

SCSX Group 3 Code Crafters

MoveInLo!

By: Eugene (U2120698A),
Iain (U2221382K), Yifei (U2201092F),
Joseph (U2223041J), Kim (U2020316D)



Table of Contents



1. MoveInLo



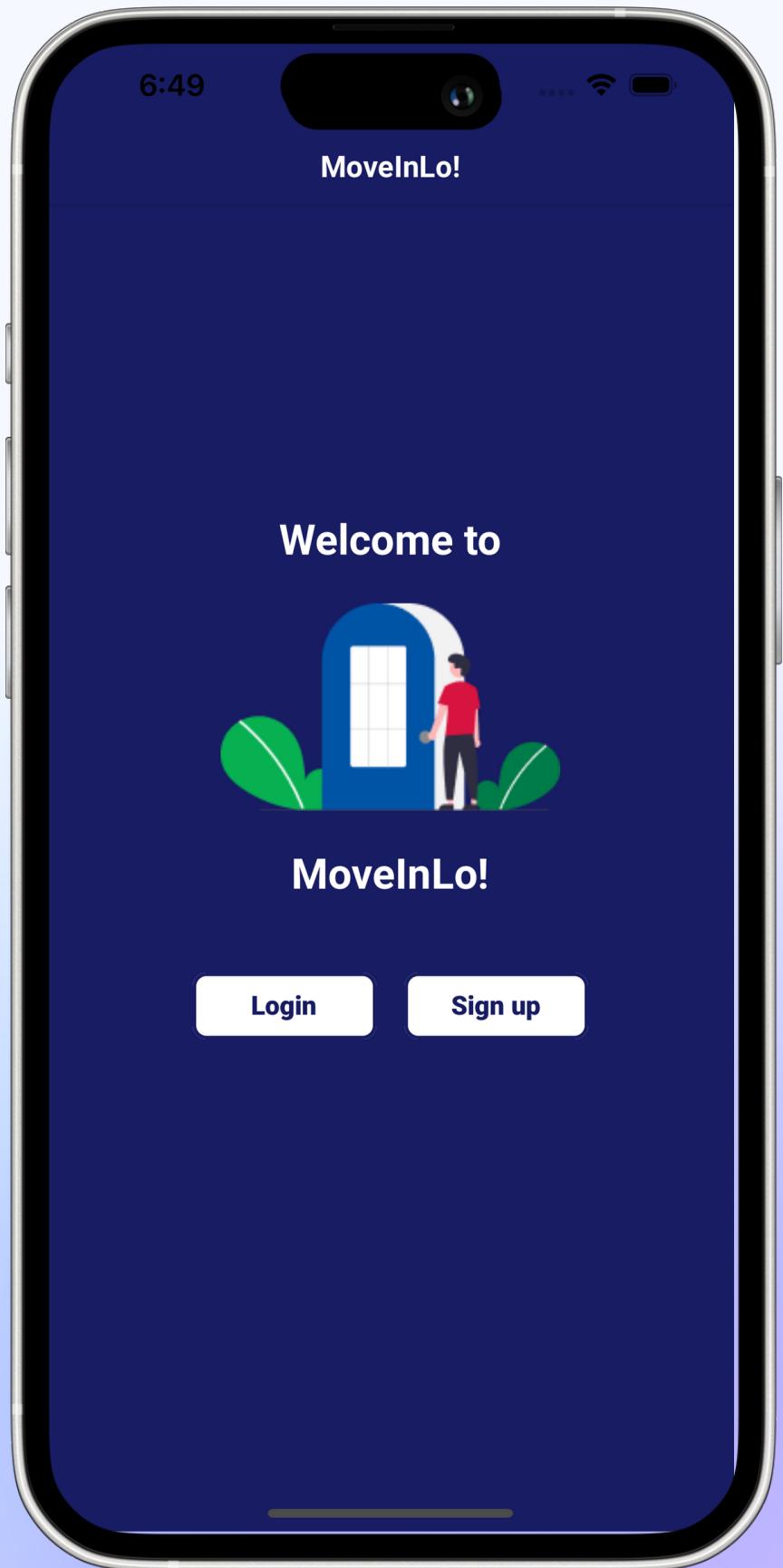
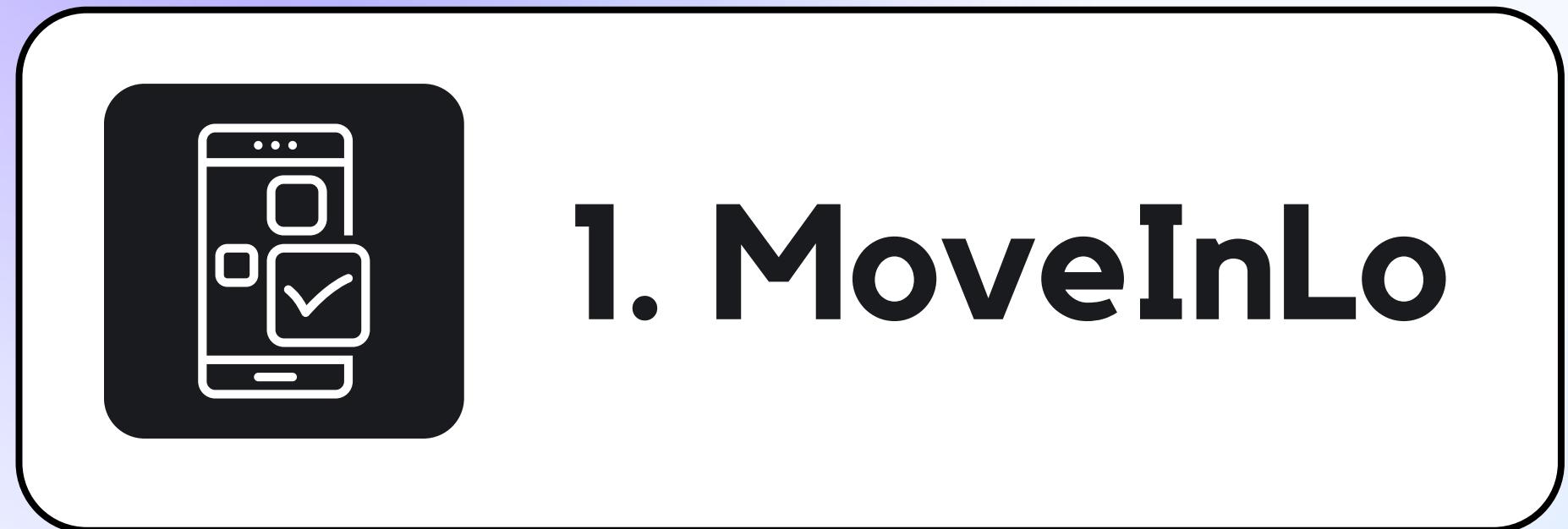
2. Overview



