Byte Property Management

Presented by

Anthony, Evgeny, and Richard

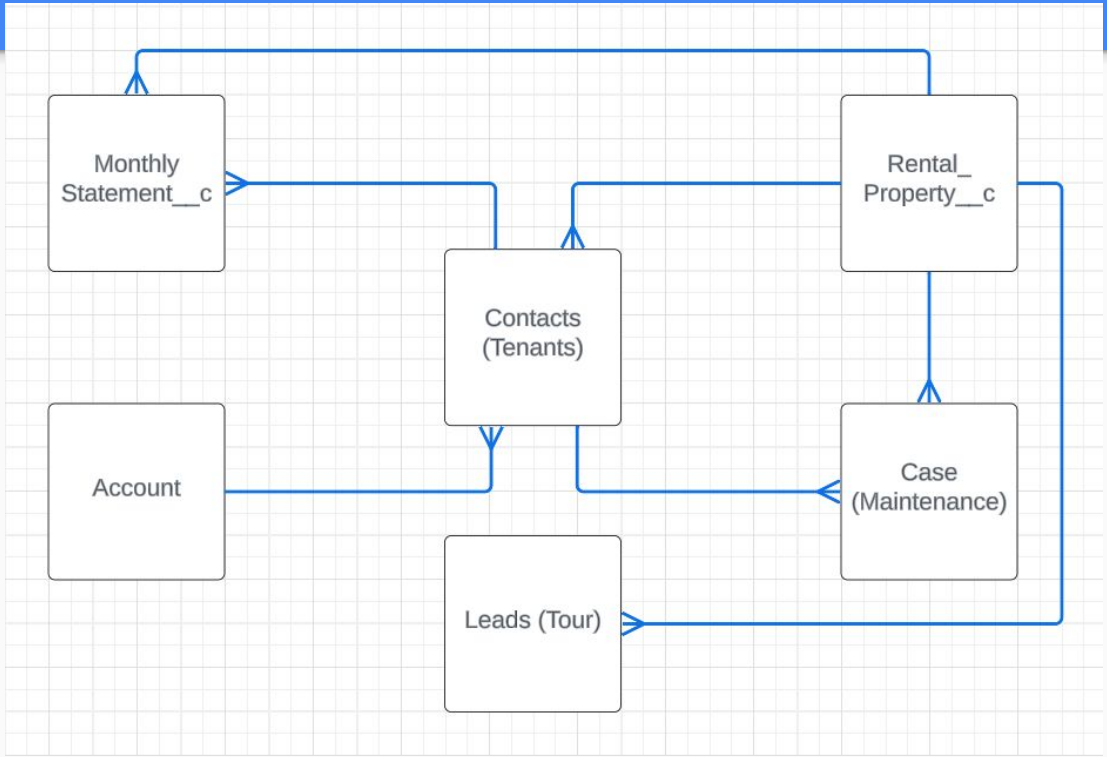
Website Link

https://revaturerealestateproject-dev-ed.trailblaze.my.site.com/bytepropertymanagement/

Login Info

Username: bytebandits@mail.com
Password: bytebandit2024

ERD


Monthly Statement__c
Rental_ Property__c
Contacts (Tenants)
Account
Case (Maintenance)
Leads (Tour)

