# Report: Sales Automobile Using Salesforce CRM

## Project Description :

The Salesforce CRM implementation for automobile sales streamlines the entire sales process, enhancing efficiency and customer satisfaction. Through this system, sales teams can manage leads, track customer interactions, and automate follow-ups. It enables comprehensive customer profiling, allowing for personalized marketing strategies and targeted campaigns. The platform facilitates inventory management, ensuring real-time updates on available vehicles and their specifications. Integration with marketing tools enables seamless communication and lead nurturing. Additionally, the system provides insightful analytics, empowering decision-making by identifying sales trends and forecasting demand. Overall, the Salesforce CRM for automobile sales optimizes operations, fosters customer relationships, and drives revenue growth within the automotive industry.

## Short Description :

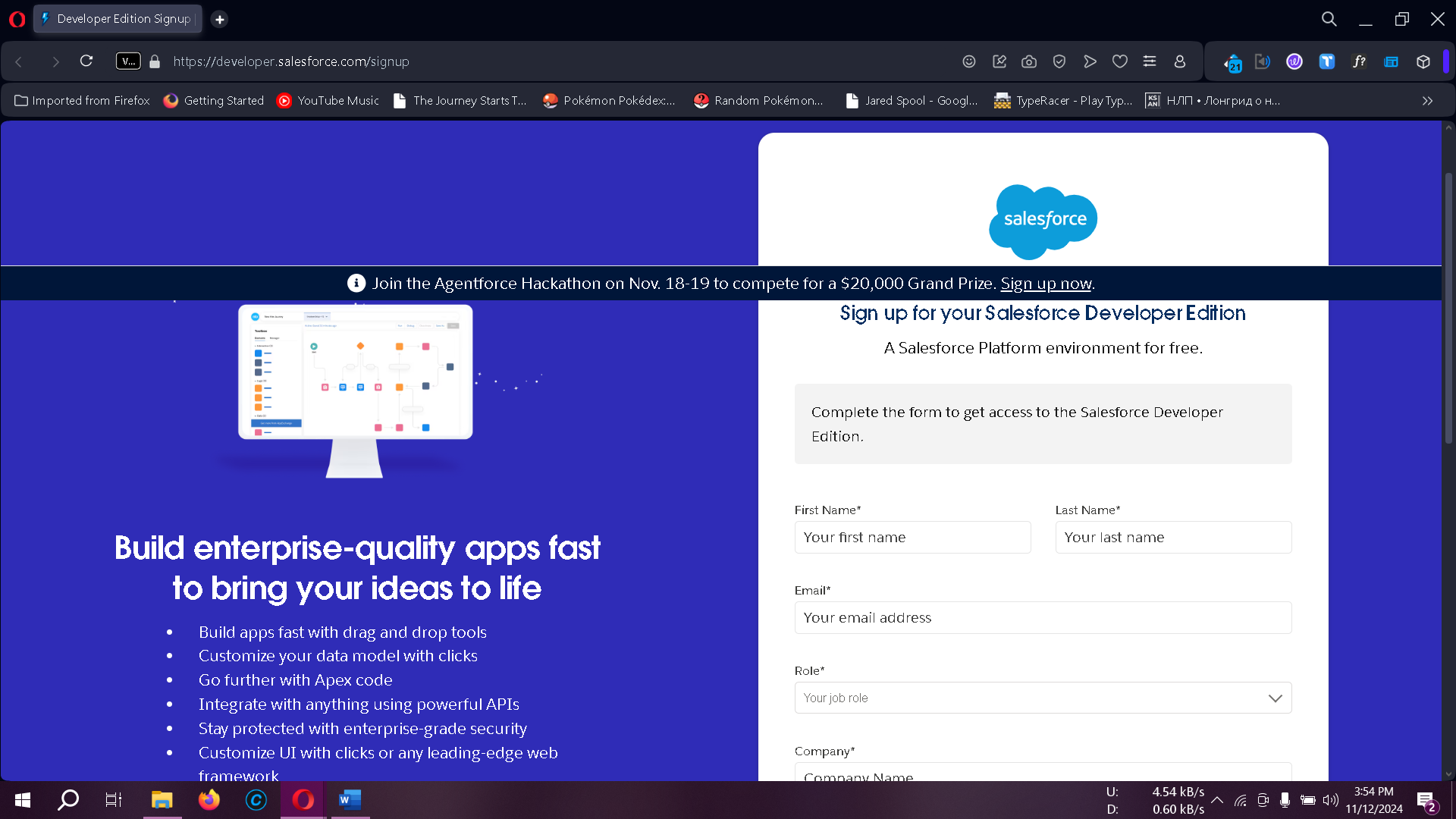
This report is to provide a comprehensive guide on creating and managing an automobile sales database using Salesforce CRM. This involves setting up a developer account, creating and managing objects, and configuring tabs, fields, and relationships to streamline data access and improve sales management in the automobile industry. This process is structured to ensure efficient and systematic handling of automobile sales data within Salesforce.

**Milestone 1-salesforce**

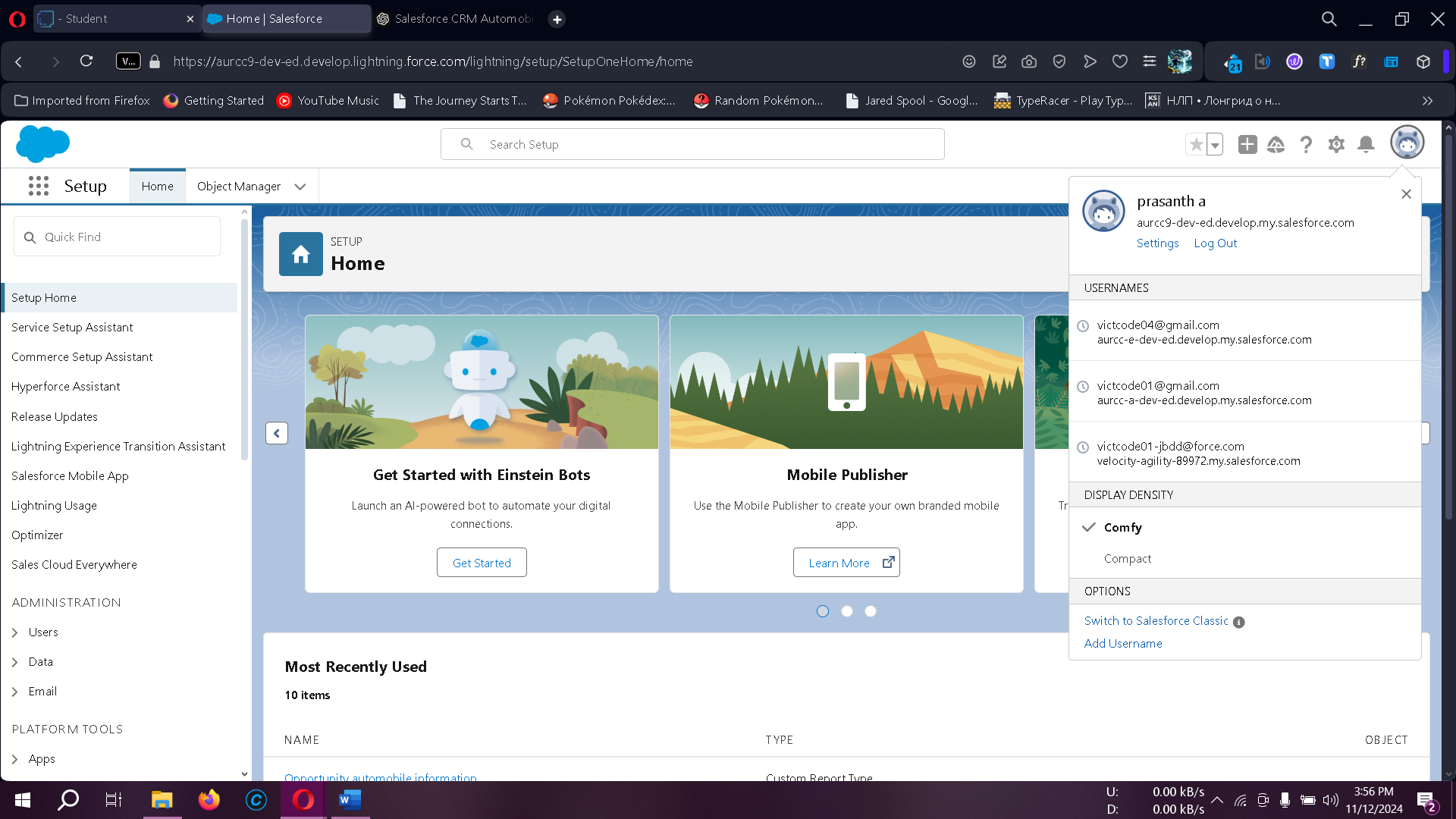
### Developer Account Creation

To start working with Salesforce CRM, a developer account is essential. Follow these steps to create an account:

1. **Sign-Up Process**
   * Go to [Salesforce Developer Sign-Up](https://developer.salesforce.com/signup).
   * Enter your **First and Last Name**, **Email**, and set **Role** as “Developer.”
   * Input your **Company** (College Name), **Country** (India), **Postal Code**, and **Username** (formatted as username@organization.com).
   * Click **Sign Me Up** after filling out the form.



1. **Account Activation**
   * Open the inbox of the email used for registration, locate the Salesforce verification email, and click **Verify Account**.
   * Set a password, choose a security question, and log into your Salesforce account to access the setup page.



### 3. Objects in Salesforce

Salesforce objects function as database tables for storing and organizing data relevant to the organization.

* **Standard Objects**: Provided by Salesforce by default (e.g., Accounts, Contacts).
* **Custom Objects**: User-defined objects to store unique organizational data.

