Bhoomitalli Real Estate Platform: Property Listing Wizard

Step Management Analysis & Documentation

1. Overview of Wizard Structure

The Property Listing Wizard is a multi-step form system designed to guide users through the process of creating different types of property listings. It supports multiple "flows" for different property categories (Residential, Commercial, Land) and listing types (Rent, Sale, PG/Hostel, Co-working, Flatmates).

1.1 Core Architecture

The wizard is built with a modular architecture that consists of:

- **Step Definitions**: Individual form steps with metadata (ID, title, icon)
- **Flow Configurations**: Predefined sequences of steps for each property type
- **Dynamic Step Management**: Logic to conditionally show/hide steps based on property type
- **Navigation System**: Controls movement between steps including URL synchronization

1.2 Entry Points

The main entry point for the wizard is the `PropertyForm` component (`src/modules/owner/components/property/wizard/PropertyForm/index.tsx`). This component:

- Orchestrates the entire form experience
- Handles conditional rendering of steps
- Manages property type detection
- Controls navigation between steps
- Integrates with form state management

2. Step Management Logic

2.1 Step Definition System

Steps are defined in two primary locations:

Individual step definitions (`common.ts`):

```
 export const STEP_DEFINITIONS = {
  details: {
    id: 'details',
    title: 'Basic Details',
    icon: Home,
    description: 'Property type and details'
  },
  // Additional step definitions...
};
```

1.

Flow-specific step sequences (`flows.ts`):

```
 export const FLOW_STEPS = {
  // Residential Rent flow (default)
  RESIDENTIAL_RENT: [
    STEP_DEFINITIONS.details,
    STEP_DEFINITIONS.location,
    STEP_DEFINITIONS.rental,
    STEP_DEFINITIONS.features,
    STEP_DEFINITIONS.review
  ],
  // Additional flow definitions...
};
```

2.

2.2 Conditional Step Visibility

The primary mechanism for controlling which steps are visible is implemented in the `useStepNavigation` hook's `getVisibleSteps` function:

```
const getVisibleSteps = useCallback(() => {
  return STEPS.map(step => ({
    ...step,
    hidden:
      // Rental step hidden for non-rental flows
      (step.id === 'rental' && (isSaleMode || isPGHostelMode ||
```

```
                    isCommercialSaleMode || isCoworkingMode || isLandSaleMode ||
                    isFlatmatesMode)) ||
    // Sale step hidden for non-sale flows
    (step.id === 'sale' && (!isSaleMode || isPGHostelMode || isCommercialRentMode ||
                    isCommercialSaleMode || isCoworkingMode || isLandSaleMode ||
                    isFlatmatesMode)) ||
    // Additional condition checks for other steps...
  }));
}, [
  STEPS,
  isSaleMode,
  isPGHostelMode,
  isCommercialRentMode,
  isCommercialSaleMode,
  isCoworkingMode,
  isLandSaleMode,
  isFlatmatesMode
]);
```

This function applies a set of boolean conditions to determine if each step should be hidden based on the current property type and listing mode.

2.3 Flow Detection Logic

The system dynamically detects which flow to use based on several heuristics:

1. **URL Path Analysis**: Examines the current URL path for keywords
2. **Form Value Inspection**: Checks the current form values for property and listing types
3. **Explicit Props**: Uses passed props when provided

This detection logic is implemented in multiple `useMemo` hooks within the `PropertyForm` component:

```
// Example of PG/Hostel mode detection
const isPGHostelMode = useMemo(() => {
  if (derivedAdType) {
    return derivedAdType.toLowerCase() === 'pghostel';
  }

  // Check URL path for keywords
  const urlPath = window.location.pathname.toLowerCase();
  return urlPath.includes('pghostel');
}, [derivedAdType]);
```

```
// Similar patterns for other flow types...
```

2.4 Step Sequence Selection

Once the property type is detected, the appropriate step sequence is selected:

```
// Determine which flow steps to use based on property type
const flowSteps = useMemo(() => {
  if (isPGHostelMode) {
    return FLOW_STEPS.RESIDENTIAL_PGHOSTEL;
  } else if (isCommercialRentMode) {
    return FLOW_STEPS.COMMERCIAL_RENT;
  }
  // Additional flow selections...
  else {
    return FLOW_STEPS.RESIDENTIAL_RENT; // Default
  }
}, [
  derivedAdType,
  isPGHostelMode,
  isCommercialRentMode,
  isCommercialSaleMode,
  isCoworkingMode,
  isLandSaleMode,
  isFlatmatesMode
]);
```