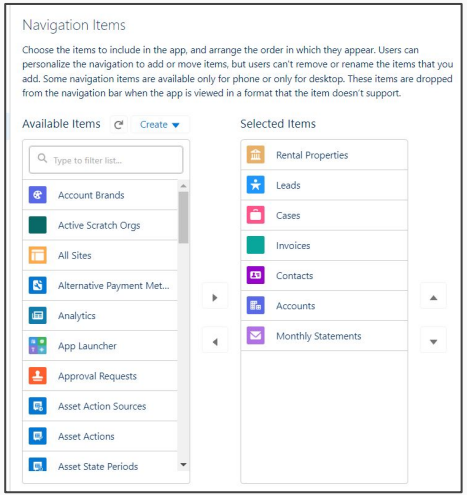
Anthony Schultz

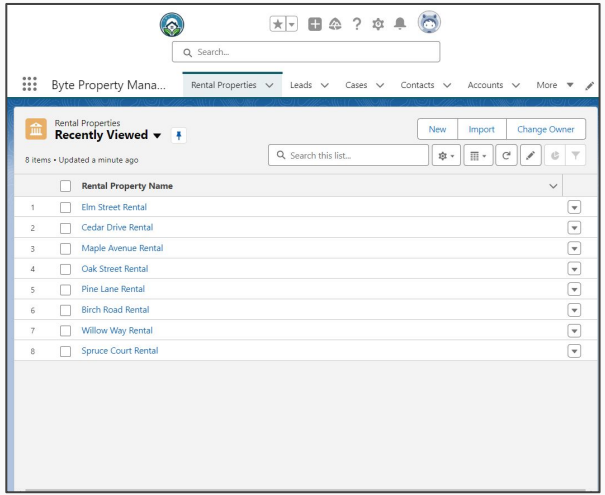
Go Through The LWC Handled As User

- Go to Home Page and Show off the Hero LWC.
- Talk about the Nav LWC and show it off, mention it conditionally renders navbar options based on if user is Guest or Logged in. Nav bar items change as hovered over.
- Go to Sign Up Page and Show They Can Sign Up. Logout.
- Go to Login Page and Show new User Can Login. Logout
- Go to Login Page and login as ByteBandits@mail.com
- Go to Account Portal Page and show Maintenance Request Being Submitted.

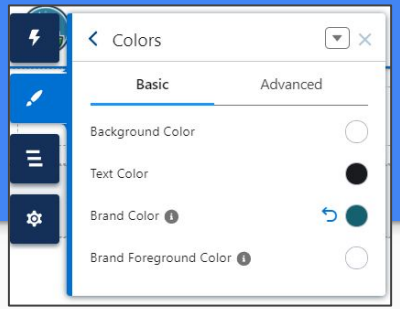
Setting Up the App in Salesforce

- **App Creation:** Navigated to Setup > App Manager > New Lightning App. Configured the app settings, including the app name, description, and branding.
- **Navigation Tabs:** Added relevant tabs such as Rental Properties, Leads, Cases, Contacts, Accounts, and Monthly Statements to the app navigation bar.

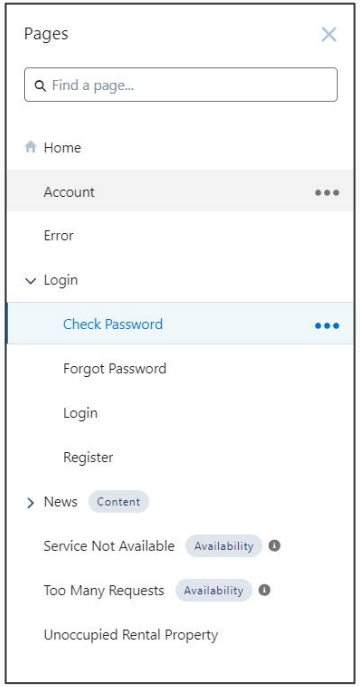


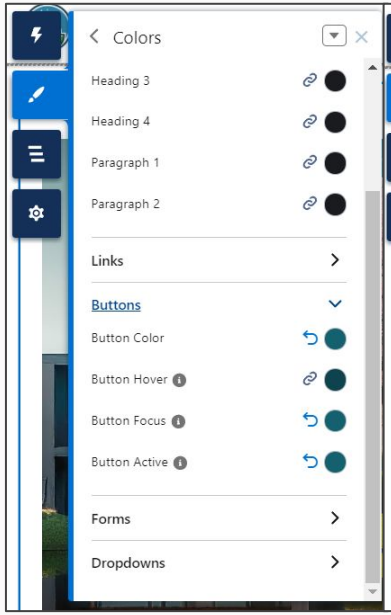


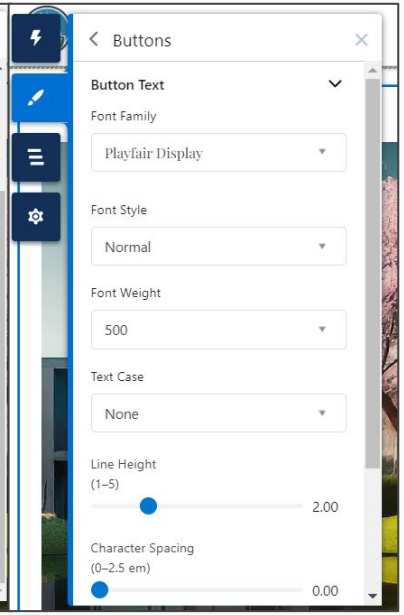
Setting Up the Experience Site



- **Site Creation:** Navigated to Setup > Digital Experiences > All Sites and created a new site, utilizing prebuilt pages such as Home, Registration, and Login, and created new pages for Account and Unoccupied Rental Property.
- **Color Theme:** Navigated to Builder > Themes > Colors to set up the core color theme used throughout the site and for buttons.
- **Button Styling:** Navigated to Builder > Themes > Buttons to modify the font family and font weight of buttons.