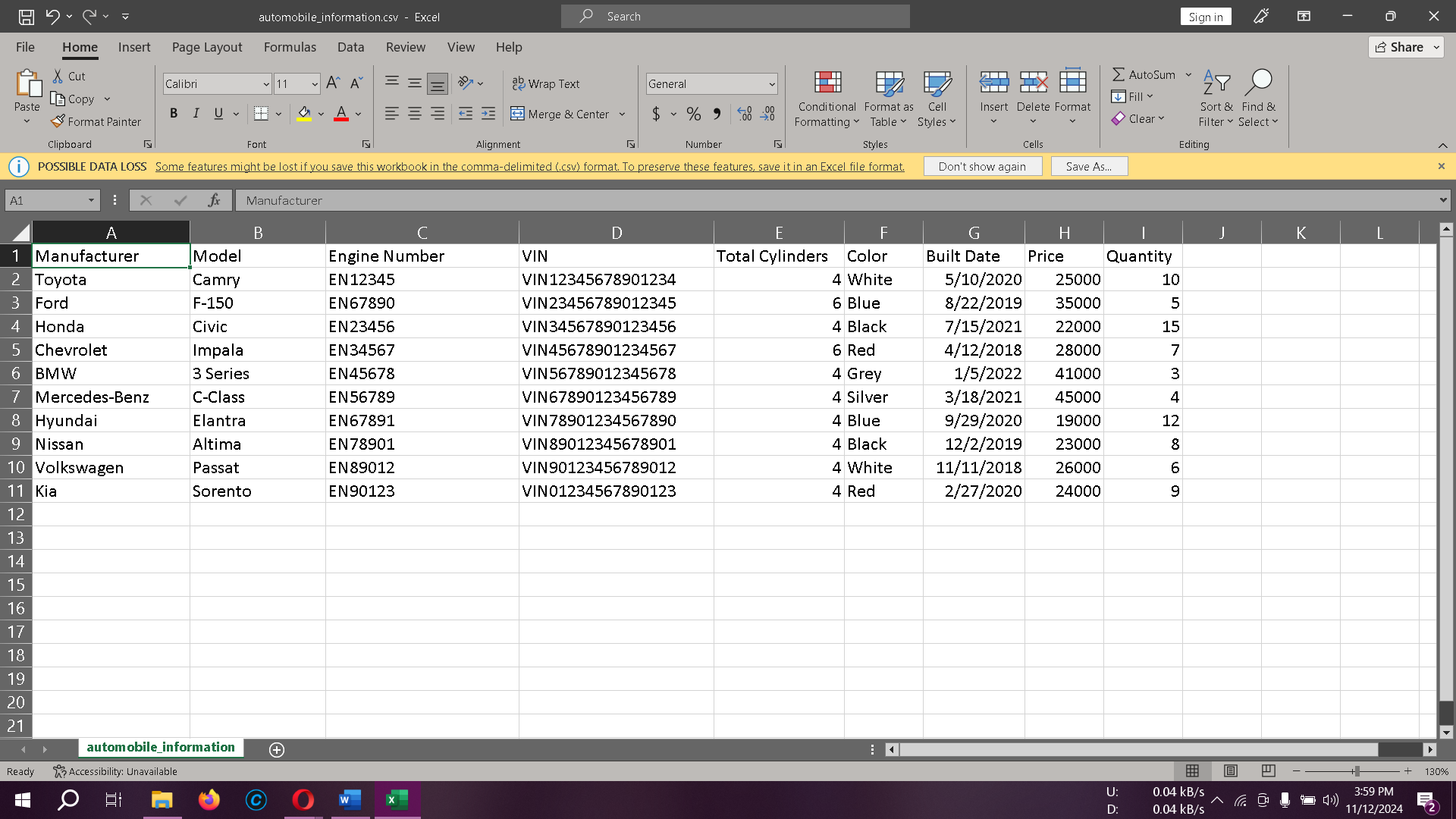
#### Use Case

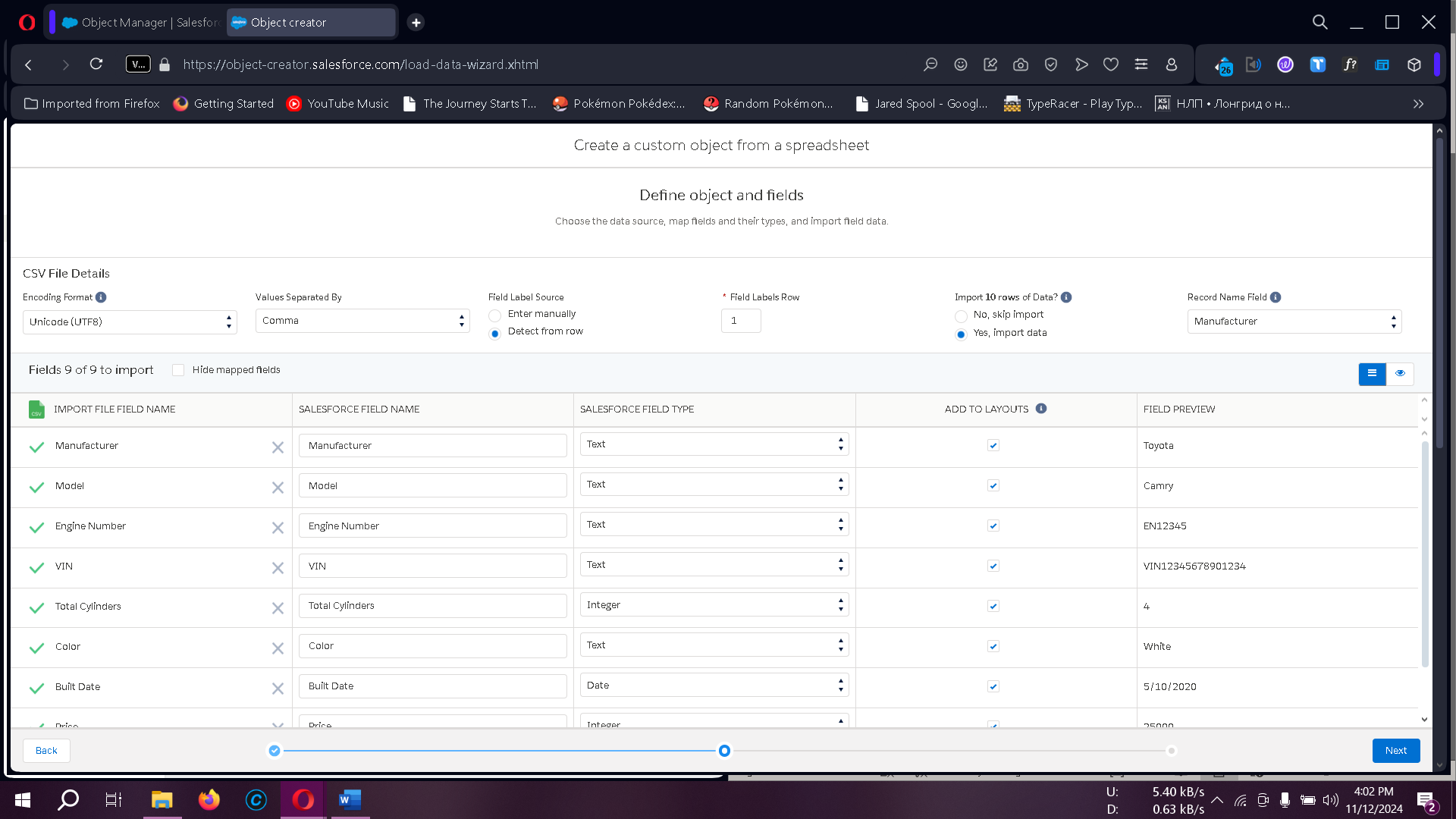
Creating custom objects such as Automobile Information and Invoice enables efficient data handling, process automation, and tailored reporting for automobile sales.

### 4. Creating Objects

#### 4.1 Automobile Information Object

1. Download and open **AutomobileInformation.csv**.
2. Log into Salesforce, navigate to **Setup > Object Manager**.
3. Select **Custom Object from Spreadsheet**, login if prompted, upload the CSV file, and map fields with the correct data types.
4. Follow prompts and finish to complete object creation.