3. SWE Practices



4. Demonstration



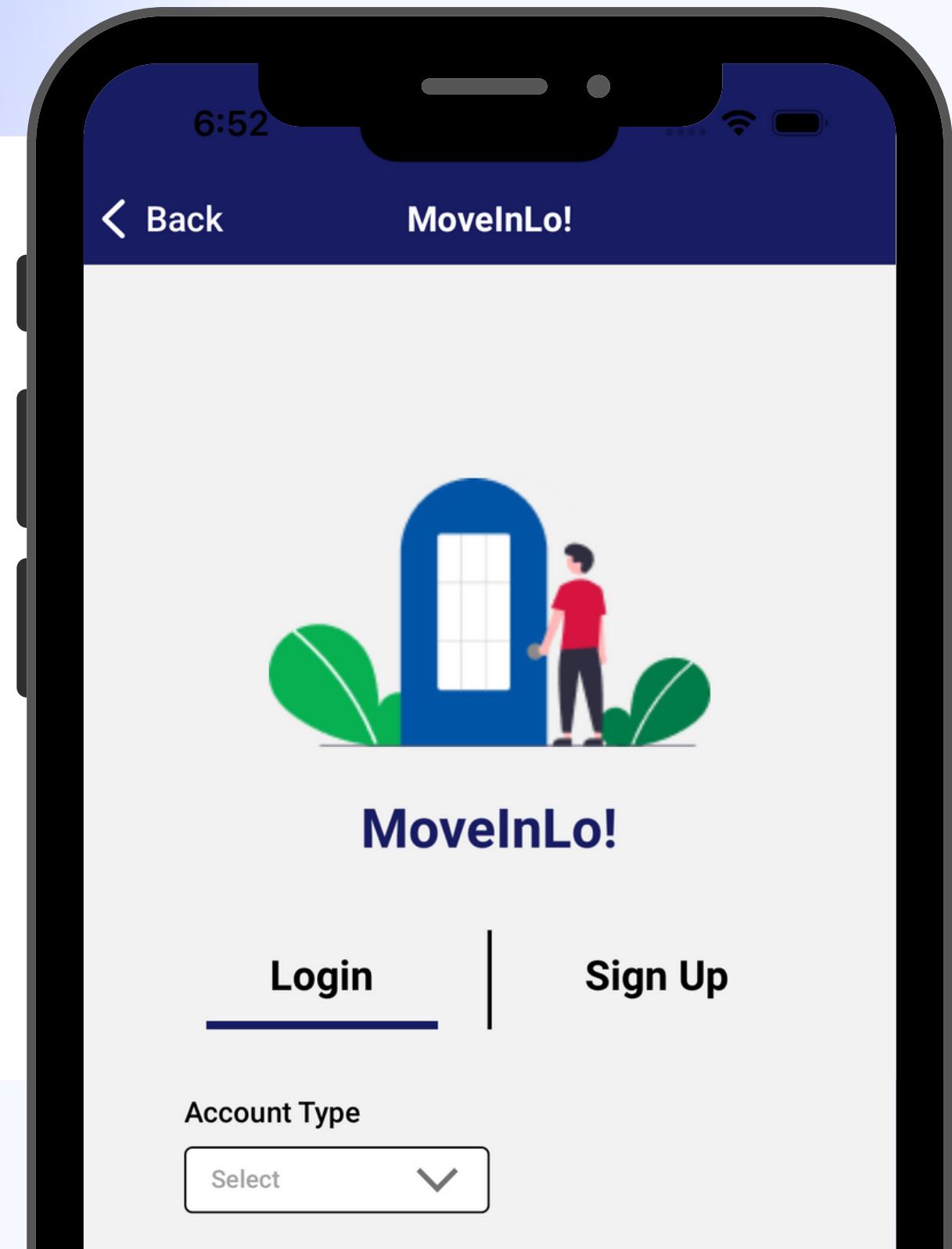
What is MoveInLo?

- MoveInLo is a mobile application that seeks to smoothen the process of moving in and out of university halls/ hostels for new and existing students in Singapore. MoveInLo achieves this by providing a seamless experience for moving in/ out as well as a new avenue for students to seek temporary jobs.
- The inspiration for the name “MoveInLo” was drawn from the commonly-used term in Singapore national service, “ORDLo” which signifies the end of a significant milestone, yet start of a new journey.

