures, limited visibility into agent performance, and often, suboptimal outcomes.

There's no standard way for sellers to compare agents based on their bidding rates, past reviews, or success records effectively.

Meanwhile, existing platforms are mainly focused on listing of properties, not matching agents to sellers — which is where PropBidder comes in.

Slide 3: Solution – PropBidder (Yong Khang)

Speaker 3 :

PropBidder is a real estate agent bidding platform that adds transparency and competition to the agent selection process.

Agents submit their commission bids, and sellers can easily compare them alongside past reviews and transaction history.

This gives sellers the power to make informed decisions based on data — not just personal referrals.

Additionally, to help sellers set the right price, we fetch similar past resale transactions via government APIs, ensuring sellers list their properties with realistic market-aligned pricing."

Slide 4: Use Case Diagram (Yong Khang)

Speaker 4 :

Here's an overview of how different users interact with PropBidder.

We've broken it down into 4 different groups:

Users – can register, log in, and recover passwords via OTP.

Sellers – can list and manage properties, view agents bids, and leave reviews.

Agents – can view active listings, bid on the listed property, and track their own bidded properties.

And the system handles tasks like OTP generation, database updates, and displaying property info.

Our architecture ensures a smooth experience for both agents and sellers while enforcing data integrity through validations and structured flows."

Slide 5: Secondary Features – Part 1 (Qi En)

Presenter 5:

Thank you Yong Khang,

Beyond this core features that yong khang has described,, we've added some thoughtful enhancements :

Under the Home, Agent and Seller Dashboard, Sold properties are shown with seller and agent details – making the platform transparent for future users.

Under the Listing Property Section,

We incorporated Auto-updating town dropdowns to prevent mismatches during listing.

and Sellers can instantly fetch up to 5 recent similar resale transactions to guide their pricing.

Slide 6: Secondary Features – Part 2 (Qi En)

Presenter 6:

On the seller's end, they can: View all agent bids in a clean modal, Accept the best offer, Review the selected agent, and Edit or delete listings if needed. and lastly on the agent's end,

We've introduced: Bid tracking where agents can see their offer and the current lowest bid. There will be a Live feedback that is shown in green when you're leading, and red when outbid as well as A conditional rebid button, only shown if outbid and the listing is still active, keeping the interface clean and relevant.

Slide 7 (Qi En)

Presenter 7:

To power key features in PropBidder, we've integrated several APIs to streamline user experience and data accuracy.

1. Twilio SMS API

We use Twilio for SMS OTP verification during account registration and password reset.

This adds a layer of security and identity verification, ensuring each user is legitimate.

2. CEA Salesperson Information API from data.gov.sg

This API allows us to validate agent credentials during registration by cross-checking with CEA's public database — ensuring only real agents can bid on listings.

3. HDB Resale Flat Prices API

Also from data.gov.sg, this is used in two key ways:

To populate the Town and Street dropdowns dynamically, reducing errors in listing entry.

To generate the top 5 similar resale transactions when a seller lists a property, helping them make pricing decisions based on actual government data.

These APIs help us maintain reliability, transparency, and automation across core workflows in our application.

Slide 8: System designs (Shreya)

Now let's dive into the underlying architecture that supports PropBidder's functionalities.

Our system is designed using modular and layered architecture principles to separate concerns, promote maintainability, and enhance scalability."

Slide 9: Architecture Diagram LAYERED ARCHITECTURE (Shreya)

This is our high-level system architecture organised into multiple layers

1. The user layer interface is the entry point for sellers, agents, and users without an account. There is a tailored dashboard for each
2. Then UI Management & Authentication handles form validation and role-based interfaces, displaying role-appropriate content
3. Next, the application logic layer houses the business logic for the various functionalities in our application, for instance, account management and user verification to name a few
4. The persistent data layer stores all of our system databases
5. And finally we have the external API layer that includes Twilio for otp verification and data.gov.sg APIs as discussed earlier for agent validation and up-to date HDB resale flat data.

Slide 10: Class diagram (Shreya)

Our class diagram follows the boundary-control-entity design pattern to distinguish responsibilities clearly."

- Entity Classes (in orange):
  These represent core data structures such as User, Agent, Seller, Property, Review, and BiddingSystem. Each class holds data and methods that model real-world behaviors.
- Control Classes (in green):
  These classes handle logic like validation, password checking, authentication, and controlling property views or bid submissions.
- Boundary Classes (in blue):
  These represent the UI components or external interaction points such as OTP sending, registration forms, and profile interfaces.

"This clear segregation between data, control, and boundary elements ensures that our system remains modular and testable—enabling faster debugging, collaborative development, and future scalability."

Slide 11: Live Demo (Spencer)

Now we will move on to our live Demo of our product, PropBidder, a real estate commission bidding platform designed to help sellers find the most competitive property agents in Singapore.

In this demo, we'll walk you through key features from both the seller's and agent's perspectives.