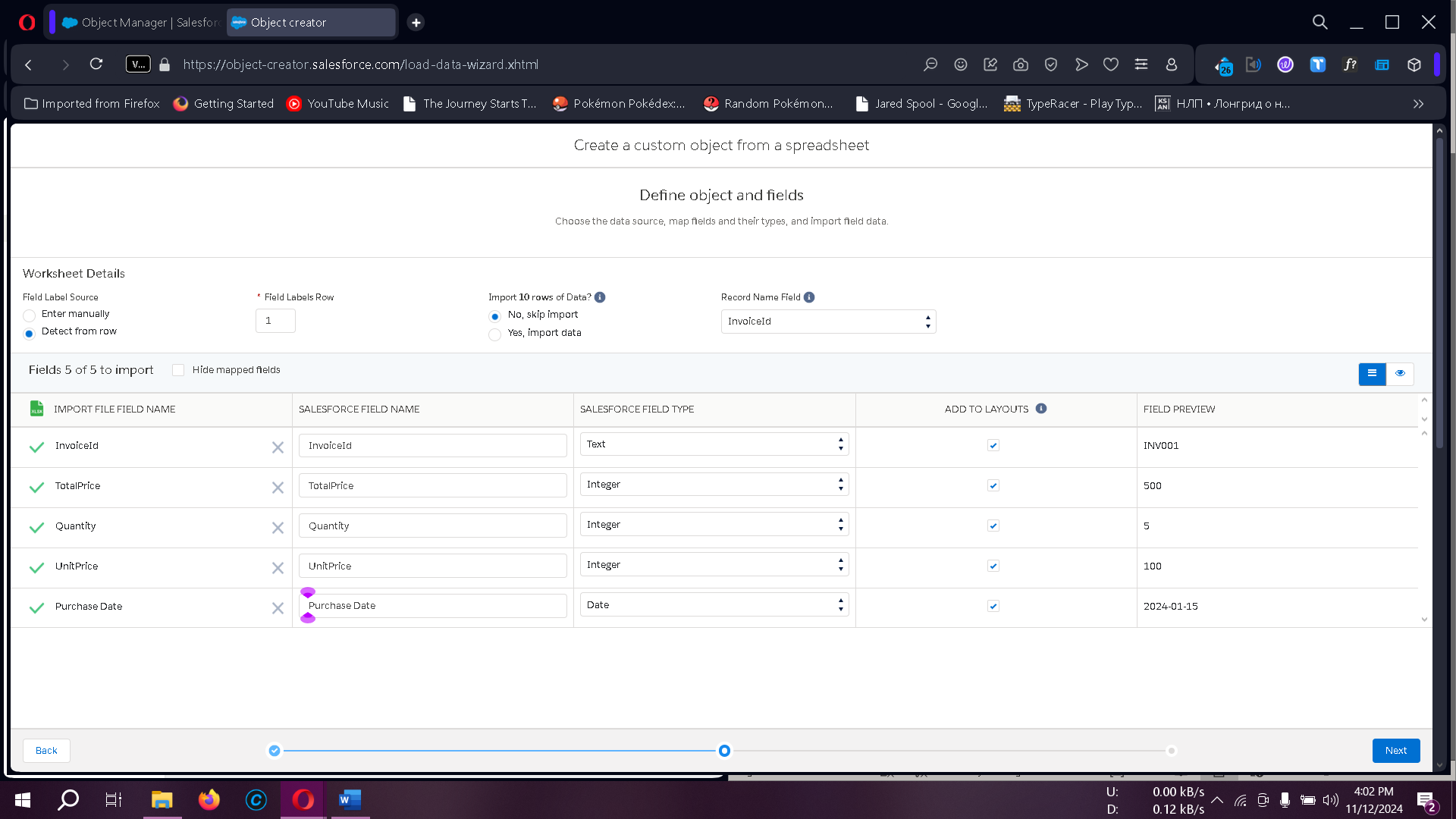




#### 4.2 Invoice Object

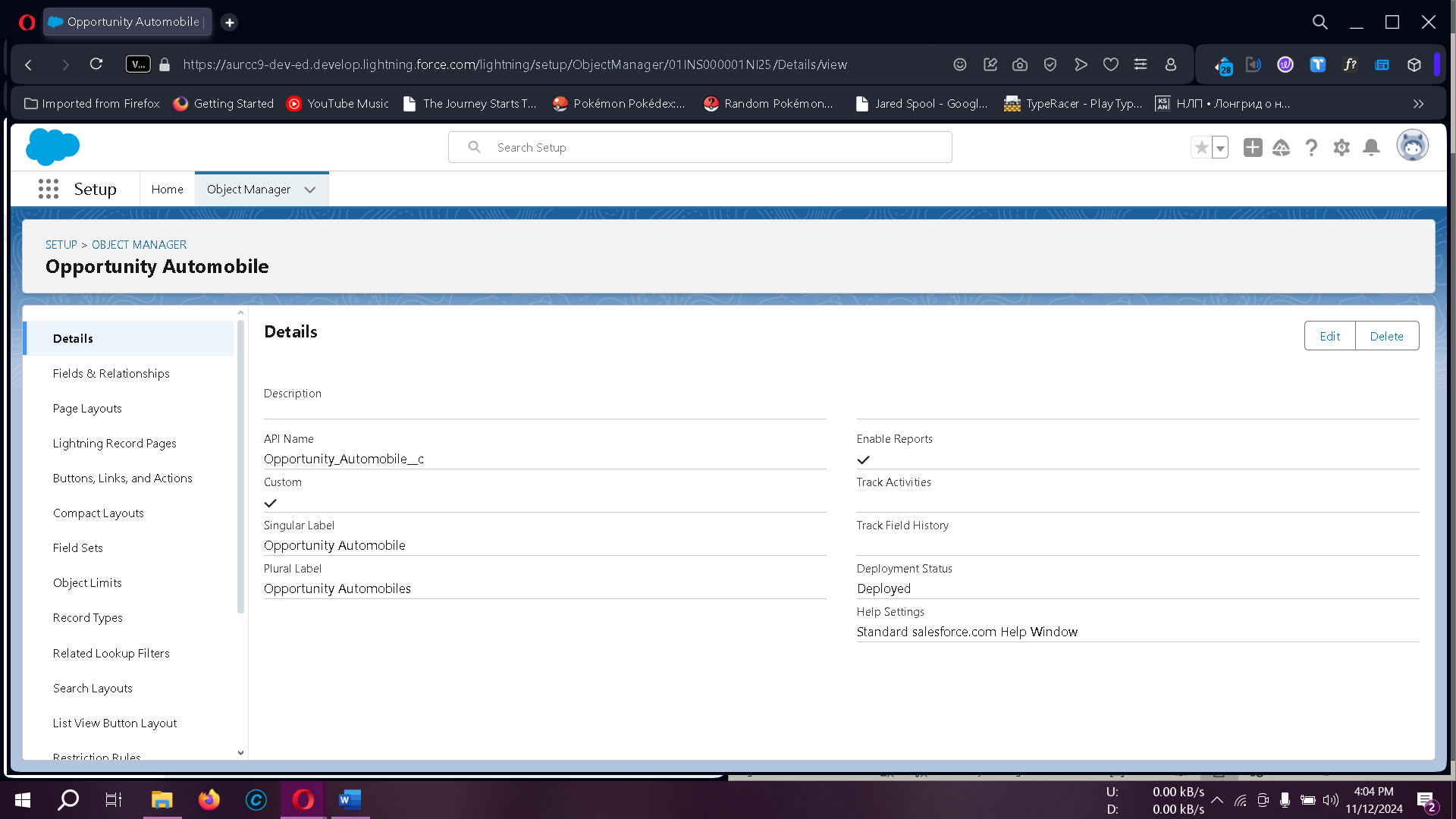
Repeat the steps above with the **Invoice CSV** file, ensuring proper field mapping.





#### 4.3 Opportunity Automobile Object

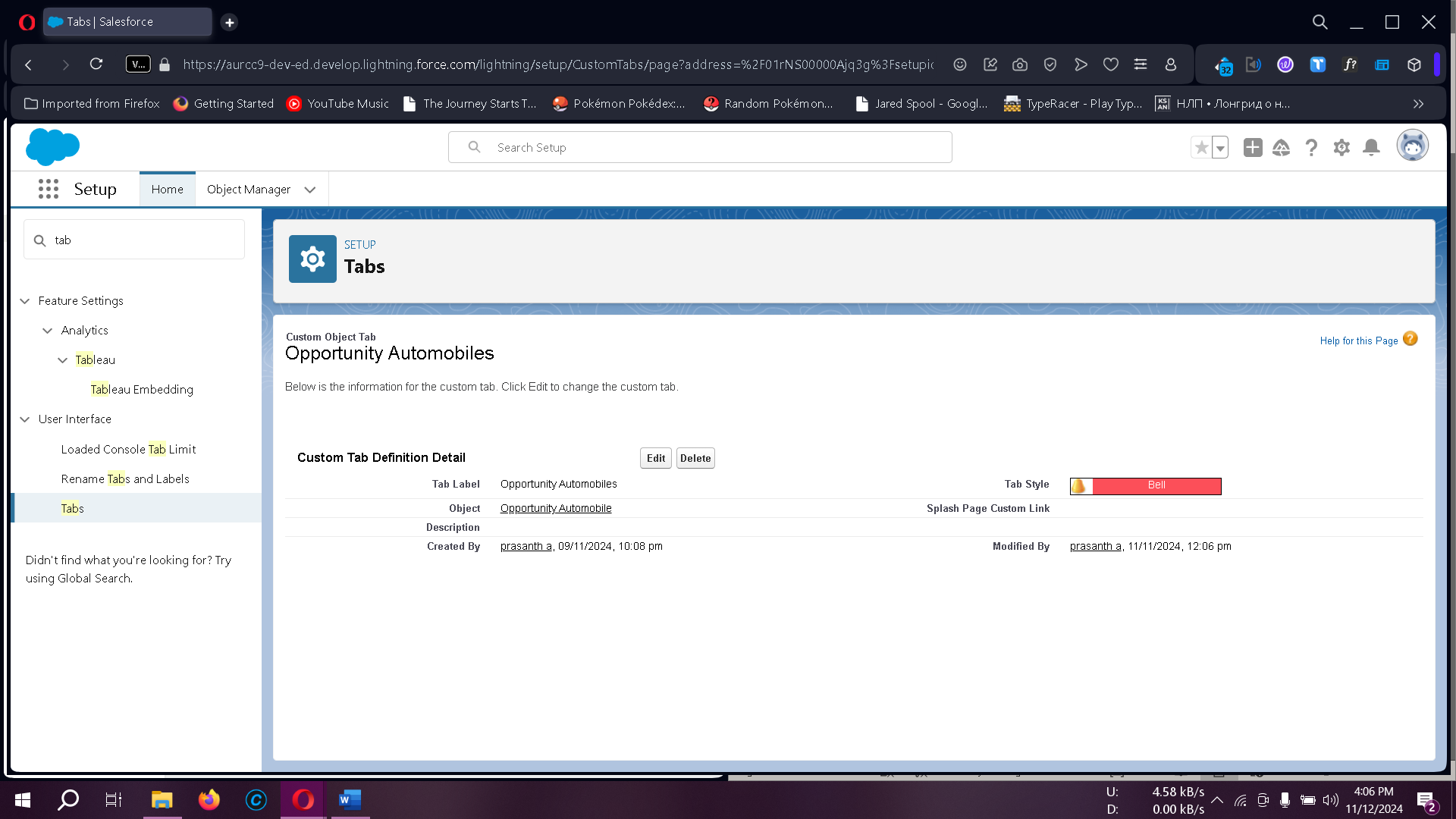
* From **Setup > Object Manager**, select **Create > Custom Object**.
* Configure labels, record names, and data types (e.g., **Auto Number** with display format OA-{0000}, starting at 1).
* Enable reporting and search functionality, then save the object.



### 5. Tabs

Tabs in Salesforce provide a user interface for managing and viewing records.

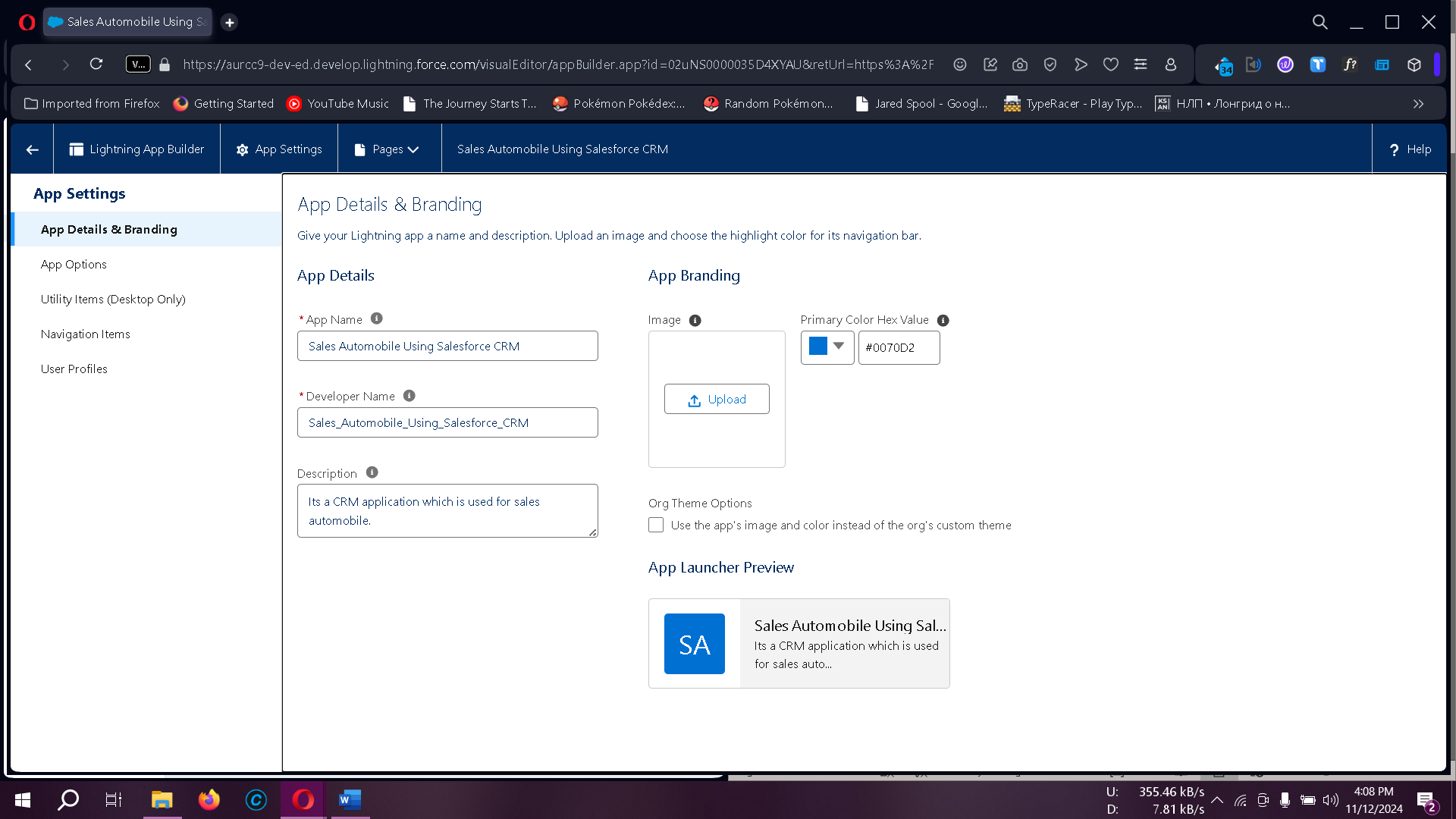
1. **Types of Tabs**:
   * **Custom Tabs**: Specific to custom objects.
   * **Web Tabs**: Display web content.
   * **Visualforce Tabs**: Display Visualforce pages.
   * **Lightning Component Tabs**: Add Lightning components to the navigation.
   * **Lightning Page Tabs**: Add Lightning Pages to mobile app navigation.
2. **Creating a Custom Tab**
   * From **Setup**, search **Tabs** and select **New (Custom Object Tab)**.
   * Choose **Opportunity Automobile** and complete the setup.



### 6. The Lightning App

Lightning Apps enable organized navigation, grouping objects, tabs, and settings into a bundle.

1. **Creating the Sales Automobile Lightning App**
   * From **Setup**, search **App Manager** and select **New Lightning App**.
   * Provide **App Name** (Sales Automobile Using Salesforce CRM) and branding details.
   * Add navigation items (e.g., Account, Contact, Opportunities, etc.).
   * Assign profiles to complete the setup.



### 7. Fields & Relationships

Fields in Salesforce store valuable data and support relational database functionalities.