Features and Requirements

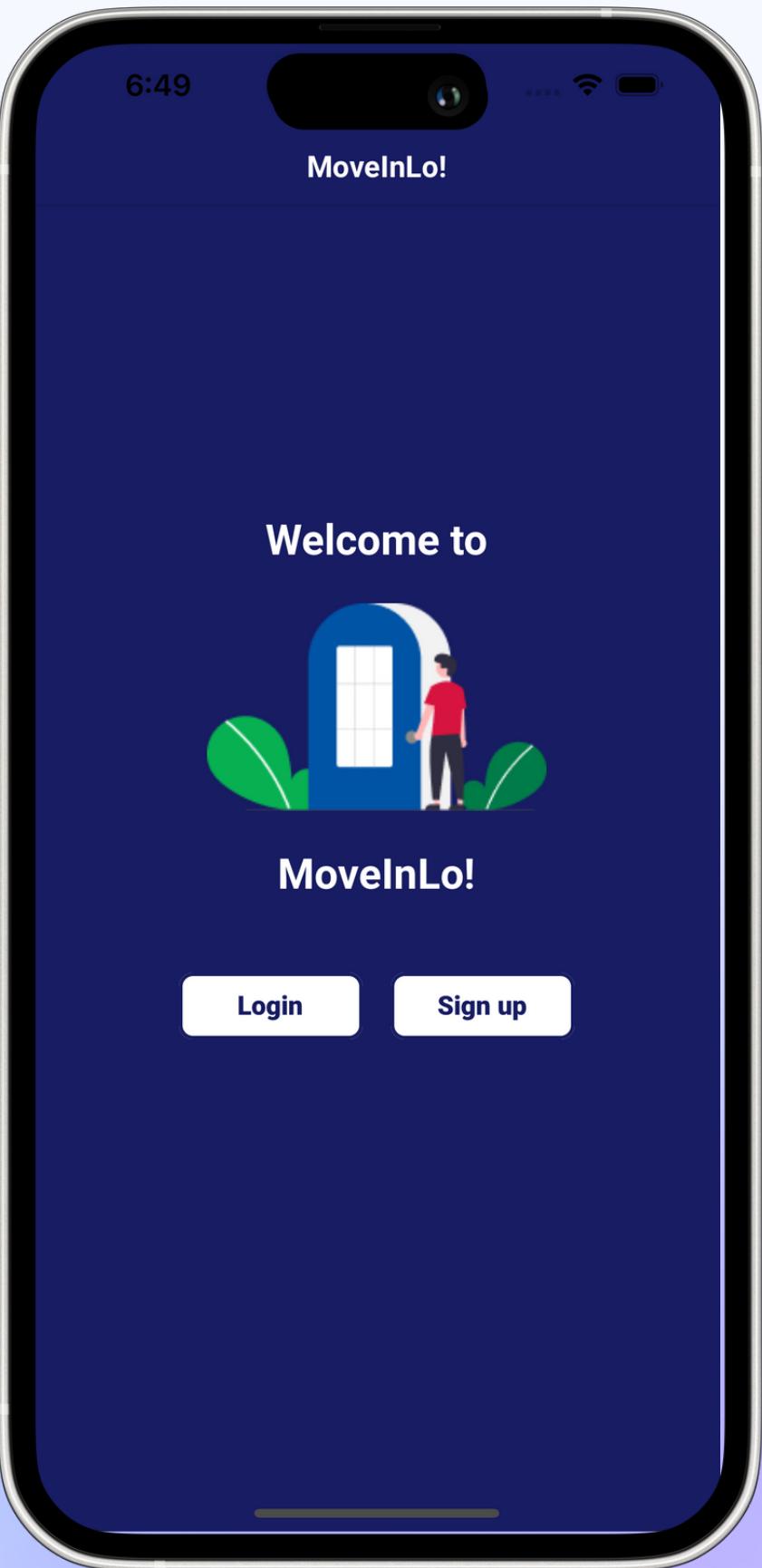
Services

- ✓ Move In/ Out Services
- ✓ Temporary Storage
- ✓ Progress Tracker

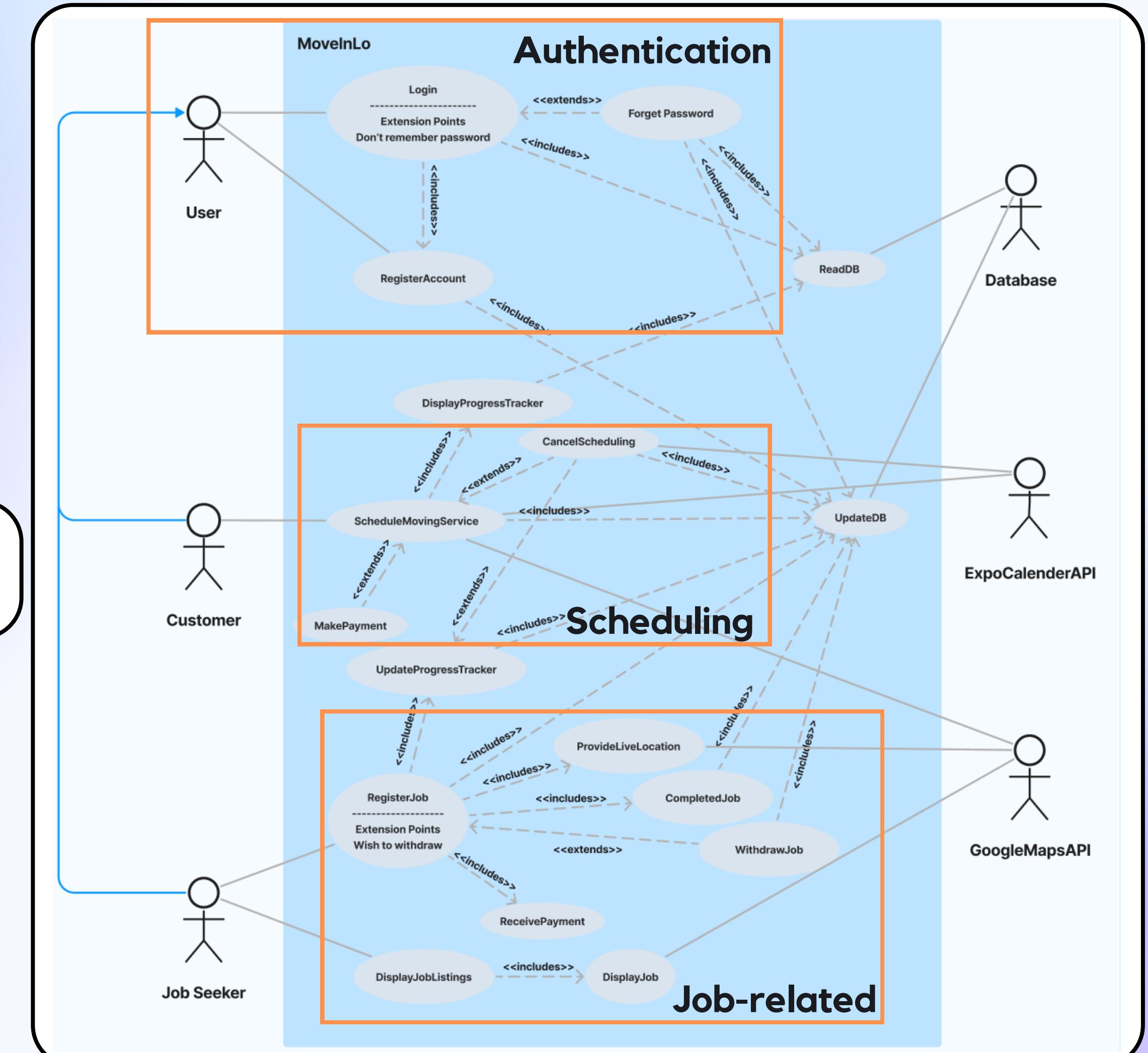


Users

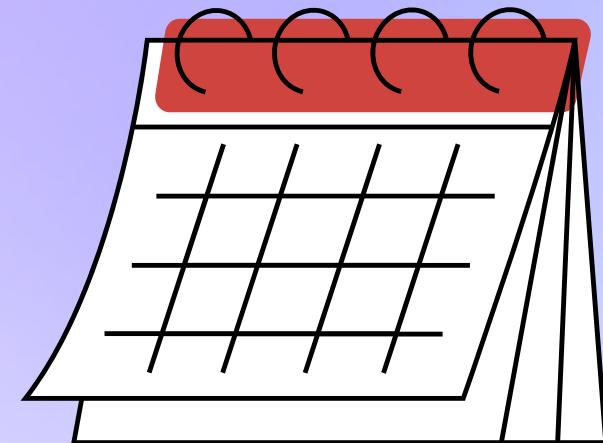
- ✓ Job Seekers
- ✓ Customers



Use Case Diagram



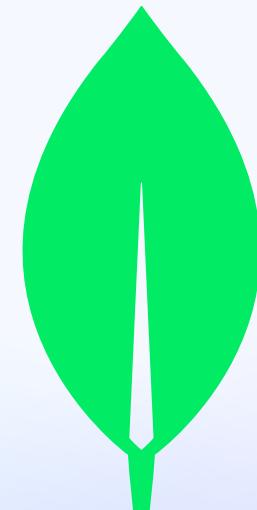
APIs Used