Overriding Sign Up & Login Button Styles

- **Custom CSS File:** Created a CSS file named `customStyles.css`.
- **Button Theming:** Targeted the login and sign-up buttons to add CSS, ensuring they match the theme of the rest of the buttons implemented by the Experience Builder theme tool.
- **Static Resource:** Imported the CSS file as a static resource.
- **Experience Builder:** Navigated to Builder > Settings > Advanced > Edit Markup and added the custom CSS file at the end of the markup to ensure it overrides the default CSS of the login and registration pages.

package.xml | customStyles.css 1 | project-scratch-def.json

force-app > main > default > staticresources > customStyles.css > ...

```
1
2  /* Styling for login button and self-register button */
3  .comm-login-form__login-button,
4  .comm-self-register__submit-button {
5      font-family: 'Playfair Display', serif !important;
6      font-style: normal !important;
7      font-weight: 500 !important;
8      text-transform: none !important;
9  }
10
11 /* Adjust carousel image max-width */
12 .slds-carousel__image {
13     max-width: 600px;
14 }
15
```

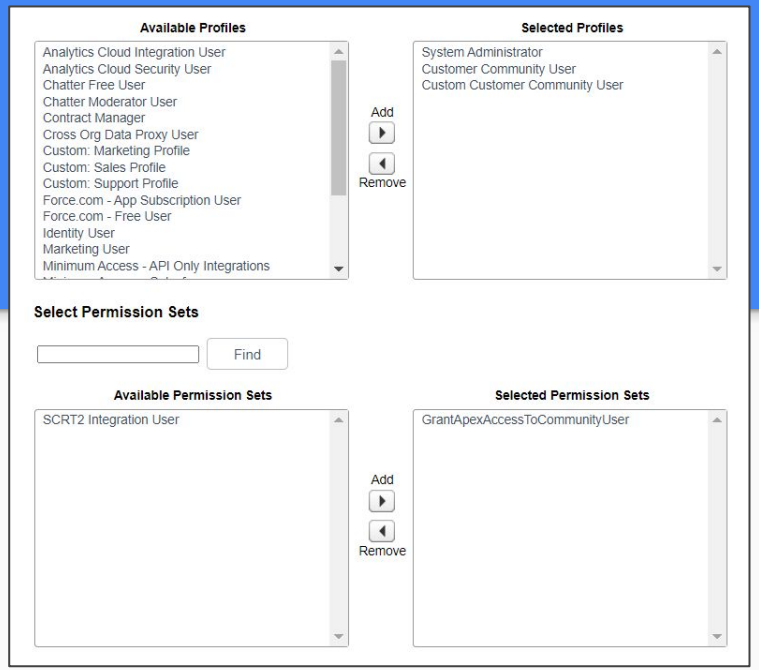
Head Markup

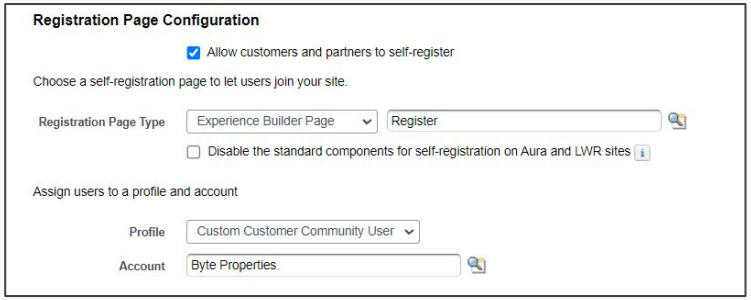
For security purposes, we allow only specific tags, attributes, and values in the <head> section. Learn M

```
1  <meta charset="UTF-8" />
2  <meta name="viewport" content="width=device-width, initial-scale=1" />
3  <title>Welcome to LWC Communities!</title>
4
5  <!-- branding stylesheets -->
6  <link rel="stylesheet" href="{ basePath }/assets/styles/salesforce-lightnin
7  <link rel="stylesheet" href="{ basePath }/assets/styles/dxp-site-spacing-st
8  <link rel="stylesheet" href="{ basePath }/assets/styles/dxp-styling-hooks.m
9  <link rel="stylesheet" href="{ basePath }/assets/styles/dxp-slds-extensions
10
11
12 <!-- branding stylesheets-->
13 <link rel="stylesheet" href="{ styles/styles.css }" />
14 <link rel="stylesheet" href="{ styles/print.css }" media="print"/>
15
16
17
18 <!-- Custom CSS for Sign up / Login Buttons and Courosal Image for Tour Det
19 <link rel="stylesheet" href="/sfsites/c/resource/customStyles" />
```

Enabling Usage of Registration / Login / Check Password / Forgot Password Pre Built Pages

- **Custom Profile:** Added a Custom Customer Community User Profile so new users can be assigned to that profile.
- **Permission Set:** Added a permission set to grant new users access to Apex Classes called in LWC.
- **Pre-Built Registration Page:** Used the pre-built registration page included in Salesforce, assigning new users to the custom profile mentioned earlier and to the singular account, Byte Properties.