1. **Types of Fields**:
   * **Standard Fields**: Predefined by Salesforce and include basic fields like Created By, Owner, Last Modified.
   * **Custom Fields**: User-defined fields tailored to specific organizational needs.
2. **Field Creation**:
   * Define fields such as Quantity, Unit Price, and Total Price in **Opportunity Automobile** using appropriate data types (e.g., Number, Formula).
   * Use formulas to calculate fields (e.g., Unit\_Price\_\_c \* Quantity\_\_c for Total Price).

#### Use Case

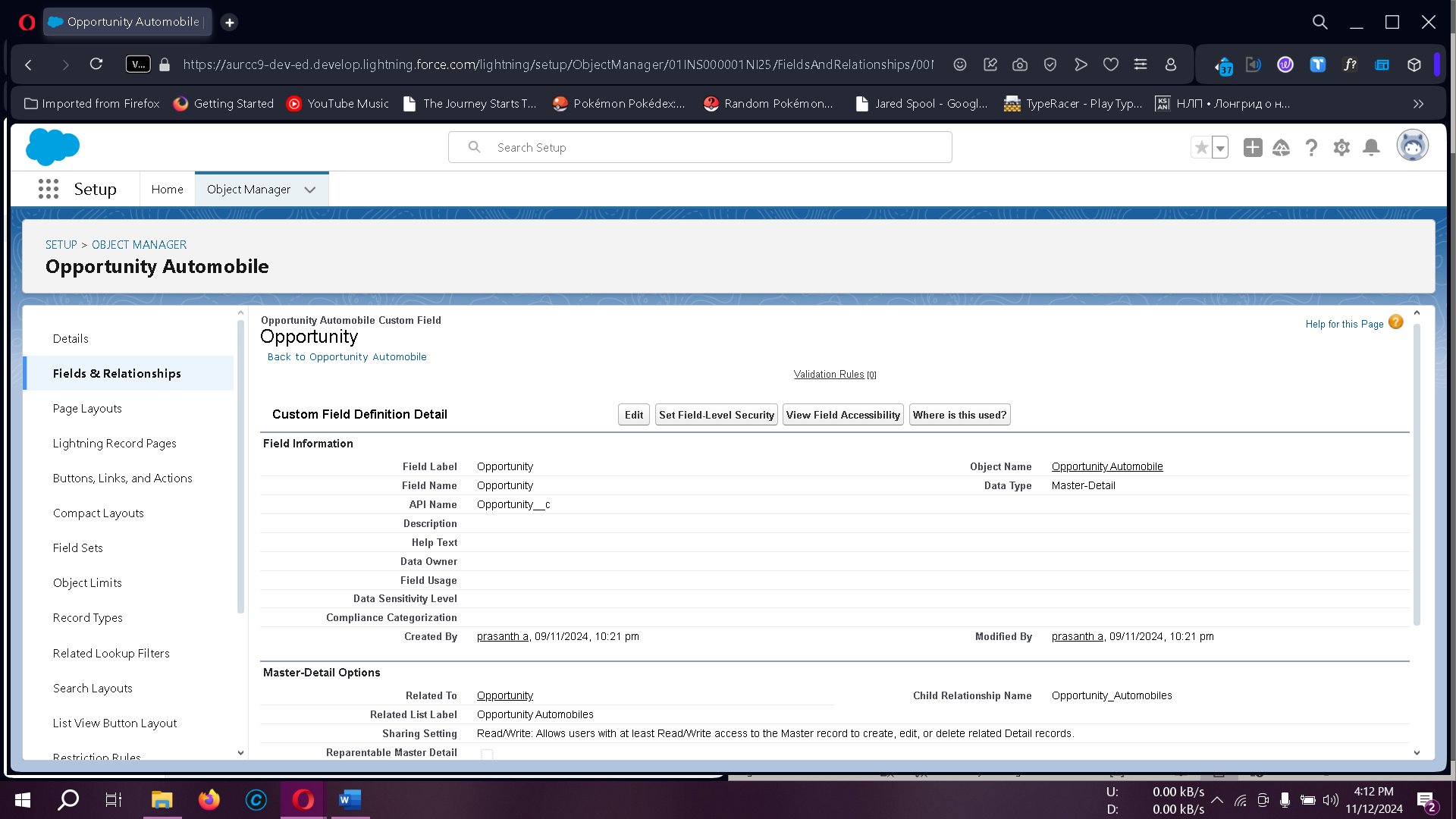
Creating fields for objects allows storing structured information, optimizing record accessibility, and enabling automated data calculations.

### 8. Field Relationships

Establishing relationships between fields enables data consistency and organization within related objects.

#### 8.1 Creating Master Detail Relationship in Opportunity Automobile

* From **Object Manager**, select **Fields & Relationships > New** and choose **Master-Detail Relationship**.
* Link to the **Opportunity** object for data hierarchy.

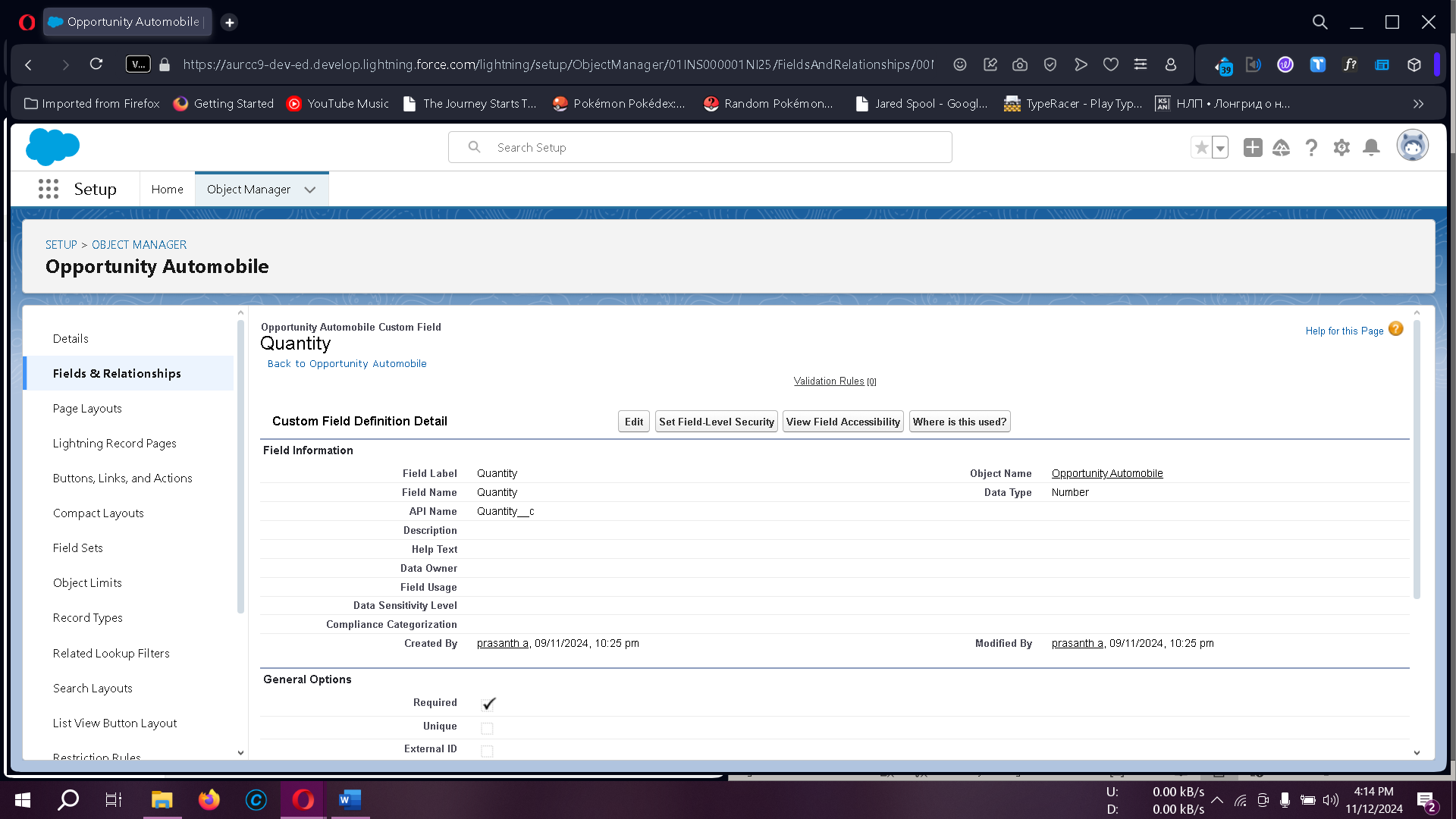


#### 8.2 Creating Lookup Field for Automobile

* Use **Lookup Relationship** to connect **Automobile Information** to **Opportunity Automobile**.

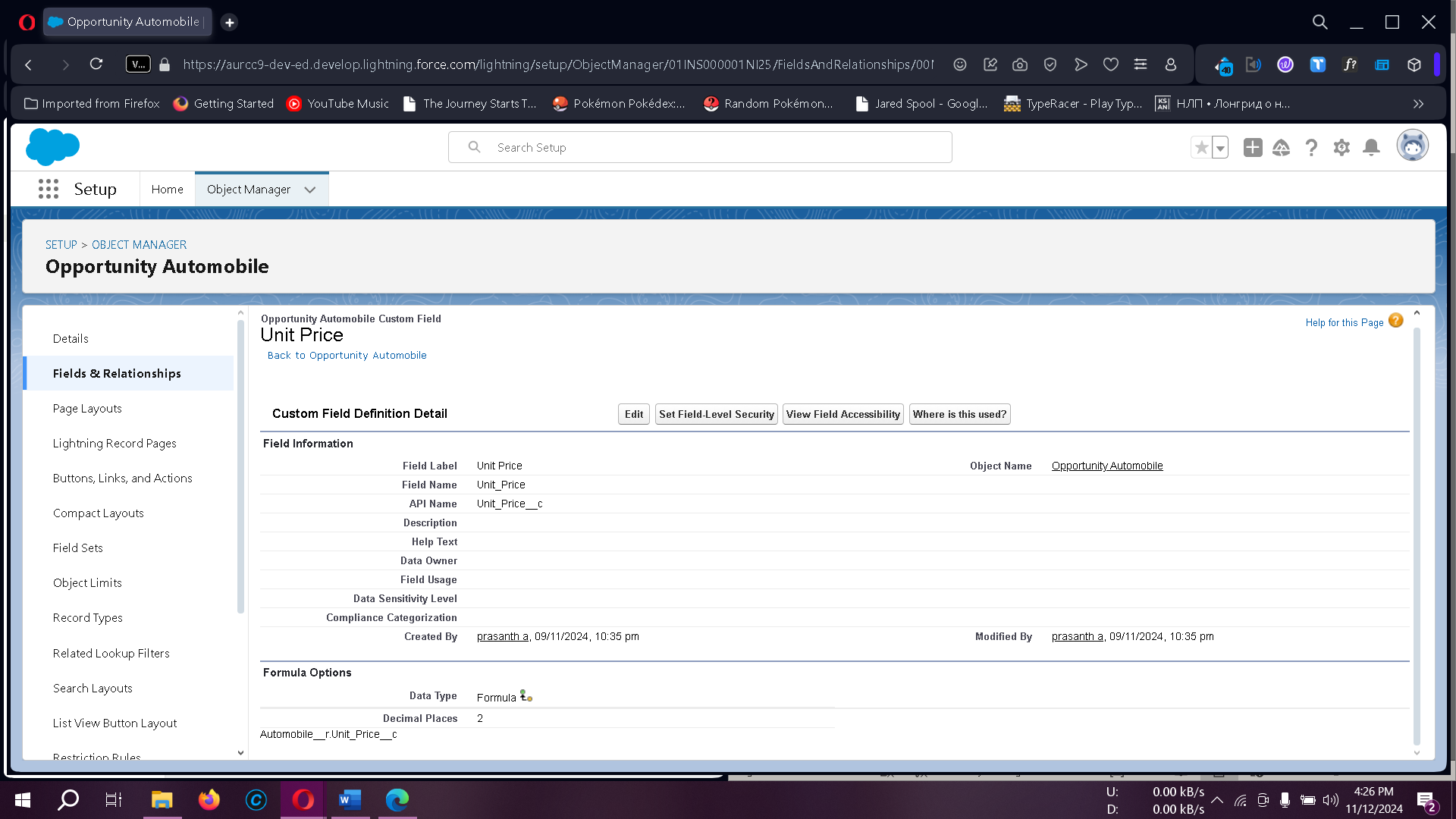
#### 8.2 Creating Quantity Number Field in Opportunity Automobile Object