3. File & Component Mapping

3.1 Core Files

File	Responsibility
`src/modules/owner/components/property/wizard/constants/common.ts`	Defines individual step metadata including IDs, titles, and icons
`src/modules/owner/components/property/wizard/constants/flows.ts`	Defines flow-specific step sequences

`src/modules/owner/components/property/wizard/PropertyForm/index.tsx`	Main wizard component that orchestrates the entire form experience
`src/modules/owner/components/property/wizard/hooks/usePropertyForm.ts`	Central hook that manages form state and operations
`src/modules/owner/components/property/wizard/hooks/usePropertyFormNavigation.ts`	Handles navigation between steps with URL synchronization
`src/modules/owner/components/property/wizard/PropertyForm/hooks/useStepNavigation.ts`	Manages property-type specific navigation logic
`src/modules/owner/components/property/wizard/PropertyForm/components/StepNavigation.tsx`	UI component for step navigation buttons
`src/modules/owner/components/property/wizard/types.ts`	TypeScript definitions for form data structure

3.2 Component Breakdown

3.2.1 PropertyForm (Main Wizard Component)

- **Responsibility**: Orchestrates the entire form experience
- **Key Functions**:
  - Property type detection
  - Form initialization
  - Step sequence selection
  - Step rendering
  - Navigation control

3.2.2 Step Definition System

- **common.ts**: Central registry of all possible steps
- **flows.ts**: Predefines sequences of steps for each property type

3.2.3 Navigation System

- **usePropertyFormNavigation.ts**: General navigation logic with URL syncing
- **useStepNavigation.ts**: Property-type specific navigation with step visibility

3.2.4 Form State Management

- **usePropertyForm.ts**: Central form state management
- **usePropertyFormState.ts**: Form initialization and persistence
- **usePropertyFormOperations.ts**: CRUD operations for form data

4. Code Flow Analysis

4.1 Initialization Flow

1. **PropertyForm Component Loads**
    - Analyzes URL and props to determine property type
    - Initializes form with appropriate data structure
    - Selects the correct flow steps sequence
2. **usePropertyForm Hook Initialization**
    - Sets up form state
    - Establishes validation rules
    - Connects to backend services
3. **Step Sequence Selection**
    - Based on detected property type, selects the appropriate step sequence from `FLOW_STEPS`
    - Initializes navigation with the correct steps
4. **Initial Step Determination**
    - Attempts to get initial step from URL
    - Falls back to localStorage for returning users
    - Uses step 1 as default for new forms

4.2 Navigation Flow

1. **User Clicks Next/Previous**
    - `StepNavigation` component handles button click

2. **Step Validation (Next only)**
    - Current step form data is validated before proceeding
3. **Step Calculation**
    - For standard navigation: Simple increment/decrement
    - For flow-specific navigation: Consults the active sequence

```
const activeSequence = getActiveSequence();
const currentIndex = activeSequence.indexOf(currentStepId);
const nextStepId = activeSequence[currentIndex + 1];
```

4.
5. **URL Synchronization**
    - Updates URL to reflect new step
    - Preserves property type and listing type in URL
6. **Form Persistence**
    - Saves current form data to localStorage
7. **Step State Update**
    - Updates the `currentStep` state to trigger re-render

4.3 Step Rendering Flow

1. **Step Content Determination**
    - `FormContent` component receives current step index
    - Looks up corresponding step definition
2. **Step Visibility Filtering**
    - `getVisibleSteps` applies conditional logic to hide irrelevant steps
3. **Step UI Rendering**
    - Renders the appropriate form section based on current step ID

5. Change Strategy Guide

5.1 Adding a New Step

To add a new step to the wizard, follow these steps:

Define the Step in common.ts:

```
export const STEP_DEFINITIONS = {
  // Existing steps...

  // Add your new step
  new_step: {
    id: 'new_step',
    title: 'New Step Title',
    icon: YourIconComponent,
    description: 'Description of new step'
  }
};
```