Developed hero LWC for Logged-In Users & Guests

- **One-Way Data Binding:** Imported the hero image URL from Salesforce static resources and bound it to a property (`heroUrl`) to dynamically set the image source in the HTML template.
- **Static Resources:** Ensured the image URL is dynamically set by importing the image URL from Salesforce static resources and binding it to a property (`heroUrl`).
- **Template Usage:** Used the `heroUrl` property in the HTML template by setting it in the `src` attribute of the `img` tag.

package.xml | customStyles.css 1 | hero.html | project-scratch-def.json

force-app > main > default > lwc > hero > hero.html > template

```html
<template>
    <div class="hero-container">
        <div class="hero-image-container">
            <img src={heroUrl} alt="Luxurious Apartment Building" class="hero-image" />
        </div>
        <div class="hero-text-container">
            <h1 class="hero-title">Welcome to Your Next Home</h1>
            <p class="hero-subtitle">Find the best rental properties in the city</p>
        </div>
    </div>
</template>
```

package.xml | customStyles.css 1 | hero.js | project-scratch-def.jsc

force-app > main > default > lwc > hero > hero.js > Hero

```js
import { LightningElement } from 'lwc';
import heroImage from '@salesforce/resourceUrl/heroImage';

export default class Hero extends LightningElement {
    heroUrl = heroImage;
}
```

Developed Navigation LWC for Guest Users and Logged-In Users

- **One-Way Data Binding:** Imported the logo URL from Salesforce static resources and bound it to a property (`logoUrl`) to dynamically set the image source in the HTML template.
- **Conditional Rendering:** Used conditional rendering (`<template lwc:if={isGuest}>` and `<template lwc:if={isLoggedIn}>`) to display different navigation links based on the user's login status.
- **Navigation Handling:** Implemented JavaScript methods to handle navigation events, using `window.location.href` to redirect users to different pages based on their actions.

hero.js | MaintenanceRequestController.cls M | navigation.js

force-app > main > default > lwc > navigation > navigation.js > Navigation > navigat

```
5
6   export default class Navigation extends NavigationMixin(Lightning
7     @track isGuest = isGuest;
8     @track isLoggedIn = !isGuest;
9
10    logoUrl = logo;
11
12    navigateToHome(event) {
13      event.preventDefault();
14      window.location.href = "/bytepropertymanagement";
15    }
16
17    navigateToSignUp(event) {
18      event.preventDefault();
19      window.location.href = "/bytepropertymanagement/SelfRegister"
20    }
21
22    navigateToLogin(event) {
23      event.preventDefault();
24      window.location.href = "/bytepropertymanagement/login";
25    }
26
27    navigateToAccount(event) {
28      event.preventDefault();
```

Developed Maintenance Request

LWC - Apex Controller Approach

```
get showForm() {
    return this.isFormVisible && !this.isGuest;
}

get showConfirmation() {
    return this.isConfirmationVisible && !this.isGuest;
}
```

- **Controller Logic:** Created a controller to insert the case based on the details entered into the form as well as the logged-in user's ContactId and Rental Property. `@AuraEnabled` annotation allowed this controller to be used in LWC.
- **Form and Submission:** Collects maintenance request details (subject and description) from the user and submits the data to the controller.
- **Data Retrieval:** Fetches and displays rental property information associated with the logged-in user using an Apex method.

```
handleSubmit() {
    // Prevent additional clicks by checking the flag
    if (this.isSubmitDisabled) {
        return;
    }

    // Disable the submit button and prevent further clicks
    this.isSubmitDisabled = true;

    // Hide the form and show the confirmation page
    this.isFormVisible = false;
    this.isConfirmationVisible = true;

    const requestPayload = {
        subject: this.subject,
        description: this.description,
        contactId: this.contactId,
        rentalPropertyId: this.rentalPropertyId
    };
    console.log('Submitting Maintenance Request:', requestPayload);

    submitMaintenanceRequest(requestPayload)
```

