PropertySummary Refactoring Implementation Plan

This document outlines the step-by-step implementation plan for refactoring the PropertySummary component. Each phase focuses on specific components and includes expected outcomes to track progress.

Phase 1: Core Infrastructure

1.1. Set Up Folder Structure

- Create the folder structure as defined in the architecture document
- Set up the basic files and exports

1.2. Create Base Components

- Implement `BaseSummaryFlow.tsx` abstract class
- Implement `SummaryHeader.tsx` component
- Implement `SummaryContent.tsx` component
- Implement `PropertySummaryPage.tsx` entry point

1.3. Set Up Registry and Hooks

- Create `sectionComponentRegistry.ts` with initial mappings
- Migrate existing hooks (`useFlowDetection.ts`, `usePropertyData.ts`, `usePropertyTitle.ts`)
- Create `useSummaryData.ts` hook

1.4. Implement Field Components

- Create base field components in `/components/fields/`:
  - `FieldText.tsx` for text values
  - `FieldBoolean.tsx` for yes/no values
  - `FieldCurrency.tsx` for monetary values
  - `FieldArea.tsx` for area values
  - `FieldList.tsx` for array values

1.5. Create Common Section Components

- Implement `LocationSection.tsx` (common to all flows)
- Implement base `SummarySection.tsx` wrapper component

- Port existing `DescriptionSection.tsx`

1.6. Set Up Flow Factory

- Implement `FlowFactory.tsx` with mappings to flow components (stub implementations initially)

Expected Outcome Phase 1:

- Complete folder structure with all necessary files
- Basic rendering pipeline that can detect flow type and display placeholder content
- Field components that can render different data types
- Registry with placeholder mappings for all section types
- No actual flow-specific functionality yet, but the infrastructure is ready for implementation

Phase 2: Residential Flows Implementation

2.1. Residential Sale Flow

- Implement `ResidentialSaleSummary.tsx` flow component
- Implement required section components:
  - `PropertyDetailsSection.tsx`
  - `SaleDetailsSection.tsx`
  - `AmenitiesSection.tsx`
- Map sections in the registry
- Test with sample data

2.2. Residential Rent Flow

- Implement `ResidentialRentSummary.tsx` flow component
- Implement additional required section components:
  - `RentalDetailsSection.tsx`
- Map sections in the registry
- Test with sample data

2.3. Residential PG/Hostel Flow

- Implement `ResidentialPGHostelSummary.tsx` flow component
- Implement required section components:
  - `RoomDetailsSection.tsx`
  - `PGDetailsSection.tsx`
- Map sections in the registry
- Test with sample data

2.4. Residential Flatmates Flow

- Implement `ResidentialFlatmatesSummary.tsx` flow component
- Implement required section components:
  - `FlatmateDetailsSection.tsx`
- Map sections in the registry
- Test with sample data

Expected Outcome Phase 2:

- Fully functional residential flows with all specific section components
- Each flow displays the correct sections in the right order
- All residential-specific fields are properly rendered
- Flow detection works correctly for all residential property types
- Registry is updated with actual section component implementations
- Visual styling is consistent across all residential flows

Phase 3: Commercial Flows Implementation

3.1. Commercial Sale Flow

- Implement `CommercialSaleSummary.tsx` flow component
- Implement required section components:
  - `CommercialBasicDetailsSection.tsx`
  - `CommercialSaleDetailsSection.tsx`
  - `CommercialFeaturesSection.tsx`
- Map sections in the registry
- Test with sample data

3.2. Commercial Rent Flow

- Implement `CommercialRentSummary.tsx` flow component
- Implement additional required section components:
  - `CommercialRentalDetailsSection.tsx`
- Map sections in the registry
- Test with sample data

3.3. Commercial Coworking Flow

- Implement `CommercialCoworkingSummary.tsx` flow component
- Implement required section components:
  - `CoworkingBasicDetailsSection.tsx`
  - `CoworkingDetailsSection.tsx`
- Map sections in the registry
- Test with sample data

Expected Outcome Phase 3:

- Fully functional commercial flows with all specific section components
- Commercial property fields and sections render correctly
- Field formatting is appropriate for commercial data (e.g., pricing formats, area units)
- Registry is updated with all commercial section component implementations
- Commercial flows can be detected and rendered correctly based on the JSON data

Phase 4: Land Flow Implementation

4.1. Land Sale Flow

- Implement `LandSaleSummary.tsx` flow component
- Implement required section components:
  - `LandDetailsSection.tsx`
  - `LandFeaturesDetailsSection.tsx`
- Map sections in the registry
- Test with sample data

Expected Outcome Phase 4:

- Fully functional land flow with specific section components
- Land-specific fields render correctly (plot area, dimensions, approvals, etc.)
- Registry is updated with land section component implementations
- Land flow can be detected and rendered correctly based on the JSON data