(Homepage Walkthrough)

Firstly, Let's start with our homepage, which is designed for clarity and usability.

Scroll down

You'll see the Latest Properties section showing recently sold units.

Below that, we highlight our platform's benefits in the Our Services area.

(Scroll to show the latest properties and service section.)

Clicking 'View Details' or 'Login' will bring you to the account portal.

(Registration)

While clicking on the Register will bring you to the registration page, new users can register as a Seller or Property Agent here. Registration requires basic information such as full name, username, password, confirmation of password as well as a phone number. The phone number is used for Twilio OTP verification for security purposes.

For Property agents, there's an additional validation step where we check the Agent ID against the CEA database via data.gov.sg API

(Show OTP and Agent ID validation flow)

(Register an agent account with AgentTest7,AgentTest7, Password1!,Password1! P015089H)

Show OTP received via phone

Due to API limitations, only one phone number is currently used for OTP simulation.

(Login + Forgot Password)

After account creation, users can log in.

If they forget their password, a secure reset is available—again using OTP verification.

For this demo, we'll be logging in using pre-registered accounts in our database.

(Login to Test1,iLove2006!, Seller Account (Testing Account))

Once logged in, sellers land on their personalized dashboard.

The homepage remains, but now displays seller-specific content like tools and latest transactions.

Sellers can easily manage properties from here.

(Listing Property Flow)

Now , Let us list a new property by going to the List Property page.

Sellers input details here such as flat type, town, street name, floor area, lease remaining, commission cap, and listing price.

Street and town fields are linked dynamically to avoid mismatches—selecting a street auto-updates the town.

Select street name first then town (town will be auto updated)

For Example: We'll list a 5-room flat in Yishun, 120 sqft, 57 years remaining, 5% max commission

(List 5 Room Flat, Yishun, Any Yishun Street, 120 sqft, 57 Years, 5% Commision)

Clicking 'Fetch Past Prices' fetches up to 5 similar resale transactions from the HDB API to help sellers price realistically

(List 4 Room Flat, Woodlands, Woodlands Ave 3, 90 sqft, 90 Years, 2% Commision)

Once listed, the system redirects to the 'Your Properties' page.

(View Your Properties – Seller's View])

Here, sellers will be able to:

View all their listings.

Click ‘View Biddings’ to open a modal showing incoming agent offers.

Click ‘Edit Listing’ or ‘Delete’ if needed.

(Show sample listing, bidding modal, and edit interface.)

Once a bid is accepted:

The property status changes to ‘Sold’.

Further bids are disabled.

A review section appears to rate the agent.

The sold property also appears on the homepage and other dashboards as part of the transaction history.

(Agent Dashboard)

Let’s now log in as an agent.

(Logins to Test2,iLove2006!, Agents Test Account)

- Agents see a dashboard tailored to active listings. Over here you are able to see the previous listed property by the Test Seller Account.
- They can view property details and submit a commission bid—must be less than the max commission.

If the bid is valid, the lowest bid gets updated.

(enter 5.1% Shows error), (enter 4.9% gets updated)

(View Bidded Property)

After submitting a valid bid, Agents can then check the ‘View Bidded Property’ page.

- Each listing card shows your bid, the lowest bid, and a status indicator:
- *You’re currently leading!*
- *"You’ve been outbid."*

If outbid, a rebid button appears dynamically—agents can adjust their offer without cluttering the interface.

(Scenario Demo – Bidding Competition)

So Now Let's simulate a real bidding battle where we have another agent try to bid on the same property but with a lower bid price and see what happens.

We first Log in as another agent, submit a lower bid (e.g., 4.7% vs 4.9%).

(Login to Test4, iLove2006!, AgentTest2)

(Submit a lower Bid and relog in to Test2, Agent Test1)

The previous agent now sees they've been outbid, and the rebid button appears.

This keeps the process competitive and transparent.

Sellers are empowered to choose not just based on price, but also by viewing the agent's profile and past reviews before accepting.

Through PropBidder, we:

- Bridge the gap between property sellers and agents.
- Enable data-driven commission bidding.
- Improve transparency in real estate agent selection.

Thank you for your attention—let's move on to the next part of our presentation which will talk about Software engineering practices and design patterns.

Slide 12: Software Engineering Practices and Design Pattern (Spencer)

Now that we've walked through our full platform demo, let's move on to the Software Engineering Practices and Design Patterns we followed while building PropBidder."

This was essential for ensuring not only functionality, but long-term maintainability, scalability, and team collaboration across different components.

Slide 13: Waterfall or Agile? (Spencer)

"Initially, our team adopted a Waterfall approach, planning everything upfront and assigning fixed tasks across the project timeline. But we soon realized this wasn't ideal for our situation —

we needed to deliver a Minimum Viable Product (MVP) quickly to gather feedback from our TA during lab sessions.

That's when we decided to shift to an Agile approach, using the Scrum framework to support faster iteration and feedback cycles. Each Sprint lasted two weeks, aligning with our lab schedule, so we could present tangible progress regularly and adapt based on feedback.