61 OUTPUT DEBUG CONSOLE TERMINAL PORTS Salesforce CLI

```
> prio                                  Aa ab .* No results

@AuraEnabled
public static void submitMaintenanceRequest(String subje
    try {
        if (rentalPropertyId == null) {
            throw new AuraHandledException('Rental Prope
        }

        Case maintenanceRequest = new Case(
            Subject = subject,
            Description = description,
            ContactId = contactId,
            Rental_Property__c = rentalPropertyId
        );
        insert maintenanceRequest;
        System.debug('Maintenance request inserted succe
    } catch (Exception e) {
        System.debug('Error submitting maintenance reque
        throw new AuraHandledException('Error submitting
    }
}
```

Developed AddFamilyMember

LWC - Javascript Approach

- **Standard API Usage:** Uses `lightning/uiRecordApi` to import necessary modules and fields.
- **Form and Submission:** Collects user inputs (first name, last name, email, phone) and creates a new Contact record.
- **Record Handling:** Fetches the current user's rental property information directly using `getRecord`.

addFamilyForm.js × | maintenanceRequestForm.css 2 | maintenanceRequestForm.js

force-app > main > default > lwc > addFamilyForm > addFamilyForm.js > AddFamilyMember > email

```
1  import { LightningElement, track, wire } from 'lwc';
2  import { createRecord } from 'lightning/uiRecordApi';
3  import { ShowToastEvent } from 'lightning/platformShowToastEvent';
4  import { getRecord } from 'lightning/uiRecordApi';
5  import USER_ID from '@salesforce/user/Id';
6  import CONTACT_OBJECT from '@salesforce/schema/Contact';
7  import FIRSTNAME_FIELD from '@salesforce/schema/Contact.FirstName';
8  import LASTNAME_FIELD from '@salesforce/schema/Contact.LastName';
9  import EMAIL_FIELD from '@salesforce/schema/Contact.Email';
10 import PHONE_FIELD from '@salesforce/schema/Contact.Phone';
11 import RENTAL_PROPERTY_FIELD from '@salesforce/schema/Contact.Rental_Proper
12 import isGuest from '@salesforce/user/isGuest';
13
```

addFamilyForm.js × | maintenanceRequestForm.css 2 | maintenanceRequestForm.js

force-app > main > default > lwc > addFamilyForm > addFamilyForm.js > AddFamilyMember > handleSu

```
16 export default class AddFamilyMember extends LightningElement {
64     }
65
66     handleSubmit() {
67         if (this.isSubmitDisabled) {
68             return;
69         }
70
71         this.isSubmitDisabled = true;
72         this.isFormVisible = false;
73
74         const fields = {};
75         fields[FIRSTNAME_FIELD.fieldApiName] = this.firstName;
76         fields[LASTNAME_FIELD.fieldApiName] = this.lastName;
77         fields[EMAIL_FIELD.fieldApiName] = this.email;
78         fields[PHONE_FIELD.fieldApiName] = this.phone;
79         fields[RENTAL_PROPERTY_FIELD.fieldApiName] = this.rentalProperty;
80
81         console.log('Creating record with fields:', fields);
82
83         const recordInput = { apiName: CONTACT_OBJECT.objectApiName, fields };
84
85         createRecord(recordInput)
```

Creating Test Data / Imagery

- **Developer Console:** Used the Developer Console's execute anonymous feature to create rental property test data.
- **AI-Generated Images:** Created several AI-generated images representing various rental properties, including their bedrooms and bathrooms, with 3 images for each unit.

Enter Apex Code

```
// Define a list to hold rental properties
List<Rental_Property__c> rentalProperties = new List<Rental_Property__c>();

// Add rental properties to the list with detailed fields
rentalProperties.add(new Rental_Property__c(
    Name = 'Birch Road Rental',
    Address__c = '123 Birch Road, Springfield, IL 62704',
    Bedrooms__c = 3,
    Bathrooms__c = 2,
    Garage__c = true
));
rentalProperties.add(new Rental_Property__c(
    Name = 'Spruce Court Rental',
    Address__c = '124 Spruce Court, Springfield, IL 62704',
    Bedrooms__c = 4,
    Bathrooms__c = 3,
    Garage__c = true
));
rentalProperties.add(new Rental_Property__c(
    Name = 'Pine Lane Rental',
    Address__c = '125 Pine Lane, Springfield, IL 62704',
    Bedrooms__c = 4,
    Bathrooms__c = 2,
    Garage__c = true
));
rentalProperties.add(new Rental_Property__c(
    Name = 'Oak Street Rental',
    Address__c = '126 Oak Street, Springfield, IL 62704',
    Bedrooms__c = 3,
    Bathrooms__c = 2,
```

generate pic of apartments

Here are images featuring a variety of apartment buildings:

A modern apartment building with sleek lines and floor-to-ceiling windows.