Phase 5: Integration and Testing

5.1. Comprehensive Testing

- Test all flows with real data
- Test empty data cases
- Test edge cases
- Fix any issues discovered during testing

5.2. Performance Optimization

- Review component rendering performance
- Implement memoization where beneficial
- Optimize section rendering logic

5.3. Documentation and Handover

- Document the new component architecture
- Create usage examples for each flow
- Update any related documentation

Expected Outcome Phase 5:

- All flows rendering correctly with actual production data
- Empty/null values handled gracefully
- No rendering issues or visual glitches
- Component performance is optimized, especially for large data sets
- Complete documentation for the new architecture
- A smooth transition to the new component for all property types

Implementation Details and Code Samples

Below are key code samples for the first implementation phase to help you get started:

BaseSummaryFlow.tsx

```
//
src/modules/owner/components/property/wizard/sections/PropertySummary/flows/base/Bas
eSummaryFlow.tsx
// Version: 1.0.0
// Last Modified: 21-05-2025 10:30 IST
// Purpose: Base abstract class for all flow components

import React from 'react';
import { getSectionWithMetadata } from '../../registry/sectionComponentRegistry';
import { FormData } from '../../../../types';

export interface BaseSummaryFlowProps {
  formData: FormData;
}

export abstract class BaseSummaryFlow extends
React.Component<BaseSummaryFlowProps> {
  // Abstract method that must be implemented by child classes
  abstract getSectionIds(): string[];

  render() {
    const { formData } = this.props;
    const sectionIds = this.getSectionIds();

    return (
      <div className="space-y-6">
        {sectionIds.map(sectionId => {
          const { Component, metadata } = getSectionWithMetadata(sectionId);
          const sectionData = formData.steps?.[sectionId] || {};

          return (
            <div key={sectionId} className="mb-6">
              <h3 className="text-lg font-semibold flex items-center gap-2 mb-4">
                {metadata.icon && React.createElement(metadata.icon, { className: "h-5 w-5" })}
                {metadata.name}
```

```
            </h3>
            <Component
              data={sectionData}
              flowType={formData.flow?.category}
              listingType={formData.flow?.listingType}
            />
          </div>
        );
      })}
    </div>
  );
 }
}
```

Section Component Registry

```
//
src/modules/owner/components/property/wizard/sections/PropertySummary/registry/section
ComponentRegistry.ts
// Version: 1.0.0
// Last Modified: 21-05-2025 10:35 IST
// Purpose: Maps section IDs to components

import { ComponentType } from 'react';
import { STEP_METADATA } from '../../../../../constants/flows';

// Import placeholder component for initial setup
import { PlaceholderSection } from '../sections/PlaceholderSection';

// Map of section IDs to their corresponding components
// Initially we'll use placeholders, and replace them as we implement each section
export const SECTION_COMPONENT_REGISTRY: Record<string, ComponentType<any>>
= {
  // Residential sections
  'res_sale_basic_details': PlaceholderSection,
  'res_sale_location': PlaceholderSection,
  'res_sale_sale_details': PlaceholderSection,
  'res_sale_features': PlaceholderSection,

  'res_rent_basic_details': PlaceholderSection,
  'res_rent_location': PlaceholderSection,
  'res_rent_rental': PlaceholderSection,
  'res_rent_features': PlaceholderSection,

  'res_flat_basic_details': PlaceholderSection,
  'res_flat_location': PlaceholderSection,
  'res_flat_flatmate_details': PlaceholderSection,
```

```
  'res_flat_features': PlaceholderSection,

  'res_pg_basic_details': PlaceholderSection,
  'res_pg_location': PlaceholderSection,
  'res_pg_pg_details': PlaceholderSection,
  'res_pg_features': PlaceholderSection,

  // Commercial sections
  'com_sale_basic_details': PlaceholderSection,
  'com_sale_location': PlaceholderSection,
  'com_sale_sale_details': PlaceholderSection,
  'com_sale_features': PlaceholderSection,

  'com_rent_basic_details': PlaceholderSection,
  'com_rent_location': PlaceholderSection,
  'com_rent_rental': PlaceholderSection,
  'com_rent_features': PlaceholderSection,

  'com_cow_basic_details': PlaceholderSection,
  'com_cow_location': PlaceholderSection,
  'com_cow_coworking_details': PlaceholderSection,
  'com_cow_features': PlaceholderSection,

  // Land sections
  'land_sale_basic_details': PlaceholderSection,
  'land_sale_location': PlaceholderSection,
  'land_sale_land_features': PlaceholderSection
};

// Function to get a section component with its metadata
export function getSectionWithMetadata(sectionId: string) {
  const Component = SECTION_COMPONENT_REGISTRY[sectionId];
  const metadata = STEP_METADATA[sectionId] || {
    name: sectionId.replace(/_/g, ' ').replace(/\b\w/g, l => l.toUpperCase()),
    icon: null
  };

  if (!Component) {
    console.warn(`No component found for section ID: ${sectionId}`);
    return {
      Component: PlaceholderSection,
      metadata
    };
  }

  return {
    Component,
    metadata
```

```
  };
}
```