* **Set the Data Type to Number and proceed with Next.**
* **Label the field as Quantity with the same Field Name, and mark it as Required.**
* **Complete the process by clicking Next > Next > Save .**



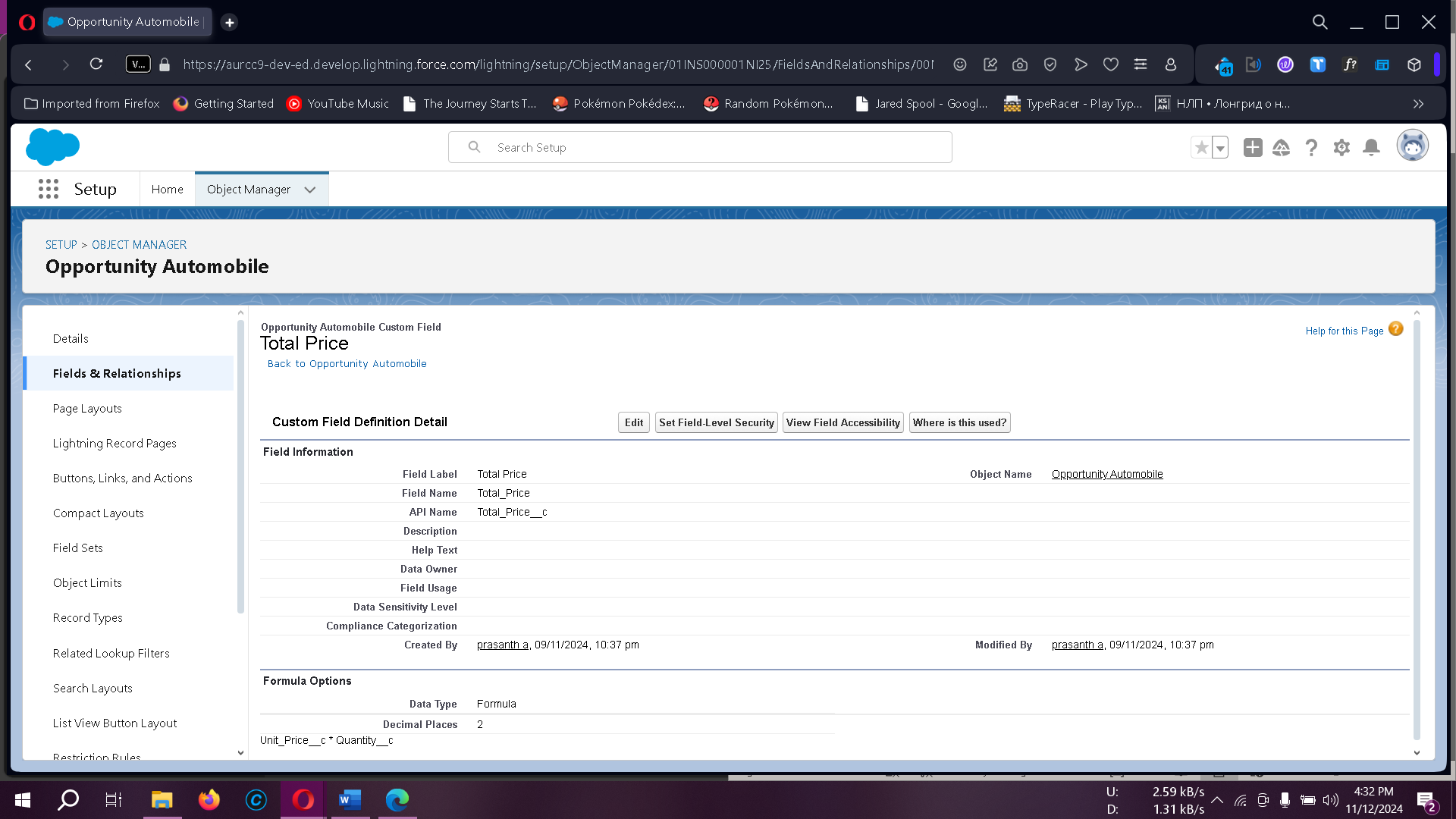
**To create a formula field in the Opportunity Automobile object:**

* **In Setup > Object Manager, locate Opportunity Automobile.**
* **Under Fields & Relationships, click New > Data Type: Formula > Next.**
* **Set Field Label and Field Name to Unit Price, select Currency as the formula return type, and set decimal places to two.**
* **Enter Automobile\_\_r.Price\_\_c in Advanced Formula > Check Syntax.**
* **Click Next > Next > Save & New.**



**To create a formula field in the Opportunity Automobile object:**

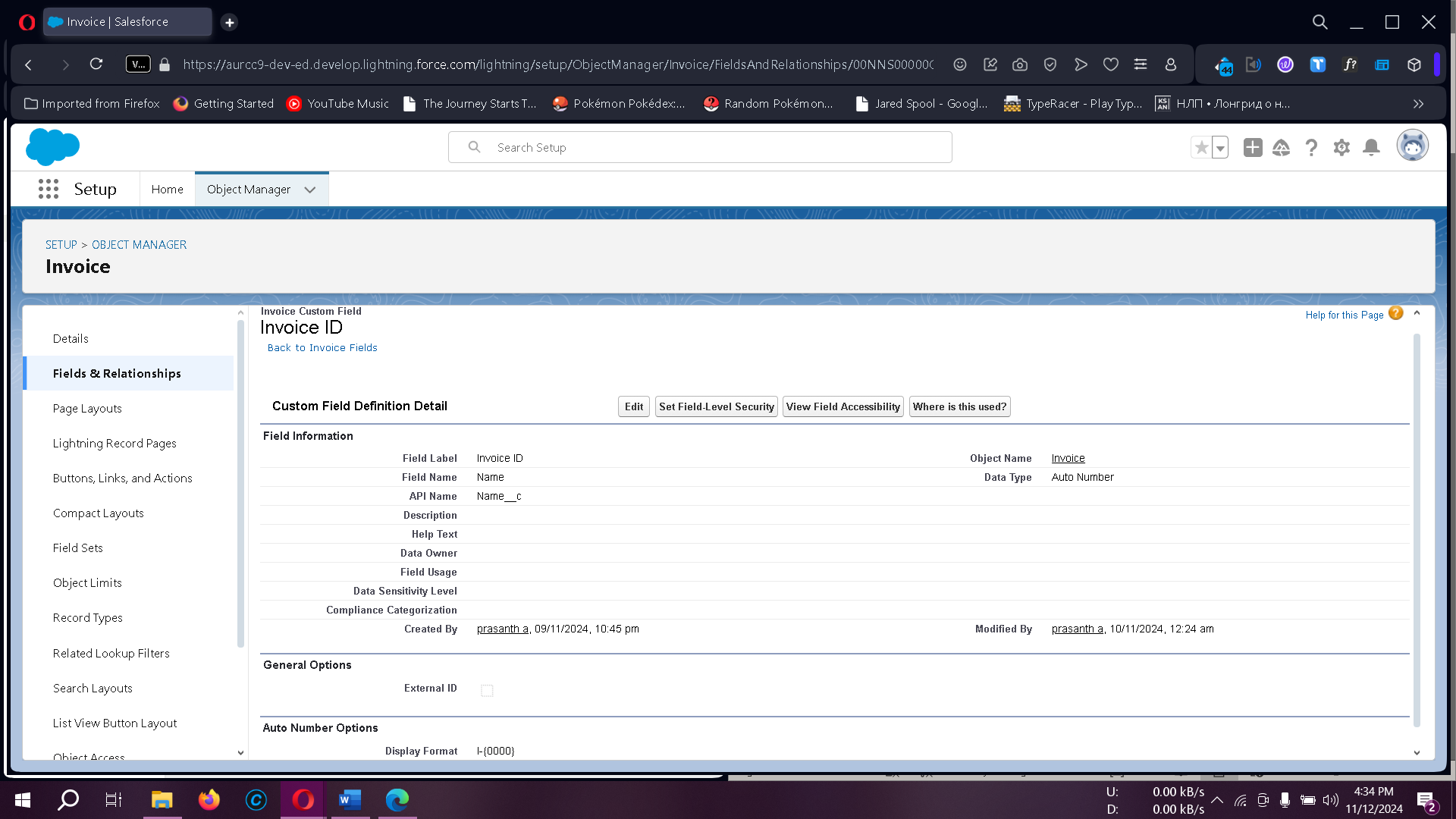
* In Setup > Object Manager, locate Opportunity Automobile.
* Go to Fields & Relationships > New > Data Type: Formula > Next.
* Set Field Label and Field Name to Total Price, select Currency as the return type, and set decimal places to two.
* In Advanced Formula, enter Unit\_Price\_\_c \* Quantity\_\_c > Check Syntax.
* Click Next > Next > Save.



### 9. Updating Fields

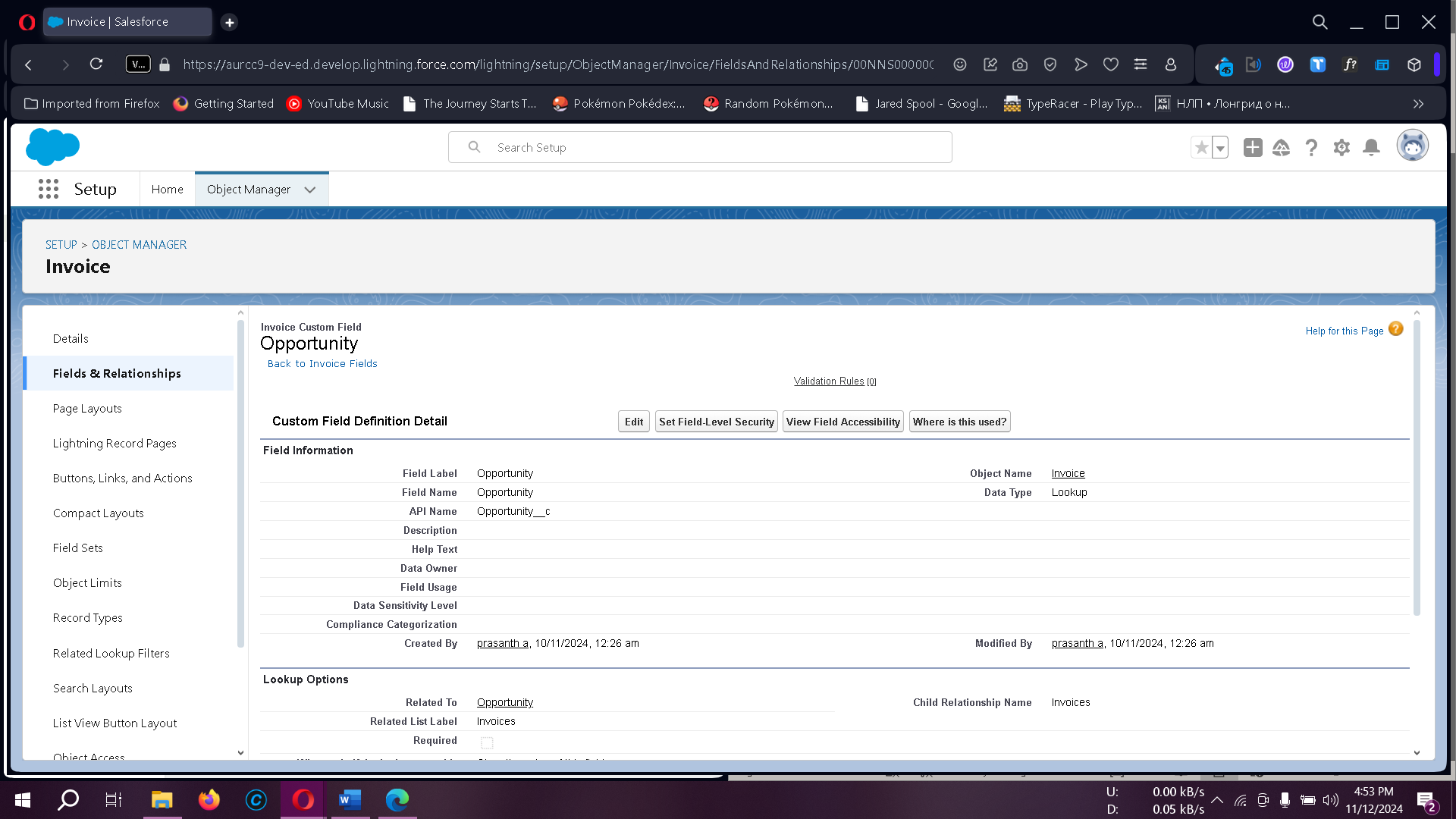
To update fields (e.g., Invoice ID in the Invoice Object):