A cozy apartment building with a brick facade and a welcoming front porch.

A bright and airy apartment building with a rooftop terrace and stunning city views.

Evgeny Todorov

Properties Listing Iteration and Viewport Sizing

```
<template>
  <div lwc-ref="container">
    <template lwc:if={properties.data}>
      <div class="slds-grid slds-gutters slds-grid_pull-padded-small slds-wrap">
      <template for:each={properties.data} for:item="property">
        <div key={property.Id} class="slds-col slds-size_1-of-1 slds-medium-size_6-of-12 slds-large-size_4-of-12
        <article class="slds-card">
```

- Second template element conditionally displays property listing if there are available properties
- Third template element uses a directive to iterate through a collection
- Second div element implements a grid for controlled organization of visual elements
- Third div element specifies how much of the screen a visual element should occupy depending on the viewport

Organization and Retrieval of Content Body

- Lightning-layout-item served for columns
- Image retrieved through child component handling specialized SOQL queries
- Profile changes, permissions, and sharing were necessary for broad image access
- Rental property fields

```
<div class="slds-card__body slds-card__body_inner">
  <lightning-layout>
    <lightning-layout-item padding="around-small">
      <c-listing-image property-id={property.Id}></c-listing-image>
    </lightning-layout-item>
    <lightning-layout-item padding="around-small">
      <div class="header-column">
        <p class="field-title" title="Address">Address</p>
        <p>{property.Address__c}</p>
      </div>
      <div class="header-column">
        <p class="field-title" title="Bedrooms">Bedrooms</p>
        <p>{property.Bedrooms__c}</p>
      </div>
      <div class="header-column">
        <p class="field-title" title="Bathrooms">Bathrooms</p>
        <p>{property.Bathrooms__c}</p>
      </div>
    </lightning-layout-item>
  </lightning-layout>
</div>
```

Apex for Record Retrieval

- @AuraEnabled
- SOQL query
- Importing method in JavaScript file and using @wire decorator to store result into properties property (properties.data in for:each template directive)

```
@AuraEnabled(cacheable=true)
public static List<Rental_Property__c> getUnoccupiedRentalProperties() {
  try {
    return [
      SELECT Name, Occupied__c, Address__c, Bathrooms__c, Bedrooms__c
      FROM Rental_Property__c
      WHERE Occupied__c = FALSE
    ];
  } catch (Exception e) {
    throw new AuraHandledException(e.getMessage());
  }
}
```

```
import getUnoccupiedRentalProperties from "@salesforce/apex/PropertyListingsController.getUnoccupiedRentalProperties";

export default class Home extends NavigationMixin(LightningElement) {
  @wire(getUnoccupiedRentalProperties)
  properties;
```

Navigation from Listing to Property Detail

- Lightning button passes property ID to event handler method on click
- Destination is the Experience Page API name directed to
- Record ID passed through c__recordId assigned from event

```
navToRentalPropertyPage(e) {
  // the name Unoccupied_Rental_Property__c needs to correspond
  // with the api name of the page made in experience builder
  this[NavigationMixin.Navigate]({
    type: "comm__namedPage",
    attributes: {
      name: "Unoccupied_Rental_Property__c"
    },
    state: {
      c__recordId: e.target.dataset.id
    }
  });
}
```

```
<lightning-button label="See More" variant="brand" data-id={property.Id} onclick={navToRentalPropertyPage}
></lightning-button>
```

```
connectedCallback() {
    this.recordId = this.currentPageReference?.state?.c__recordId;
```

Familiar Aspects

- SLDS Grid for utilizing space
- Template lwc:if directive for displaying if there is something to display
- Template directive for:each for:item for iteration through a collection
- Ease of use with developer reference building blocks like lightning-carousel
- SLDS CSS classes for styling utility, readability, and consistency

```
htning-card>
<div class="slds-grid slds-gutters slds-wrap">
    <template lwc:if={imgUrlList}>
        <div class="slds-col slds-size_1-of-1 slds-large-size_1-of-2 slds-align_absolute-center">
            <lightning-carousel disable-auto-scroll>
                <template for:each={imgUrlList} for:item="img">
                    <lightning-carousel-image key={img.Id} class="slds-align_absolute-center"
                        src={img.VersionDataUrl}></lightning-carousel-image>
                </template>
            </lightning-carousel>
        </div>
    </template>
```