Field Text Component Example

```
//
src/modules/owner/components/property/wizard/sections/PropertySummary/components/fiel
ds/FieldText.tsx
// Version: 1.0.0
// Last Modified: 21-05-2025 10:40 IST
// Purpose: Field component for text values

import React from 'react';

interface FieldTextProps {
  label: string;
  value?: string | number;
  className?: string;
}

export const FieldText: React.FC<FieldTextProps> = ({
  label,
  value,
  className = ""
}) => {
  // Don't render if no value
  if (value === undefined || value === null || value === '') {
    return null;
  }

  return (
    <div className={`flex justify-between py-2 border-b border-gray-100 ${className}`}>
      <dt className="text-sm font-medium text-gray-500">{label}</dt>
      <dd className="text-sm text-gray-900 font-medium">{value}</dd>
    </div>
  );
};
```

Initial Phase 2 Example: ResidentialSaleSummary

```
//
src/modules/owner/components/property/wizard/sections/PropertySummary/flows/Residenti
alSaleSummary.tsx
// Version: 1.0.0
// Last Modified: 28-05-2025 10:30 IST
// Purpose: Flow component for Residential Sale properties
```

```
import { BaseSummaryFlow } from './base/BaseSummaryFlow';

export class ResidentialSaleSummary extends BaseSummaryFlow {
  getSectionIds(): string[] {
    return [
      'res_sale_basic_details',
      'res_sale_location',
      'res_sale_sale_details',
      'res_sale_features'
    ];
  }
}
```

PropertyDetailsSection Implementation Example

```
//
src/modules/owner/components/property/wizard/sections/PropertySummary/sections/Propert
yDetailsSection.tsx
// Version: 1.0.0
// Last Modified: 28-05-2025 11:30 IST
// Purpose: Section component for property details

import React from 'react';
import { FieldText } from '../components/fields/FieldText';
import { FieldArea } from '../components/fields/FieldArea';
import { FieldBoolean } from '../components/fields/FieldBoolean';

interface PropertyDetailsSectionProps {
  data: any;
  flowType?: string;
  listingType?: string;
}

export const PropertyDetailsSection: React.FC<PropertyDetailsSectionProps> = ({
  data,
  flowType,
  listingType
}) => {
  if (!data) return null;

  return (
    <div className="space-y-2">
      {/* Property Type */}
      <FieldText label="Property Type" value={data.propertyType} />

      {/* BHK Type (for apartments) */}
      {data.propertyType === 'Apartment' && (
```

```
      <FieldText label="BHK Type" value={data.bhkType} />
    )}

    {/* Bedrooms & Bathrooms */}
    <div className="grid grid-cols-2 gap-4">
      <FieldText label="Bedrooms" value={data.bedrooms} />
      <FieldText label="Bathrooms" value={data.bathrooms} />
    </div>

    {/* Floor Information */}
    {data.floor !== undefined && data.totalFloors !== undefined && (
      <FieldText
        label="Floor"
        value={`${data.floor} out of ${data.totalFloors}`}
      />
    )}

    {/* Area Information */}
    {data.builtUpArea && (
      <FieldArea
        label="Built-up Area"
        value={data.builtUpArea}
        unit={data.builtUpAreaUnit || 'sqft'}
      />
    )}

    {/* Furnishing Status */}
    <FieldText label="Furnishing Status" value={data.furnishingStatus} />

    {/* Property Age */}
    <FieldText label="Property Age" value={data.propertyAge} />

    {/* Balconies */}
    <FieldText label="Balconies" value={data.balconies} />

    {/* Facing */}
    <FieldText label="Facing" value={data.facing} />

    {/* Property Condition */}
    <FieldText label="Property Condition" value={data.propertyCondition} />

    {/* Boolean features */}
    <div className="grid grid-cols-2 gap-4">
      <FieldBoolean label="Has Balcony" value={data.hasBalcony} />
      <FieldBoolean label="Has AC" value={data.hasAC} />
    </div>
  </div>
);
```

```
};

// Update the registry with this implementation
// In sectionComponentRegistry.ts:
// import { PropertyDetailsSection } from '../sections/PropertyDetailsSection';
//
// Update SECTION_COMPONENT_REGISTRY:
// 'res_sale_basic_details': PropertyDetailsSection,
// 'res_rent_basic_details': PropertyDetailsSection,
```