Expo Calendar



Google Maps API



MongoDB Atlas API

Functionalities

Request Calendar Permission

CRUD event

Functionalities

Request Location Permission

Geolocation

Reverse Geolocation

Functionalities

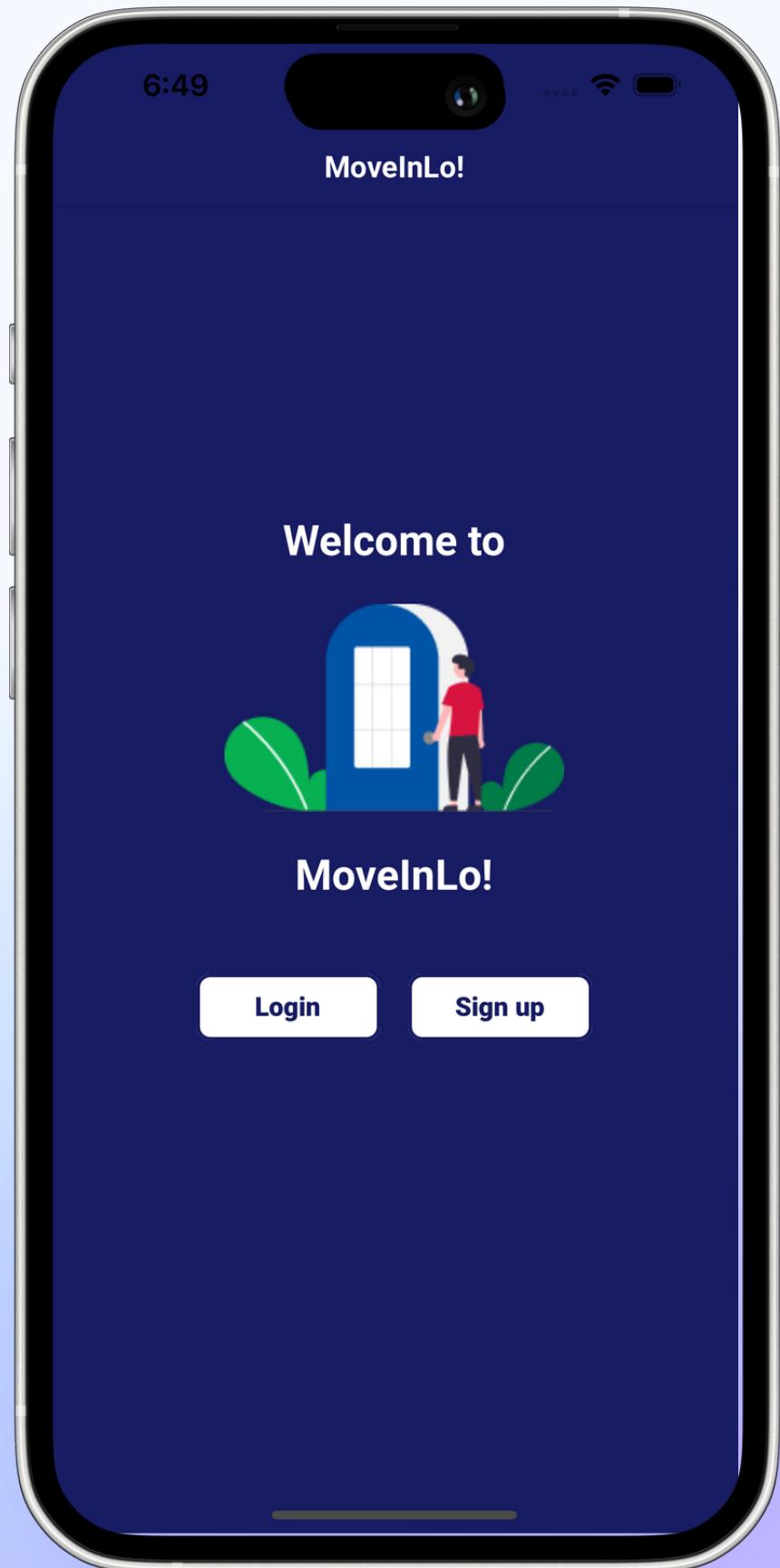
CRUD operations

Aggregate pipelines

REAL-TIME Updates to Database & Device Calendar!