Lightning Data Service

- Used to view a record without involving Apex, simplifying access to Salesforce data*
- LDS handles sharing rules and field-level security for us
- Read-only access can be entirely declarative in component's markup
- Built on highly efficient local storage that is shared across all components that use it
- When one component updates a record, the other components using it are notified, and in most cases, refresh automatically

```
import rentalPropertyReference from '@salesforce/schema/Rental_Property__c';
import NAME_FIELD from '@salesforce/schema/Rental_Property__c.Name';
import ADDRESS_FIELD from '@salesforce/schema/Rental_Property__c.Address__c';
import BEDROOMS_FIELD  from '@salesforce/schema/Rental_Property__c.Bedrooms__c';
import BATHROOMS_FIELD  from '@salesforce/schema/Rental_Property__c.Bathrooms__c';
import GARAGE_FIELD  from '@salesforce/schema/Rental_Property__c.Garage__c';
import SQUARE_FOOTAGE_FIELD  from '@salesforce/schema/Rental_Property__c.Square_Footage__c';
import PETS_ALLOWED_FIELD  from '@salesforce/schema/Rental_Property__c.Pets_Allowed__c';
import MONTHLY_RENT_FIELD  from '@salesforce/schema/Rental_Property__c.Monthly_Rent__c';
import DESCRIPTION_FIELD  from '@salesforce/schema/Rental_Property__c.Description__c';
import getImgs from '@salesforce/apex/PropertyDetailController.getRecordAttachments';
```

```
objectApiName = rentalPropertyReference;
name = NAME_FIELD;
address = ADDRESS_FIELD;
bedrooms = BEDROOMS_FIELD;
bathrooms = BATHROOMS_FIELD;
garage = GARAGE_FIELD;
squareFootage = SQUARE_FOOTAGE_FIELD;
petsAllowed = PETS_ALLOWED_FIELD;
monthlyRent = MONTHLY_RENT_FIELD;
description = DESCRIPTION_FIELD;
```

Lightning Record View Form

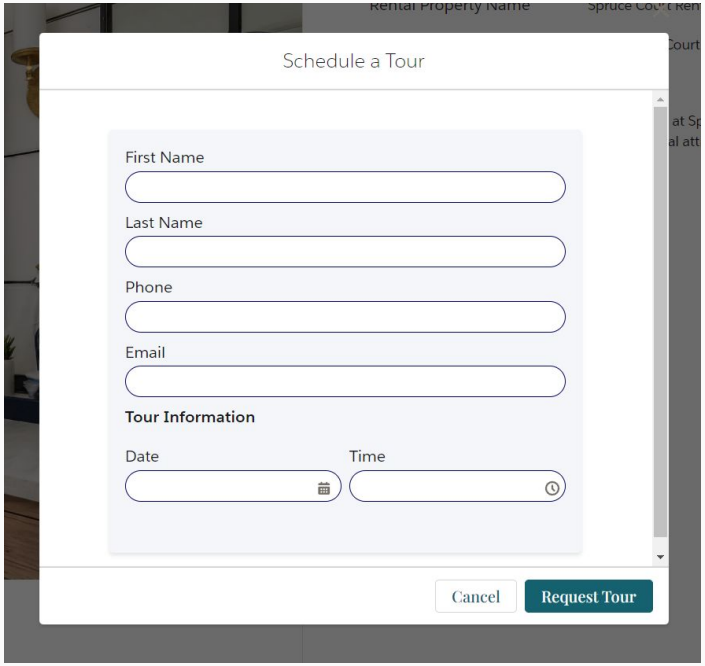
- Object-api-name and record-id necessary for form
- Lightning-output-field renders fields with their labels and current values as read-only
- Upshot is simplicity of mark-up

```
<lightning-record-view-form object-api-name={objectApiName} record-id={recordId}>
    <lightning-output-field field-name={name}> </lightning-output-field>
    <lightning-output-field field-name={address}> </lightning-output-field>
    <lightning-output-field field-name={monthlyRent}> </lightning-output-field>
    <lightning-output-field field-name={description}> </lightning-output-field>
    <lightning-output-field field-name={squareFootage}> </lightning-output-field>
    <lightning-output-field field-name={bedrooms}> </lightning-output-field>
    <lightning-output-field field-name={bathrooms}> </lightning-output-field>
    <lightning-output-field field-name={garage}> </lightning-output-field>
    <lightning-output-field field-name={petsAllowed}> </lightning-output-field>
    <lightning-button label="Schedule a Tour" variant="brand"
        onclick={handleScheduleTour}></lightning-button>
</lightning-record-view-form>
```

Richard Carranza

Schedule a Tour!

This feature takes advantage of Salesforce tools including `uiRecordApi`, `ldsUtils`, `isGuest`, and `@wire`.