Slide 14: Good SWE Practices (Naveen)

We also made sure to adopt good software engineering practices during our development cycle, and we focused on 3 main things:

Documentation Practice, Coding and Naming Convention Style, Refactoring and clustering.

Slide 15: Documentation Practice (Naveen)

First is documentation. This was important because we wanted anyone in our team—or even outside—to understand how to run and use the project.

The first thing we focused on was our README file. We made sure it had clear steps for setting up the project, including: Navigating to the right folder, Installing the necessary Python packages using the requirements.txt, and How to run the Flask backend locally. We also added the link to open the app in the browser.

The second thing was adding code comments. For example, in our change password function, we left comments explaining why certain checks were needed—like making sure the phone number is in a valid format, and validating the OTP and new password.

This helped the team understand the logic faster. Lastly, we tracked our progress with Git commit messages. Each team member used meaningful messages like "fix change password bug" or "add margin between property cards," so others could follow along easily. This made it easier to work as a team and avoid overwriting each other's work.

Slide 16: Coding & NAMING CONVENTIONS (Naveen)

Next is coding and naming conventions. This is something we tried to keep very consistent throughout the project.

For functions and variables, we used snake_case everywhere. So instead of using random styles, we stuck to names like phone_number, send_otp, and agent_username. This made it easier to read and understand each other's code.

Function names were also clear and action-based like send_otp() clearly tells you what it does. Same for accept_bid(), it's very obvious that it's related to the bidding feature.

Validation logic was written inside helper functions, like is_valid_sg_number() to check the phone number format. This kept our routes cleaner and made debugging easier. Route naming was RESTful and easy to understand. For example, /list_property is used to list a property, and /view_bidded_property shows what the agent has bid on. We followed this structure for all routes, so the naming is consistent across the whole app.

In our forms, like HTML forms for selecting a town or street, we made sure the IDs were named properly. For example, the ID for the town dropdown is just town, and the street one is street_name. This helped us easily access them in JavaScript for DOM manipulation.

Lastly, for error handling, we used try-except blocks in many places, especially for the backend and APIs. So if something goes wrong, the app doesn't crash. Instead, we show a friendly error message to the user, and the actual error is still logged for us to fix later.

Slide 17: Refactoring - N Tier (Atul)

Thirdly, we refactored our project using an N-tier architecture to improve structure and scalability

We split the application into four main layers:

- The Presentation Layer to handle routing and rendering of UI components like forms and templates.
- The Business Layer to handle the core logic, such as resale price calculations and verification workflows.
- The Data Access Layer that abstracts the database interactions from the rest of the app
- And a Security Layer, which manages utility functions and config support.

We also applied the SOLID principles to guide our design.
For SRP — the Single Responsibility Principle, we made sure each module focuses on just one task. For example, resaleprice_service.py handles only resale logic, while auth_service.py is dedicated to authentication. This keeps our code focused and easier to maintain.
For OCP — the Open/Closed Principle, our system is designed to be open for extension but closed for modification. That means we can add new features, like a different type of verification, without touching existing logic. Our use of modular services and Flask blueprints makes this really straightforward.

Slide 18: start of 3.1.6 (Spencer)

Now, let's work thru a use case in detail. We'll be focusing on the register account use case.

Slide 19:

Here is the class diagram for it.
Boundary - AuthRoutes, strategies, config
Control - AgentChecker to verify agent ID, AuthRoutes as a controller for registration flow
Entity - User and Database

Slide 20:

Here is the sequence diagram for it. Tricky part is in the middle when the TwilioOTP sends the OTP to the user and has to return the success msg to the OTP strategy which returns the OTP sent msg to the AuthRoutes. Only after does the POST with OTP and details start.

Slide 21: (Atul)
We implemented the strategy pattern here. Included the MockOTP strategy cuz:

Key Behavior of MockOTPStrategy

No Real SMS Sent to phone

Instead of calling Twilio's API, it:

Prints the OTP to the console (or logs it)

Returns a fake "mocked_sid" (for testing success flow)

Slide 22:

For the testing, we start off with BBT first. Note that this isn't an exhaustive list, we've included the key-tests here for simplicity.

Slide 23:

Here are the equivalence classes for the password part of the use case. We realised that some

tests were based off discrete values eg weak password scenarios so we separated the numerical and discrete values.

Slide 24:

Finally, this slide shows the control flow diagram for the create_account() function. It covers key validations like phone number, password strength, OTP, username checks, and agent ID verification.

We added loopbacks to show how the user is prompted to fix errors and resubmit the form, reflecting real system behavior.

From the diagram, we identified 9 decision points, giving us a cyclomatic complexity of 10 using the formula: Cyclomatic Complexity = Decision Points + 1 = 9 + 1 = 10

We used this to design 10 test cases, each covering an independent path — including both valid and invalid form submissions. This ensures the function's logic is fully exercised and behaves as expected**.**

Slide : Thank you for your attention!