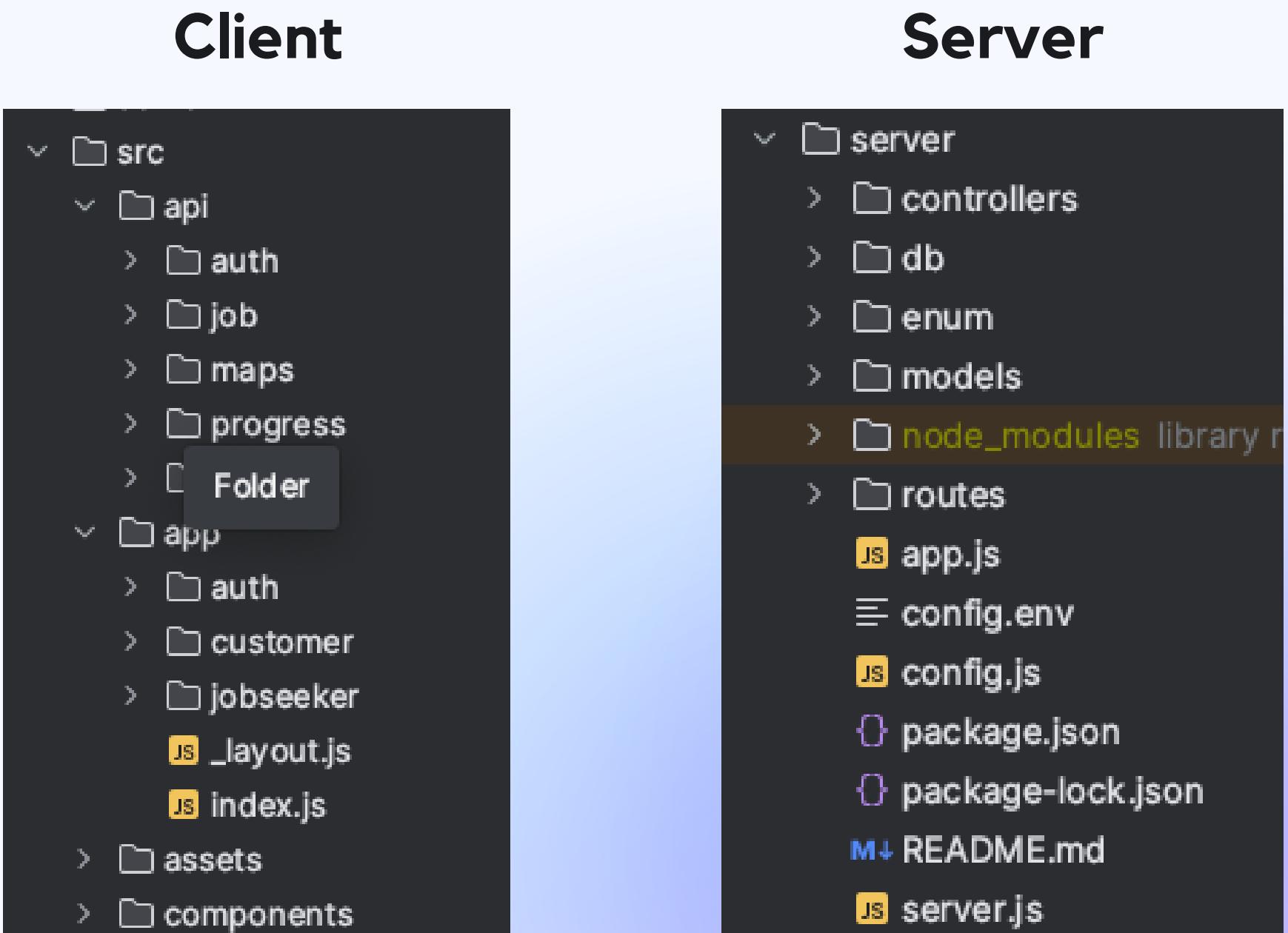
3. Software Engineering Practices



Software Engineering Practices

Client-Server Architecture (with Model-View-Controller)

- ✓ **Separation of concerns** between business logic
- ✓ **Concurrent development** speeds up app delivery
- ✓ **Ease of Readability** for future develop



Software Engineering Practices

Client

```
✓  └─ api
    >   └─ auth
    >   └─ job
    >   └─ maps
    >   └─ progress
    >   └─ service
  ✓  └─ app
    >   └─ auth
    >   └─ customer
      >     └─ calendar
      >     └─ home
      >     └─ schedule
      >       └─ _layout.js
    >   └─ jobseeker
      >     └─ home
      >     └─ joblistings
      >     └─ registered
        >       └─ _layout.js
      >       └─ _layout.js
      >       └─ index.js
    >   └─ assets
    >   └─ components
```

API Handlers

MVC (View)

- Pages for routing
- Clean directory structure

Assets (images, etc)
Reusable Components

Server

```
✓  └─ server
  ✓  └─ controllers
    └─ AccountManager.js
    └─ CalendarController.js
    └─ GoogleMapController.js
    └─ JobManager.js
    └─ ProgressManager.js
    └─ ServiceScheduler.js
  >   └─ db
  >   └─ enum
  ✓  └─ models
    └─ AccountModel.js
    └─ JobModel.js
    └─ ServiceModel.js
  >   └─ node_modules library root
  >   └─ routes
```

MVC (Controller)

Enum & Database

MVC (Model)