Property Detail Component

```
    <lightning-button label="Schedule a Tour" variant="brand"
        onclick={handleScheduleTour}></lightning-button>

</lightning-record-view-form>
<c-modal show-modal={scheduleTour} onclose={handleModalClose} modal-he

</c-modal>
```

```
    handleScheduleTour() {
        this.scheduleTour = true;
    }

    handleModalClose() {
        this.scheduleTour = false;
    }
}
```

When the `“Schedule a Tour”` button is clicked, a boolean property `scheduleTour` is set to true, which controls the visibility of the modal.

The modal component acts as a container to effectively create a pop-up window.

Modal Component

```
<div if:true={showModal} class="slds-modal slds-fade-in-open">
    <div class="slds-modal__container">
        <header class="slds-modal__header">
            <button class="slds-button slds-button_icon slds-modal__clo
                <lightning-icon icon-name="utility:close" alternative-t
                <span class="slds-assistive-text">Close</span>
            </button>
            <h2 class="slds-text-heading_medium">{modalHeader}</h2>
        </header>
        <div class="slds-modal__content slds-p-around_medium">
            <template lwc:if={showForm}>
            <div class="slds-p-around_medium custom-form">
                <lightning-input label="First Name" onchange={handleFir
                    class="slds-m-bottom_small"></lightning-input>
                <lightning-input label="Last Name" onchange={handleLast
                    class="slds-m-bottom_small"></lightning-input>
                <lightning-input label="Phone" onchange={handlePhoneCha
                <lightning-input label="Email" onchange={handleEmailCha
                <lightning-input type="datetime-local" label="Tour Info
                    class="slds-m-bottom_small"></lightning-input>
            </div>
            </template>
            <template lwc:if={showConfirmation}>
                <lightning-card title="Tour Scheduled" icon-name="utili
```

The `if:true={showModal}` directive controls whether it is rendered in the DOM, effectively controlling visibility dynamically.

A record-edit-form is housed inside the modal markup, another way to do this would be to make this its own component.

`setTimeout` delay allows user to read scheduled tour confirmation.

```
handleSave() {
    console.log('Save button clicked');
    this.createLead()
        .then(() => {
            setTimeout(() => {
                this.closeModal();
            }, 4000);
        })
        .catch(error => {
            console.error('Error creating lead:', error);
        });
}
```

Modal Component

```
import { LightningElement, track, wire } from 'lwc';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';
import { createRecord } from 'lightning/uiRecordApi';
import { reduceErrors } from 'c/ldsUtils';
import isGuest from '@salesforce/user/isGuest';
import { CurrentPageReference } from 'lightning/navigation';
```

The modal component uses `createRecord` from `uiRecordApi` to handle data operations without custom Apex.

The `reduceErrors` function from the `ldsUtils` module is employed to simplify and display error messages returned from data operations.

The component uses `isGuest` to decide whether to show confirmation messages. This context management is used across our site to manage user access.

`@wire` is used to bring the current page's state to the component's properties, automatically setting `Property_To_Tour__c`.

```
@wire(CurrentPageReference)
setCurrentPageReference(currentPageRef
    this.recordId = currentPageReferen
    if (this.recordId) {
        this.propertyToTour = this.rec
    }
```

Monthly Statements Component

```
template lwc:if={isNotGuest}>
 <div class = "slds-card">
   <lightning-card title="My Montl
       <lightning-datatable key-f
       </lightning-datatable>
   </lightning-card>
 </div>
```

```
public without sharing class MonthlyStatementController {
  @AuraEnabled(cacheable=true)
  public static List<Monthly_Statement__c> getMonthlyStatementsForLoggedInContact() {
    Id contactId = [SELECT ContactId FROM User WHERE Id = :UserInfo.getUserId()].ContactId;
      List<Monthly_Statement__c> statements = [
        SELECT Id, Name, Rental_Property__c, Rental_Property__r
        FROM Monthly_Statement__c
        WHERE Contact__c = :contactId
        LIMIT 10
      ];
```

This component displays a list of monthly statements for the authorized user by leveraging an Apex class and a `lightning-datatable`.

`@AuraEnabled` is used to call the Apex method that queries to populate Monthly Statements list.

`isGuest` is crucial for conditional rendering between guests and authorized users.

```
const COLUMNS = [
  { label: "Statement ID", fieldName: "Name" },
  { label: "Rental Property", fieldName: "RentalPropertyName", type: "text" },
  { label: "Statement Date", fieldName: "Date__c", type: "date" },
  { label: "Amount Charged", fieldName: "Amount_Charged__c", type: "currency" }
];

export default class MonthlyStatements extends LightningElement {
  isNotGuest = !isGuest;
```

Thank you!

Questions?