Add the Step to Relevant Flow Sequences in flows.ts:

```
export const FLOW_STEPS = {
  // Modify existing flows to include your new step
  RESIDENTIAL_RENT: [
    STEP_DEFINITIONS.details,
    STEP_DEFINITIONS.location,
    STEP_DEFINITIONS.rental,
    STEP_DEFINITIONS.features,
    STEP_DEFINITIONS.new_step, // Add your step here
    STEP_DEFINITIONS.review
  ],
  // Update other flows as needed
};
```

1. **Create the Step Component**:
    - Create a new component in the `sections` folder
    - Follow existing patterns for form section components

Update Step Content Rendering in FormContent.tsx:

```
// In the switch statement for step content
case 'new_step':
  return (
    <NewStepComponent
      form={form}
      // Other props...
    />
```

```
  );
```

Add Step Visibility Logic in useStepNavigation.ts:

```
 // In the getVisibleSteps function
return STEPS.map(step => ({
  ...step,
  hidden:
    // Existing conditions...

    // Add condition for your new step
    (step.id === 'new_step' && !shouldShowNewStep)
}));
```

5.2 Removing or Skipping an Existing Step

To remove a step from a flow:

Remove from Flow Sequence:

```
 // In flows.ts
export const FLOW_STEPS = {
  RESIDENTIAL_RENT: [
    STEP_DEFINITIONS.details,
    STEP_DEFINITIONS.location,
    // STEP_DEFINITIONS.rental, // Remove this step
    STEP_DEFINITIONS.features,
    STEP_DEFINITIONS.review
  ],
  // Update other flows as needed
};
```

Add Hide Condition (Alternative to Removal):

```
 // In useStepNavigation.ts - getVisibleSteps
return STEPS.map(step => ({
  ...step,
  hidden:
    // Existing conditions...

    // Add condition to hide step
```

```
    (step.id === 'step_to_hide' && someCondition)
}));
```

Skip in Navigation Logic (Alternative):

```
 // In useStepNavigation.ts - handleNextStep
if (currentStepId === 'step_before_skipped') {
  // Skip the next step and jump to the one after
  const skipToStepId = activeSequence[currentIndex + 2]; // Skip one step
  // Handle the skip...
}
```

5.3 Modifying an Existing Step

To modify an existing step:

Update Step Definition:

```
 // In common.ts
export const STEP_DEFINITIONS = {
  // Modify existing step
  existing_step: {
    id: 'existing_step',
    title: 'Updated Step Title', // Change title
    icon: NewIcon, // Change icon
    description: 'Updated description' // Change description
  },
  // Other steps...
};
```

1.
2. **Update Step Component**: Modify the corresponding component in the `sections` folder

3. **Update Visibility Logic**: If necessary, adjust conditions in `getVisibleSteps`

5.4 Creating a New Flow

To create an entirely new property type flow:

Define Flow Sequence:

```
 // In flows.ts
export const FLOW_STEPS = {
  // Existing flows...

  // Add new flow
  NEW_FLOW: [
    STEP_DEFINITIONS.details,
    STEP_DEFINITIONS.location,
    STEP_DEFINITIONS.new_step,
    STEP_DEFINITIONS.features,
    STEP_DEFINITIONS.review
  ],
};
```

1.

Add Flow Detection Logic:

```
 // In PropertyForm/index.tsx
const isNewFlowMode = useMemo(() => {
  // Check form values
  const formValues = form.getValues();
  const category = (formValues.propertyCategory || '').toLowerCase();
  const listingType = (formValues.listingType || '').toLowerCase();

  // Check URL for indicators
  const urlPath = window.location.pathname.toLowerCase();
  const isNewFlowFromUrl = urlPath.includes('new-flow-keyword');

  // Combined check
  return (category === 'new_category' && listingType === 'new_type') ||
         isNewFlowFromUrl;
}, [form]);
```

2.

Update Flow Selection Logic:

```
 // In PropertyForm/index.tsx
const flowSteps = useMemo(() => {
  if (isNewFlowMode) {
    return FLOW_STEPS.NEW_FLOW;
  }
```

```
  // Other flow selections...
}, [
  // Existing dependencies...
  isNewFlowMode
]);
```

3.