API routes

Design Patterns

Creational Pattern

Singleton Pattern

Behavioural Pattern

Chain of Responsibility Pattern

Command Pattern

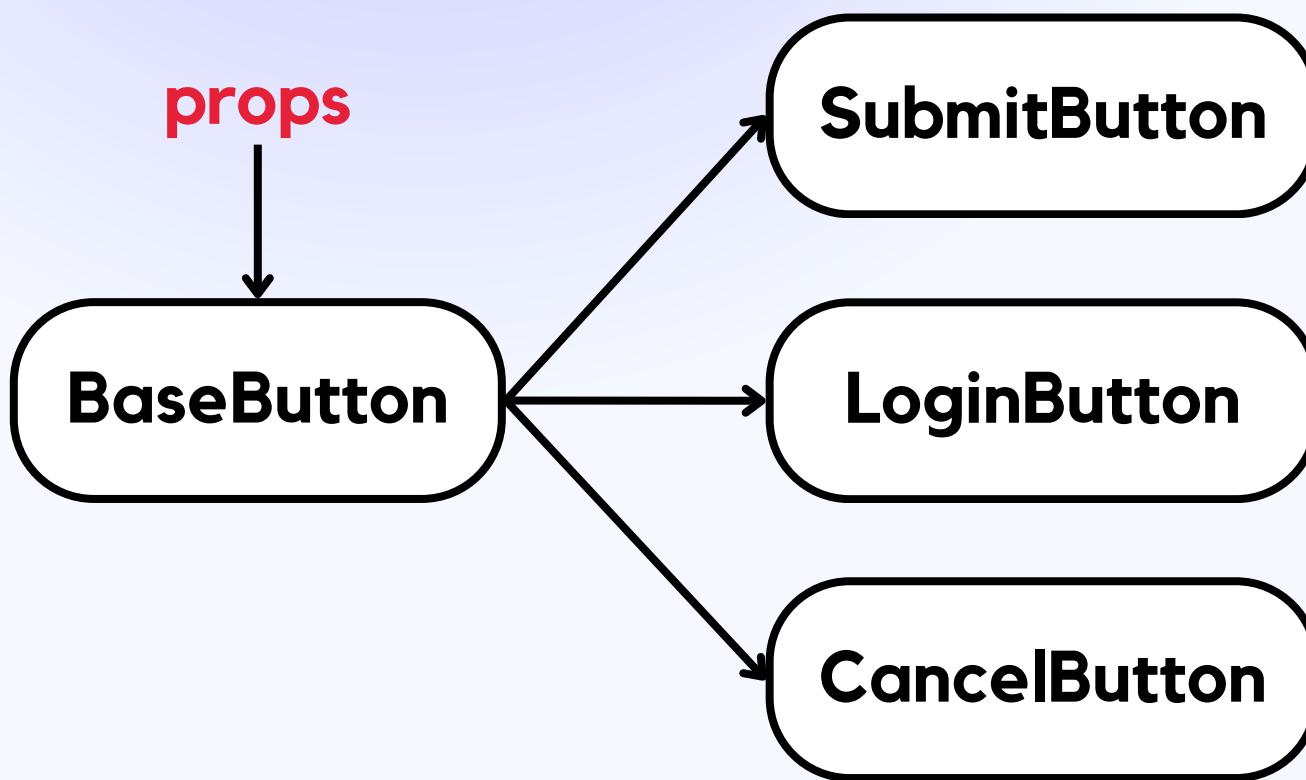
Structural Pattern

Higher-Level Order Component

Design Patterns

Higher-Level Order Pattern

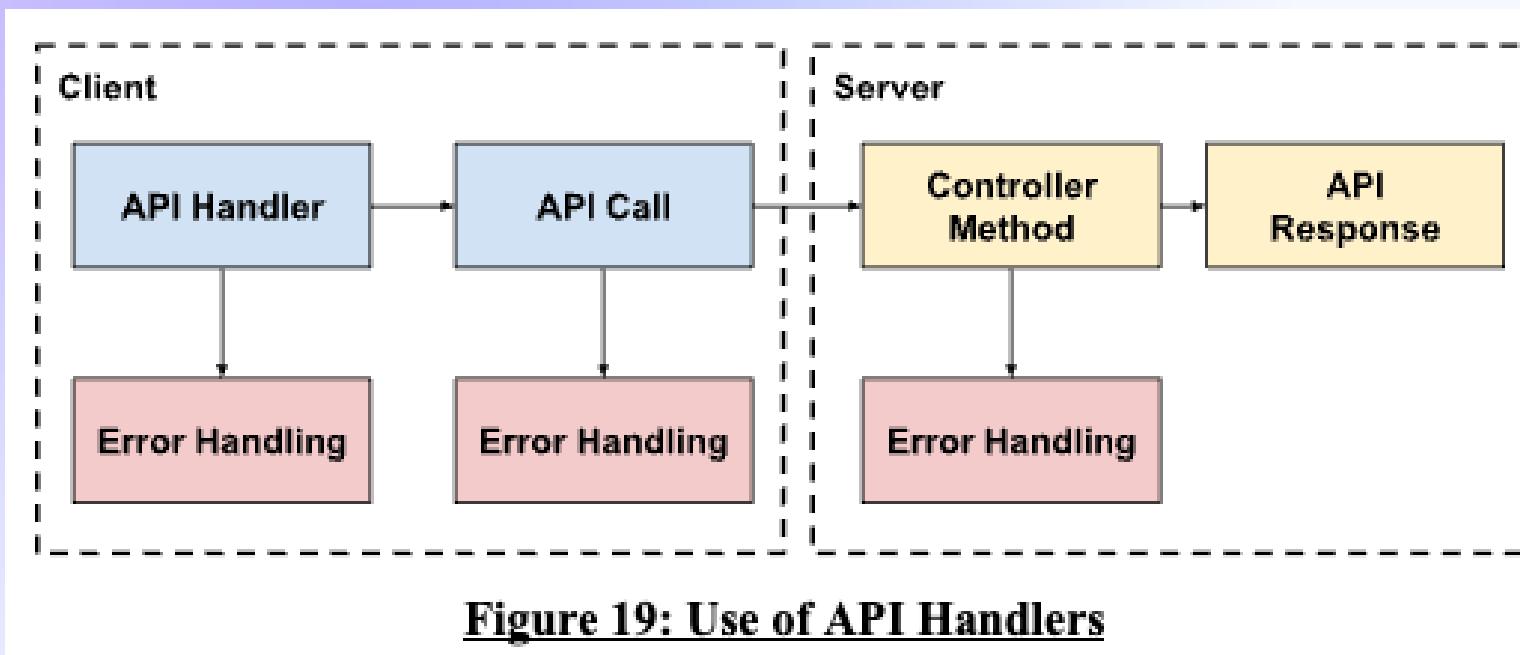
- ✓ Component Reusability
- ✓ Customizable to fit different functionalities



```
25+ usages ▲ iujinwee +1
const BaseButton = ({  
  title,  
  primary,  
  secondary,  
  width,  
  height,  
  link,  
  onPress,  
  textSize,  
  ...props  
}) => {  
  const { theme } = useTheme();  
  const StyledPressable: ForwardRef<InferRef<  
    = styled(Pressable);  
  
  1 usage ▲ iujinwee  
  const onPressHandler = () : void => {...};  
  
  return (  
    <StyledPressable  
      onPress={onPress || onPressHandler}  
      className={`border-2 border-primary rounded-lg justify-center items-center  
      font-RobotoBold hover:scale-150`}  
      style={{...}}  
    >  
    <Text  
      className="font-RobotoBlack text-base"  
      style={{...}}  
    >  
    {title}  
    </Text>  
    </StyledPressable>  
  );  
};
```