* Navigate to **Object Manager > Invoice > Fields & Relationships**.
* Set **Data Type** to **Auto Number** with a display format (e.g., I-{0000}).



### Creating Master-Detail Relationship Field in the Invoice Object

In the **Invoice** object, create a field named **Opportunity** with a **Master-Detail Relationship** data type, establishing a connection to the **Opportunity** object.



#### Introduction to Page Layouts in Salesforce

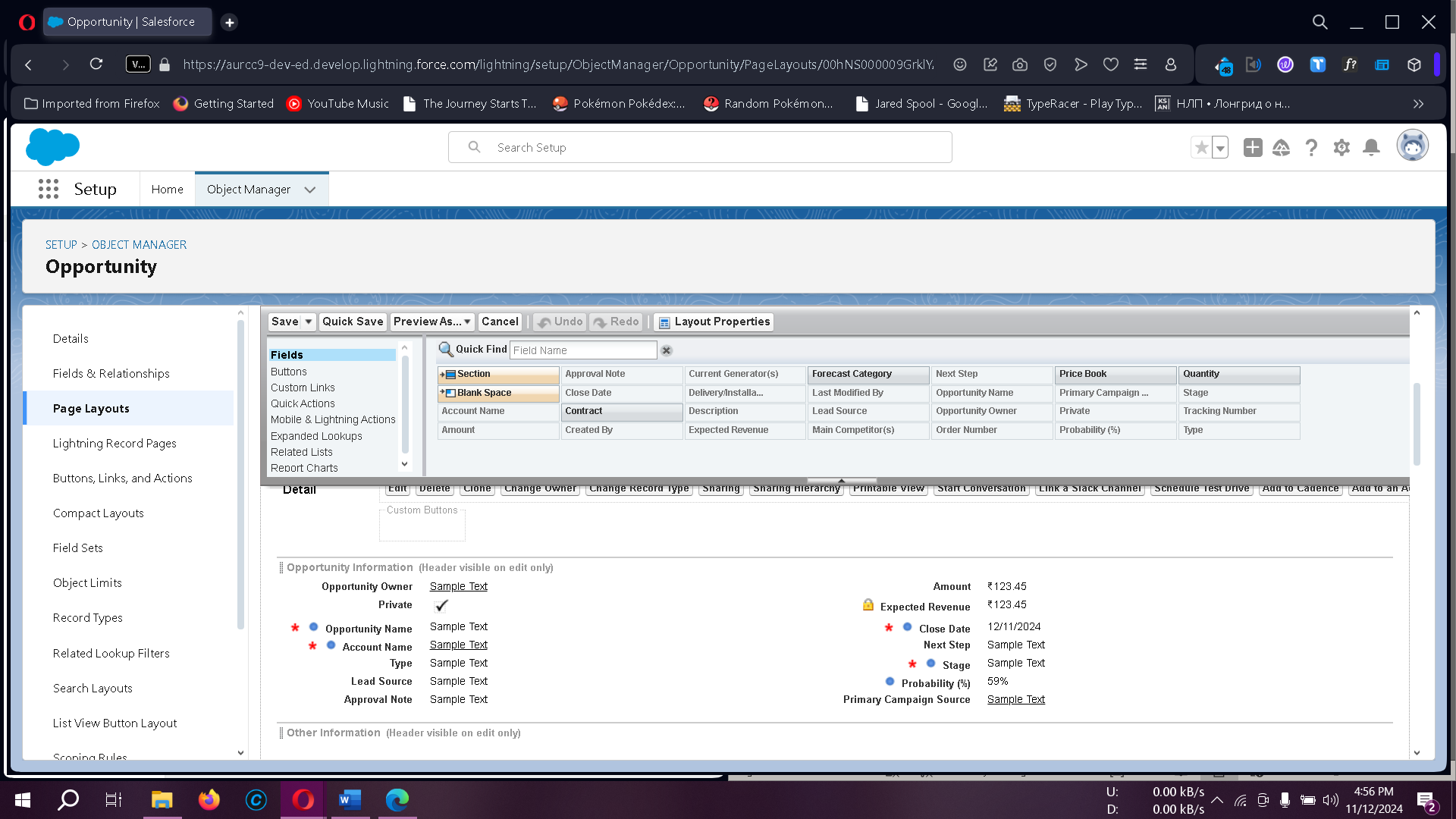
In Salesforce, **Page Layouts** allow you to design and organize detail and edit pages for both standard and custom objects, giving you control over the appearance of fields, related lists, and custom links. They help ensure an organized display of information on pages, enhancing usability and readability for users.

### Use Case: Organizing Opportunity and Automobile Information Layouts

After setting up the data model structure, the detail and edit pages for records might appear cluttered. A clean, organized layout improves data accessibility and provides a better user experience. We will edit the page layout for the **Opportunity** and **Automobile Information** objects.

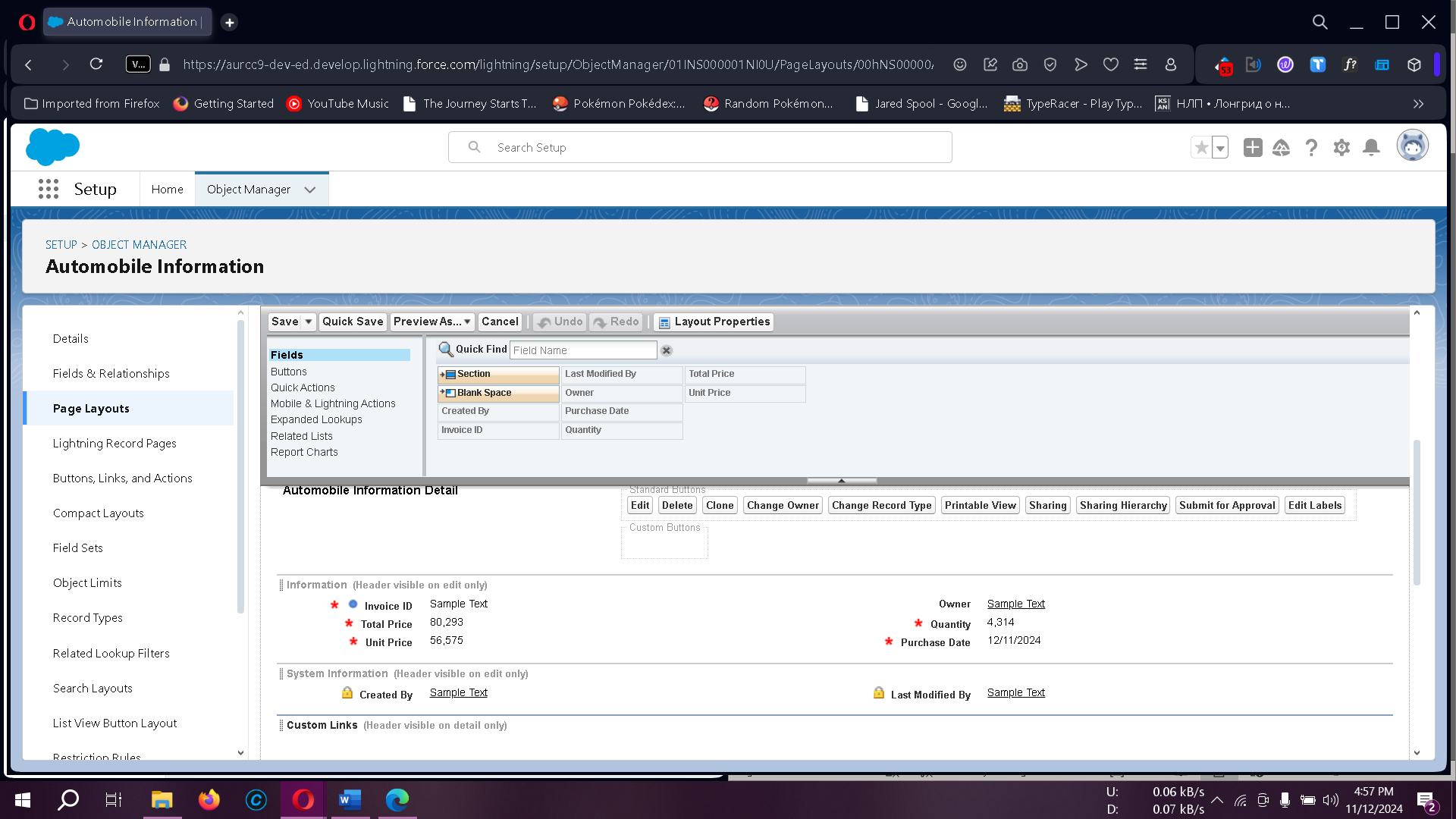
#### Editing the Page Layout for the Opportunity Object

1. **Navigate to the Setup**:
   * Go to **Setup** > **Object Manager**.
   * In the search bar, select **Opportunity Layout**.
2. **Access Page Layouts**:
   * Click **Page Layouts** in the left panel.
   * Select **Opportunity Layouts**.
3. **Customize Fields**:
   * Locate the **Opportunity Detail** section.
   * Select the **Properties icon** for the **Account Name** field.
   * Check the **Required** box and confirm by clicking **OK**.
4. **Save the Layout**:
   * Click **Save** to apply changes.



#### Editing the Page Layout for Automobile Information

1. **Navigate to the Setup**:
   * Go to **Setup** > **Object Manager**.
   * Search for **Automobile Information**.
2. **Access Page Layouts**:
   * Click **Page Layouts** and select **Automobile Information Layout**.
3. **Mark Fields as Required**:
   * For each field in the Automobile Information object, mark them as required if needed by clicking the **Gear Icon** next to each field.
4. **Adjust Field Positioning**:
   * Organize the fields in a visually appealing layout for clarity and readability.
5. **Save the Layout**:
   * Click **Save** to finalize changes.



### Implementing Apex Triggers for Automation

Apex Triggers enable custom actions before or after changes to Salesforce records, helping automate complex business processes. Below are some use cases with trigger implementations for the Opportunity and Automobile Information objects.