Update Step Visibility Logic:

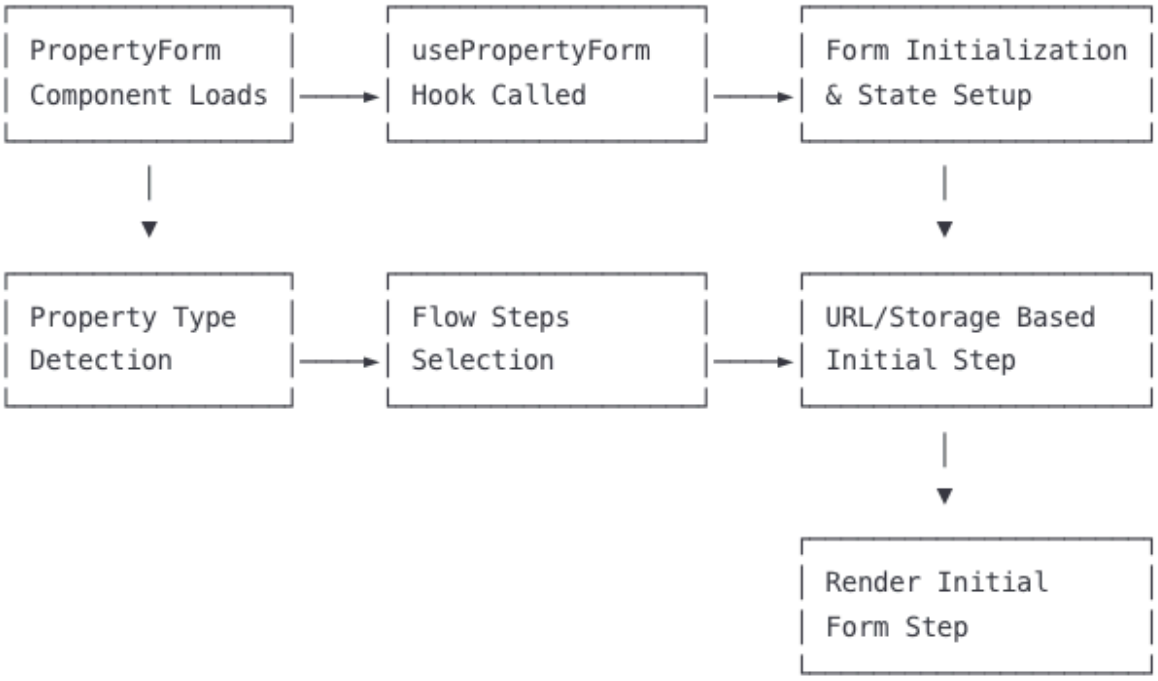
```
 // In useStepNavigation.ts - getVisibleSteps
return STEPS.map(step => ({
  ...step,
  hidden:
    // Existing conditions...

    // Add conditions for new flow
    (step.id === 'new_step' && !isNewFlowMode)
}));
```