Design Patterns

Chain of Responsibility Pattern



✓ **Facilitates Error-handling**

✓ **Better code maintainability**

```
useEffect( effect: () : void  => {
  1 usage  ± JosephT631
    async function prepare(): Promise<void> {
      try {
        await fetchAccount(notes);
        await fetchAllData(notes);
      } catch (e) {
        console.warn(e);
      } finally {
        // Tell the application to render
        setAppIsReady( value: true);
      }
    }
    if (!appIsReady) {
      prepare();
    }
  }, [ deps: [idStore] ] );
```

Design Patterns

Command Pattern

- ✓ **Encapsulates different actions**
- ✓ **Trigger actions without knowledge of specific details**

```
3 usages  ↗ iujinwee
const inputHandler = (input, field) : void => {
  setAccountInfo( value: (prevState : {...} ) => ({ ...prevState, [field]: input }));
  setShowAlert( value: false);
};
```

Traceability

Use Case Diagram

Authentication part - forget password

Forget Password

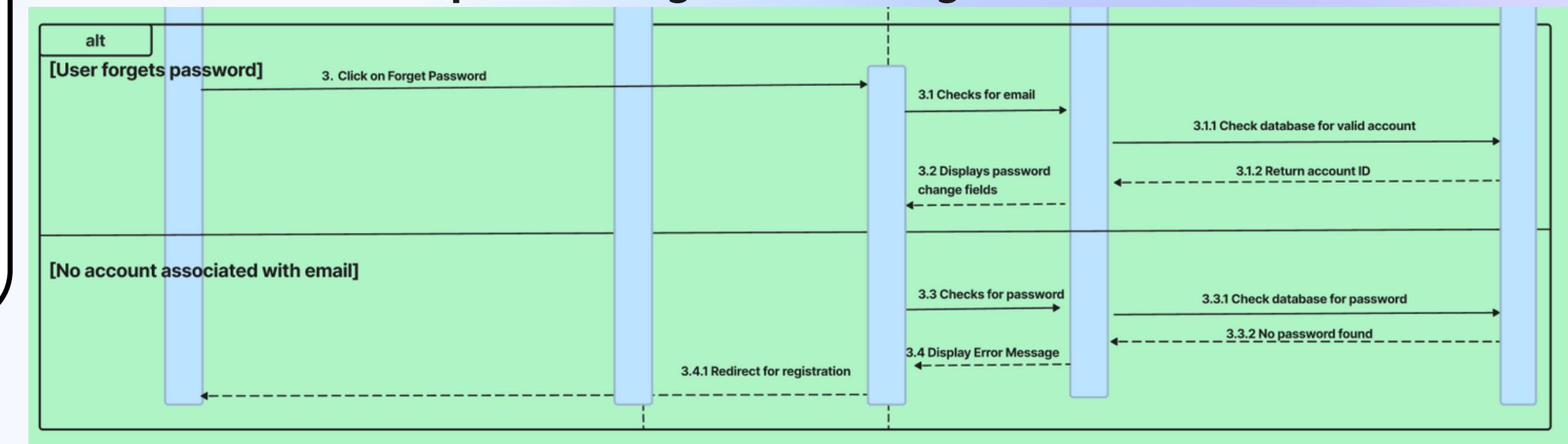
Description: The app must allow the current user to change their own password.

Input: Email, Customer type

Output: Password successfully changed



Sequence Diagram for 'Forget Password'



Authentication part
- forget password

Traceability

UI Mockup

Use Case Description

Use Case ID:	A0005		
Use Case Name:	Forget Password		
Created By:	CodeCrafters	Last Updated By:	CodeCrafters
Date Created:	10th Sep 2023	Date Last Updated:	11th Sep 2023
Actor:	User		
Description:	On the login page, if the user does not remember their password, they can reset it through email.		
Preconditions:	The user has an account.		
Postconditions:	The user's current password is changed to the new password.		
Priority:	-		
Frequency of Use:	By the intention of the user.		
Flow of Events:	<ol style="list-style-type: none">1. The user presses the "Forget Password" button.2. App verifies email and the user presses on "Change Password" button.3. The user is prompted to enter a new password.4. The user enters a new password.5. App checks if the new password is valid.6. App replaces the old password with the new one in the database.7. Password is reset.		
Alternative Flows:	<ol style="list-style-type: none">7. App checks that the password is invalid.8. Back to step 5.		
Exceptions:	<ol style="list-style-type: none">1. The user's email is not linked to any valid account.2. The user does not have access to his/her own email.		
Includes:	-		
Special Requirements:	-		
Assumptions:	The email belongs to the user's account.		
Notes and Issues:	-		

The UI mockup shows the 'Forget Password' screen of the MoveInLo! app. At the top, there is a dark blue header bar with the time '9:29', signal strength, battery level, a back arrow labeled 'Back', and the app name 'MoveInLo!'. Below the header is a light-colored form area with the title 'Forget Password' at the top center. The form fields include:

- 'Account Type' dropdown set to 'Customer'.
- 'Email' input field containing 'Abc@test.com'.
- 'Search' button to the right of the email field.
- 'New Password' input field.
- 'Re-enter new password' input field.
- 'Submit' button at the bottom right.