#### Use Case 1: Updating Automobile Quantity upon Opportunity Closure

* **Objective**: When an **Opportunity** is marked as **Closed Won**, deduct the specified **Opportunity Automobile quantity** from the available stock in the **Automobile Information** object.
* **Class and Trigger**: The OpportunityHandlerClass class and OpportunityTrigger trigger manage the logic to update quantities accordingly.

**Class Code:**

public class OpportunityHandlerClass {

    public static void opportunityAutomobileQuantity(List<Opportunity> LstOpportunity, Map<Id,Opportunity> OldMapOpportunity){

        set<Id> opportunityIds = new set<Id>();

        for(Opportunity opp : LstOpportunity){

            if(opp.StageName =='Closed Won' ){

                opportunityIds.add(opp.Id);

            }

        }

        Map<Id,Opportunity\_Automobile\_\_c> lstOpportunityAutomobile =new Map<Id,Opportunity\_Automobile\_\_c>([SELECT Id, Opportunity\_\_c, Automobile\_\_c, Quantity\_\_c, Unit\_Price\_\_c, Total\_Price\_\_c FROM Opportunity\_Automobile\_\_c Where Opportunity\_\_c IN: opportunityIds]);

        set<Id> AutoInformationIds = new set<Id>();

        for(Opportunity\_Automobile\_\_c OppAuto: lstOpportunityAutomobile.values()){

            if(OppAuto.Automobile\_\_c != null){

                AutoInformationIds.add(OppAuto.Automobile\_\_c);

            }

        }

        List<Automobile\_Information\_\_c> lstAutomobileInfomation = new List<Automobile\_Information\_\_c>();

        Map<Id,Automobile\_Information\_\_c> MapAutomobileInformation = New Map<Id,Automobile\_Information\_\_c>([SELECT Quantity\_\_c, Price\_\_c, Name, Id FROM Automobile\_Information\_\_c WHERE Id IN: AutoInformationIds]);

        For(Opportunity\_Automobile\_\_c AutoOpp : lstOpportunityAutomobile.Values()){

            decimal num = 0;

            if(AutoOpp.Automobile\_\_c == MapAutomobileInformation.get(AutoOpp.Automobile\_\_c).Id && OldMapOpportunity.get(AutoOpp.Opportunity\_\_c).stagename != 'Closed Won'){

                num = MapAutomobileInformation.get(AutoOpp.Automobile\_\_c).Quantity\_\_c- AutoOpp.Quantity\_\_c;

                MapAutomobileInformation.get(AutoOpp.Automobile\_\_c).quantity\_\_c = num;

                lstAutomobileInfomation.add(MapAutomobileInformation.get(AutoOpp.Automobile\_\_c));

            }

        }

        If(!lstAutomobileInfomation.IsEmpty()){

            update lstAutomobileInfomation;

        }

    }

}

**Trigger Code**:

trigger OpportunityTrigger on Opportunity (before update, after update) {

if (Trigger.isBefore && Trigger.isUpdate) {

OpportunityHandlerClass.opportunityAutomobileQuantity(Trigger.new, Trigger.oldMap);

}

}

#### Use Case 2: Validating Automobile Quantity Availability

* **Objective**: Ensure that the **quantity requested** in an Opportunity does not exceed the available stock in the **Automobile Information** object.
* **Class and Trigger**: The OpportunityAutomobileHandler class and OpportunityAutoMobileTrigger trigger handle validation.

**Class Code:**

public class OpportunityAutomobileHandler {

    public static void quantityErrorOnAutomobileInformation(List<Opportunity\_Automobile\_\_c> lstOpportunityAutomobile){

        set<Id> AutomobileIds = new Set<Id>();

        For(Opportunity\_Automobile\_\_c OppAutomobile : lstOpportunityAutomobile){

            if(oppAutomobile.Automobile\_\_c != null){

                AutomobileIds.add(oppAutomobile.Automobile\_\_c);

            }

        }

        Map<Id,Automobile\_Information\_\_c> lstAutomobileInformation = new map<Id,Automobile\_Information\_\_c>([SELECT Id, CreatedById, Quantity\_\_c, Price\_\_c FROM Automobile\_Information\_\_c WHERE Id IN: AutomobileIds]);

        For(Opportunity\_Automobile\_\_c OppAutomobile : lstOpportunityAutomobile){

            If(OppAutomobile.Automobile\_\_c == lstAutomobileInformation.get(OppAutomobile.Automobile\_\_c).Id && lstAutomobileInformation.get(OppAutomobile.Automobile\_\_c).Quantity\_\_c < OppAutomobile.Quantity\_\_c){

                OppAutomobile.addError('the Number of Automobile u want are not Available !! the Automobile are Available Count is ' + lstAutomobileInformation.get(OppAutomobile.Automobile\_\_c).Quantity\_\_c );

            }

        }

    }

}

**Trigger Code**:

trigger OpportunityAutoMobileTrigger on Opportunity\_Automobile\_\_c (before insert, before update) {

if (Trigger.isBefore && (Trigger.isInsert || Trigger.isUpdate)) {

OpportunityAutomobileHandler.quantityErrorOnAutomobileInformation(Trigger.new);

}

}

#### Use Case 3: Invoice Creation upon Opportunity Closure

* **Objective**: When an Opportunity is marked as **Closed Won**, automatically generate an **Invoice** using the data from **Opportunity Automobile** records.
* **Class**: The InvoiceCreation class is called from within the **OpportunityTrigger** to create an invoice without needing a separate trigger.

**Class Code:**

public class InvoiceCreation {

    public static void OpportunityClosedwonInvoiceGeneration(List<Opportunity> lstOpportunity, Map<Id,Opportunity>OldMapOpportunity){

        set<Id> oppIds = new Set<Id>();

        For(Opportunity opp : lstOpportunity){

            if(Opp.StageName == 'Closed Won' && OldMapOpportunity.get(opp.Id).StageName != opp.StageName){

                oppIds.add(opp.Id);

            }

        }

        List<Opportunity\_Automobile\_\_c> lstOpportunityAutomobile = [SELECT Unit\_Price\_\_c, Total\_Price\_\_c, Automobile\_\_c, Quantity\_\_c,Opportunity\_\_c, Id FROM Opportunity\_Automobile\_\_c WHERE Opportunity\_\_c IN: oppIds];

        List<Invoice\_\_c> lstInvoice = new List<Invoice\_\_c>();

        For(Opportunity\_Automobile\_\_c oppAuto : lstOpportunityAutomobile){

            Invoice\_\_c i = new Invoice\_\_c();

            i.Quantity\_\_c = oppAuto.Quantity\_\_c;

            i.Unit\_Price\_\_c = oppAuto.Unit\_Price\_\_c;

            i.Total\_Price\_\_c = oppAuto.Total\_Price\_\_c;

            i.Purchase\_Date\_\_c = date.today();

            i.Opportunity\_\_c = oppAuto.Opportunity\_\_c;

            lstInvoice.add(i);

        }

        if(!lstInvoice.isempty()){

            insert lstInvoice;

        }

    }

}

**Trigger Code**:

trigger OpportunityTrigger on Opportunity (before update, after update) {

if (Trigger.isBefore && Trigger.isUpdate) {

OpportunityHandlerClass.opportunityAutomobileQuantity(Trigger.new, Trigger.oldMap);

}

if (Trigger.isAfter && Trigger.isUpdate) {

InvoiceCreation.OpportunityClosedwonInvoiceGeneration(Trigger.new, Trigger.oldMap);

}

}

#### Use Case 4: Ensuring Contact Role is Assigned

* **Objective**: Before marking an Opportunity as **Closed Won**, check if a **Contact Role** is associated with it.
* **Class**: ContactRoleCheck class handles the logic for this validation, called from within **OpportunityTrigger**.

**Trigger Code**:

trigger OpportunityTrigger on Opportunity (before update, after update) {

if (Trigger.isBefore && Trigger.isUpdate) {

OpportunityHandlerClass.opportunityAutomobileQuantity(Trigger.new, Trigger.oldMap);

ContactRoleCheck.CheckcontactRoleonOpportunity(Trigger.new, Trigger.oldMap);

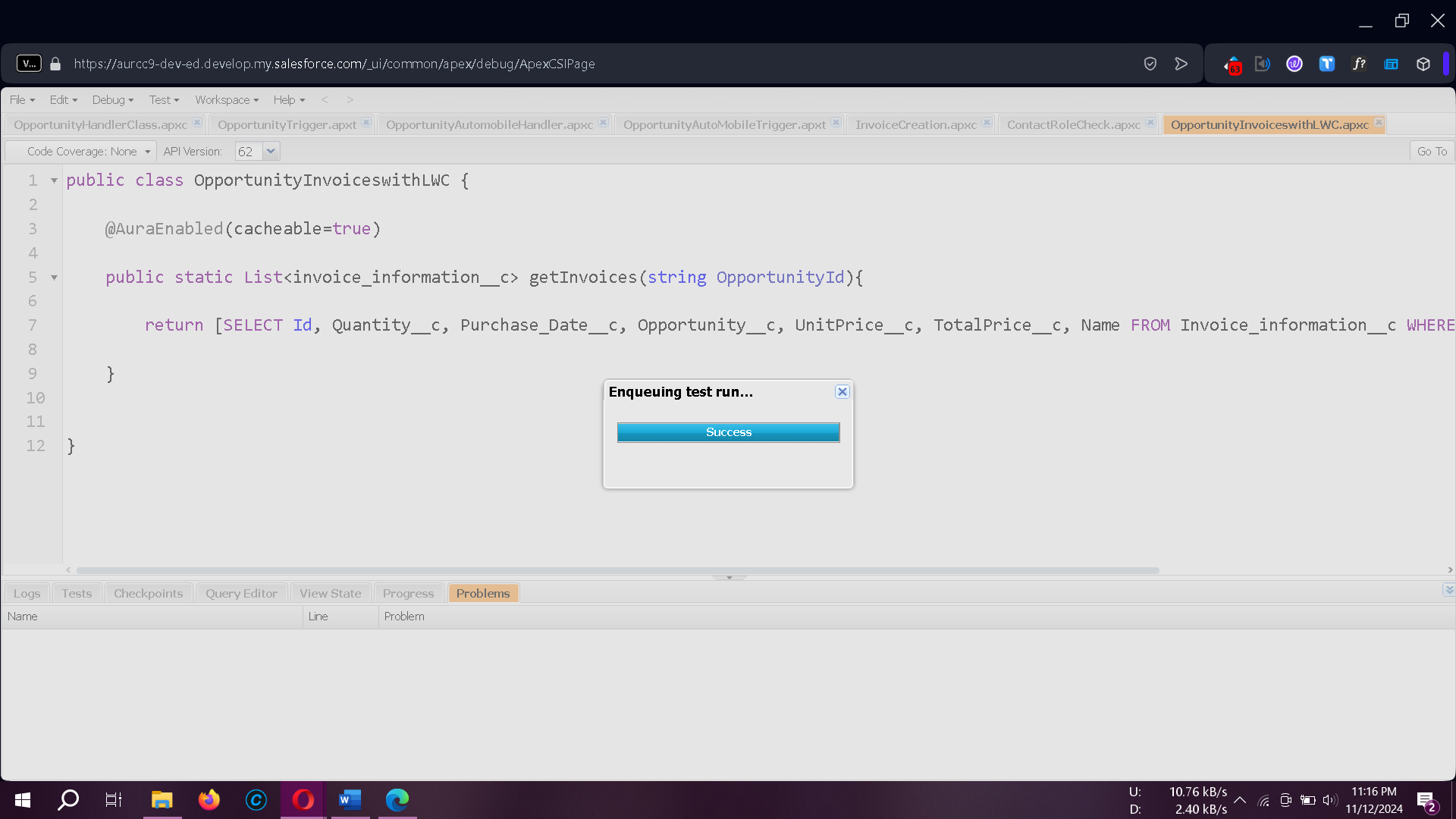
}

if (Trigger.isAfter && Trigger.isUpdate) {

InvoiceCreation.OpportunityClosedwonInvoiceGeneration(Trigger.new, Trigger.oldMap);

}

}



### ****LWC Component: Creating an Apex Class to Retrieve Invoices****

1. **Log into Salesforce**  
   Access your Salesforce account and click the gear icon in the upper-right corner. From the dropdown menu, select **Developer Console** to open a new console window.
2. **Create a New Apex Class**  
   In the Developer Console, go to **File > New > Apex Class** and name it OpportunityInvoiceswithLWC.