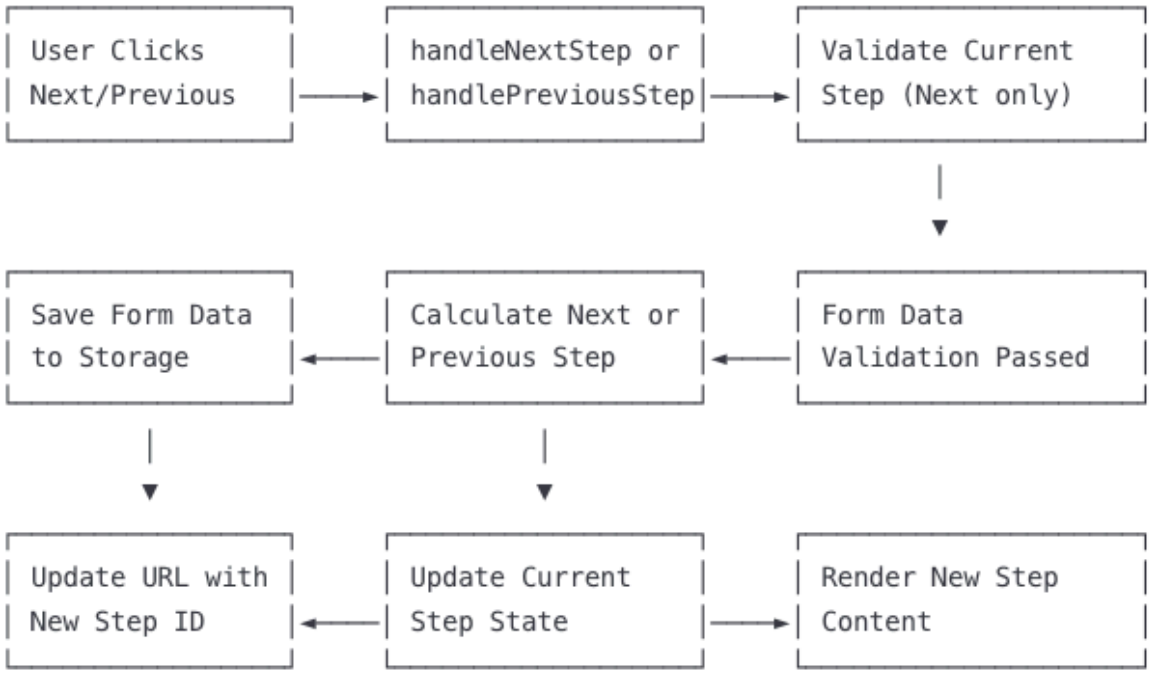
4.

6. Flow Diagrams

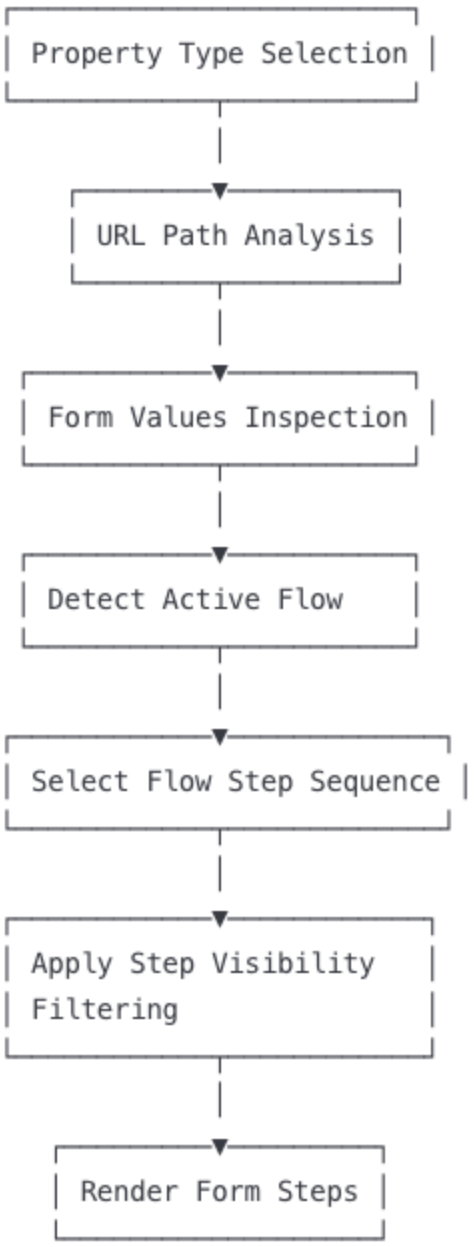
6.1 Wizard Initialization Flow



6.2 Step Navigation Flow



6.3 Property Type Flow Selection



7. Conclusion

The Bhoomitalli Real Estate Platform's Property Listing Wizard employs a sophisticated, flexible architecture for managing different property listing flows. The key to this system is:

1. **Centralized Step Definitions**: All steps are defined in one place with consistent metadata

2. **Flow-Specific Sequences**: Each property type has a predefined sequence of steps
3. **Dynamic Flow Detection**: The system intelligently detects which flow to use
4. **Conditional Visibility**: Steps are shown or hidden based on complex conditional logic
5. **URL Synchronization**: The current step is always reflected in the URL for direct access

This architecture makes it relatively straightforward to add, remove, or modify steps in existing flows, or to create entirely new flows for new property types.