Authentication part - forget password

Traceability

Implementation

```

const validationHandler = (field) : any => {
  const validEmail : boolean =
    accountInfo.email !== "" &&
    accountInfo.email.includes("@") &&
    accountInfo.email.includes(".com");

  const validType : boolean = accountInfo.type !== "";

  const validPassword : boolean =
    accountInfo.newPassword.length >= 8 &&
    /[A-Z]/.test(accountInfo.newPassword) &&
    /[0-9]/.test(accountInfo.newPassword) &&
    /[!@#$%^&*()_+]/.test(accountInfo.newPassword);

  const matchingPassword : boolean =
    accountInfo.newPassword === accountInfo.passwordCheck;

  switch (field) {
    case "type":
      return invalidHandler(validType, field);

    case "email":
      return invalidHandler(validEmail, field);

    case "newPassword":
      return invalidHandler(validPassword, field);

    case "passwordCheck":
      return invalidHandler(matchingPassword, field);
  }
};

const inputHandler = (input, field) : void => {...};

const invalidHandler = (bool, field) => {...};

const resetHandler = () : void => {...};

const validationHandler = (field) : any => {...};

const searchHandler = async () : Promise<void> => {...};

const submitHandler = async () : Promise<void> => {
  const validNewPassword : any = validationHandler({ field: "newPassword" });
  const matchingPassword : any = validationHandler({ field: "passwordCheck" });

  if (validNewPassword && matchingPassword) {
    if (accountId) {
      const updatedAccount = await postNewPassword( req: {
        id: accountId,
        body: accountInfo,
      });
      if (updatedAccount) {
        setModalVisible( value: true );
      } else {
        setShowAlert( value: true );
      }
    } else {
      setShowAlert( value: true );
    }
  }
};

```

Validating inputs &
Error handling

API Call to update
password

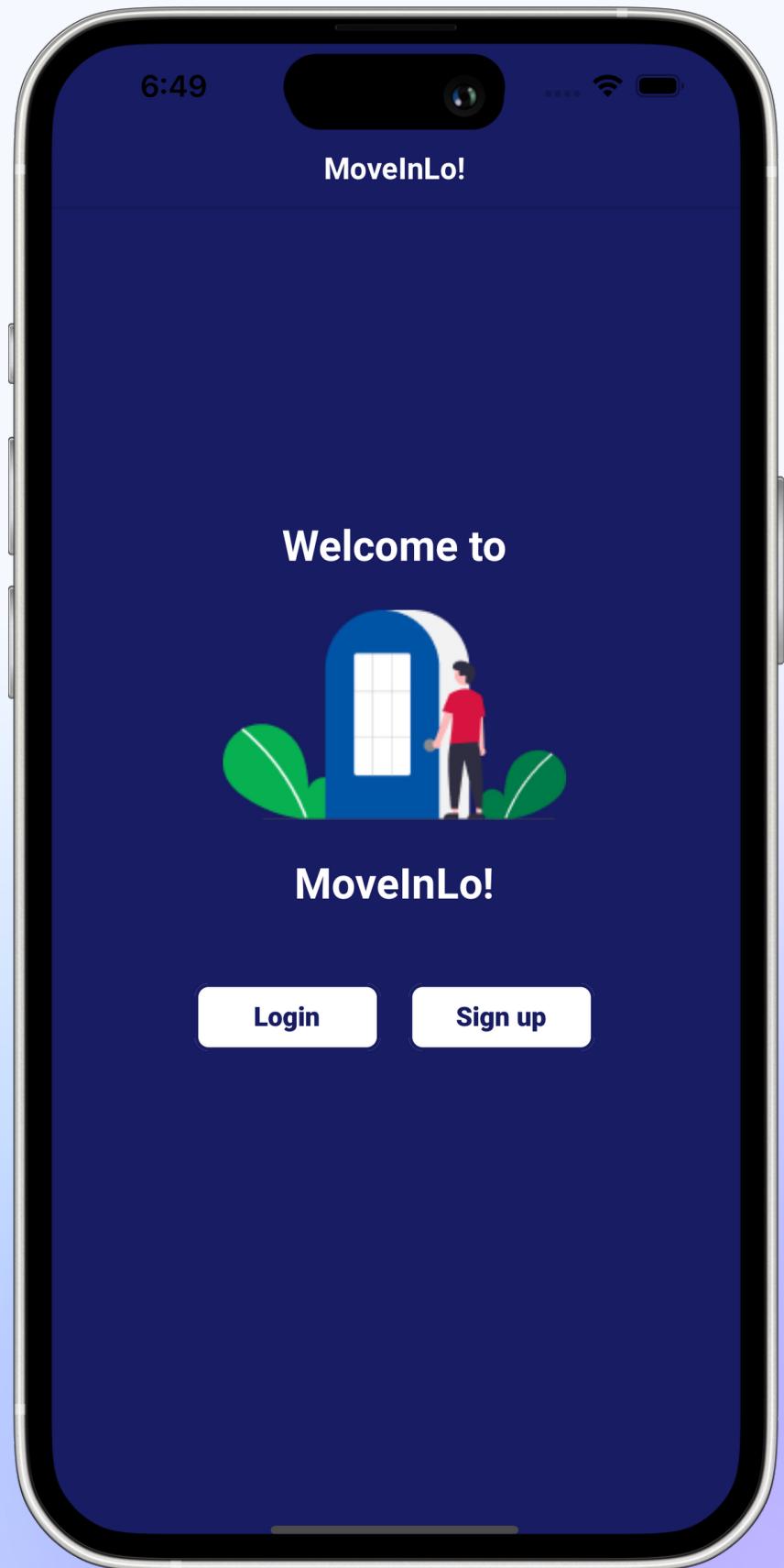
Testing

Forget password

Email	New Password	Expected Output	Actual Output
Email in database	Valid password	Success	Success
Email not in database	Valid password	Error	Error: Account doesn't exists.
Email in database	Invalid password	Error	Error: Password must be at least 8 characters with 1 capital letter, 1 number and 1 special character.
Email in database	Password doesn't match	Error	Error: Both passwords must match
Missing	N/A	Error	Error: You have missing or invalid inputs!
Email in database	Missing	Error	Error: You have missing or invalid inputs!



4. Demo



Thanks!