**Apex Class Code**:

public class OpportunityInvoiceswithLWC {

@AuraEnabled(cacheable=true)

public static List<Invoice\_\_c> getInvoices(String OpportunityId) {

return [

SELECT Id, Quantity\_\_c, Purchase\_Date\_\_c, Opportunity\_\_c, Unit\_Price\_\_c, Total\_Price\_\_c, Name

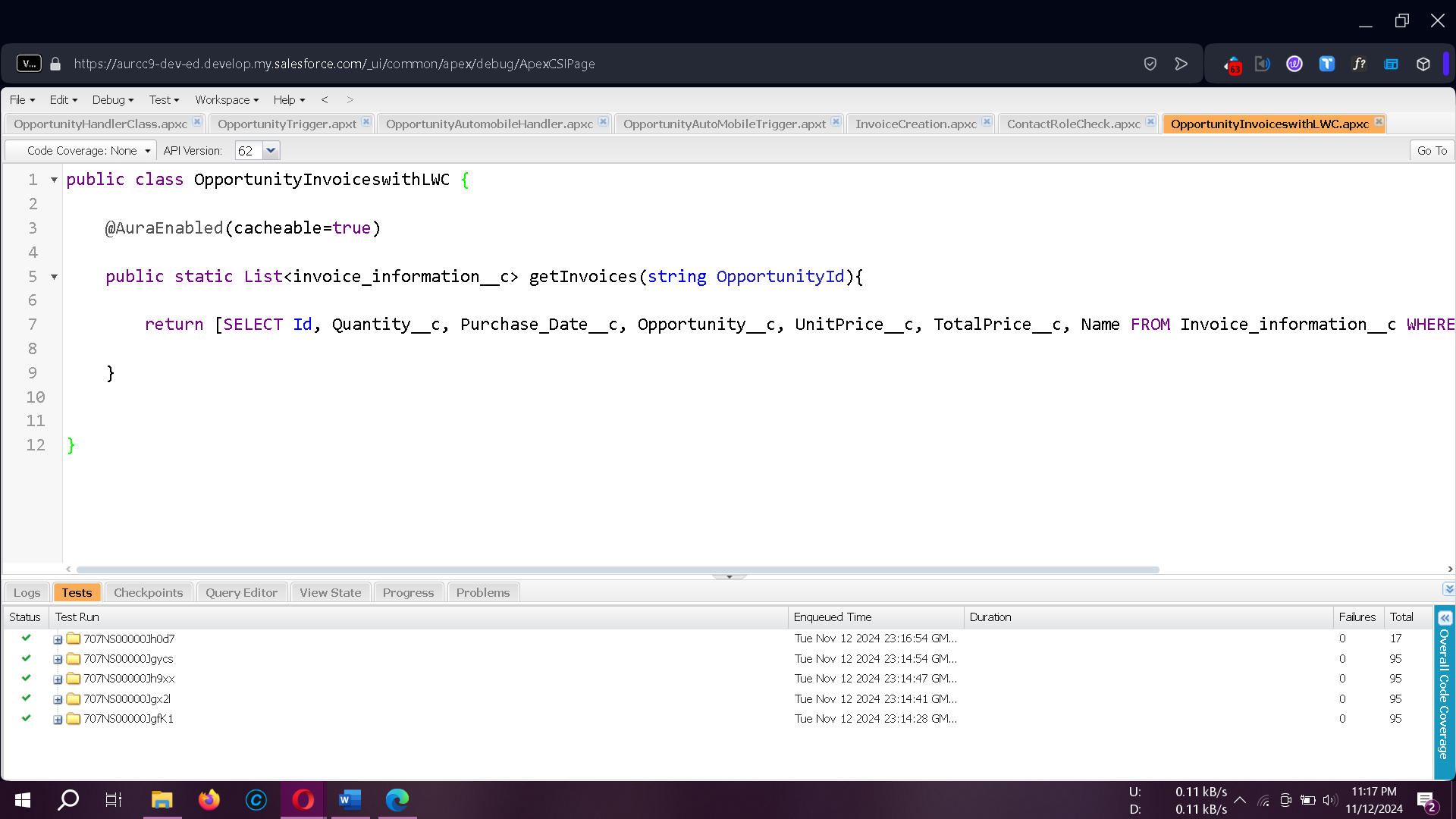
FROM Invoice\_\_c

WHERE Opportunity\_\_c = :OpportunityId

];

}

}



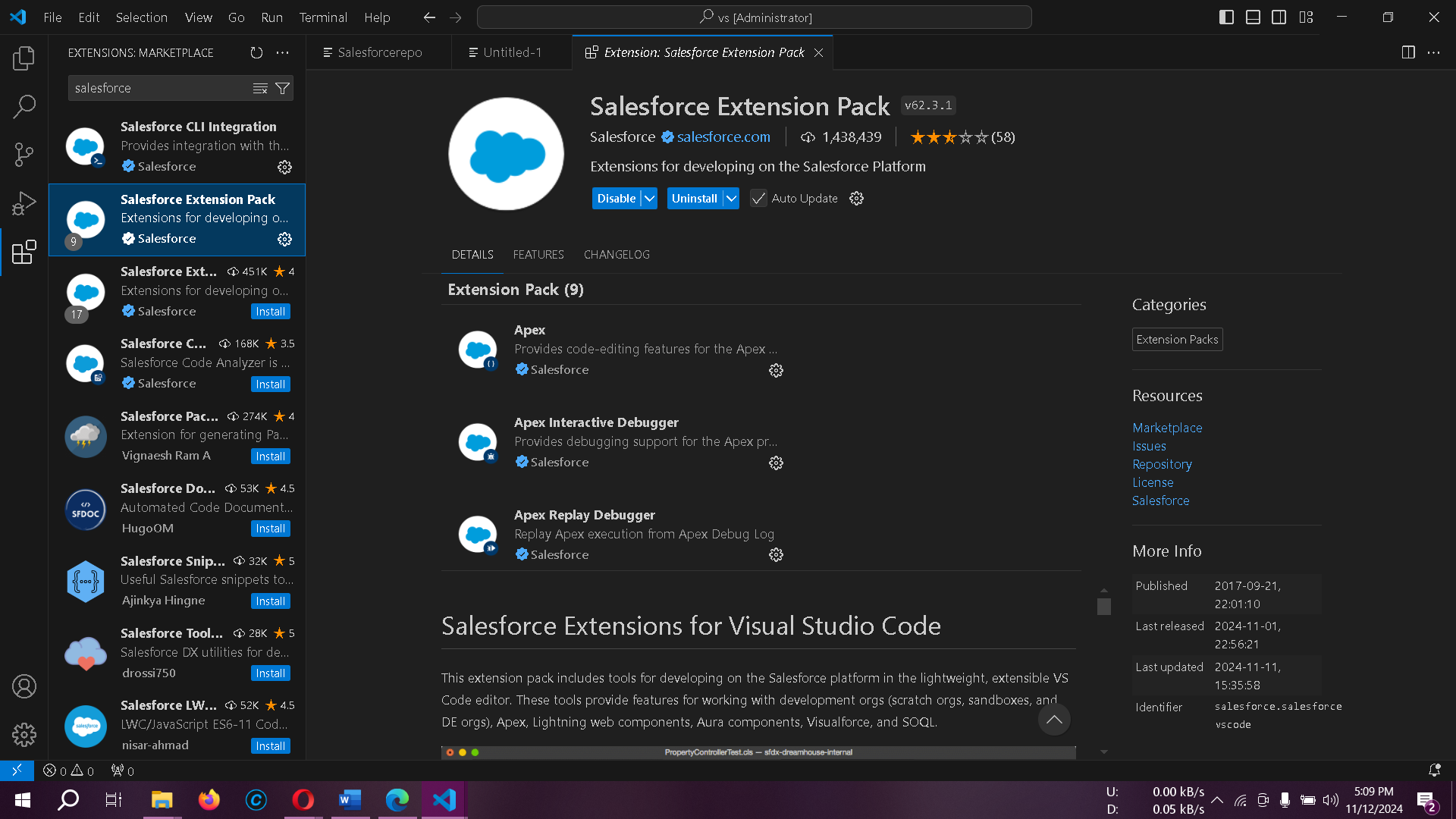
### ****Setting Up the Salesforce CLI****

1. **Download and Install**  
   The Salesforce CLI is essential for streamlining development. Download and install it, and verify the installation by typing sfdx in your command prompt.



### ****Installing and Setting Up Microsoft Visual Studio Code (VS Code)****

1. **Download and Install VS Code**  
   Ensure you download the version compatible with your operating system.
2. **Install the Salesforce Extension Pack**  
   Open VS Code, go to Extensions, and search for **Salesforce Extension Pack**. Click **Install**.



### ****Creating a Salesforce Project in VS Code****

1. **Create Project**  
   Press CTRL + SHIFT + P, type sfdx: create project with manifest, select a project template, and follow the prompts to name and save the project.
2. **Update Package.xml**  
   Update your package.xml file with the following code to work with LWC components:

xml

Copy code

<Package xmlns="http://soap.sforce.com/2006/04/metadata">

<types>

<members>\*</members>

<name>LightningComponentBundle</name>

</types>

<version>55.0</version>

</Package>

1. **Authorize an Org**  
   Establish a connection with your Salesforce org by pressing CTRL + SHIFT + P, typing sfdx: authorize an org, and selecting **Production** for the developer edition.

### ****Creating the Lightning Web Component****

1. **Create LWC Component**  
   Press CTRL + SHIFT + P, type sfdx: create lightning web component, and name it InvoiceOpportunity.
2. **JS File Code for LWC**  
   Copy and paste the following code into InvoiceOpportunity.js:

javascript

Copy code

import { LightningElement, api, track, wire } from 'lwc';

import getInvoices from '@salesforce/apex/OpportunityInvoiceswithLWC.getInvoices';

export default class InvoiceOpportunity extends LightningElement {

@api recordId;

@track invoiceCollection;

cols = [

{label: "ID", fieldName: 'Name'},

{label: "Opportunity Id", fieldName: 'Opportunity\_\_c'},

{label: "Quantity", fieldName: 'Quantity\_\_c'},

{label: "Unit Price", fieldName: 'Unit\_Price\_\_c'},

{label: "Total Price", fieldName: 'Total\_Price\_\_c'},

{label: "Purchase Date", fieldName: 'Purchase\_Date\_\_c'}

];

@wire(getInvoices, { OpportunityId: '$recordId' })

invoiceFunction({data, error}) {

if(data) this.invoiceCollection = data;

if(error) console.log('Error fetching data');

}

}

1. **HTML File Code for LWC**  
   Copy the following code into InvoiceOpportunity.html:

html

Copy code

<template>

<lightning-card title="Opportunity Invoices">

<lightning-datatable

key-field="Id"

data={invoiceCollection}

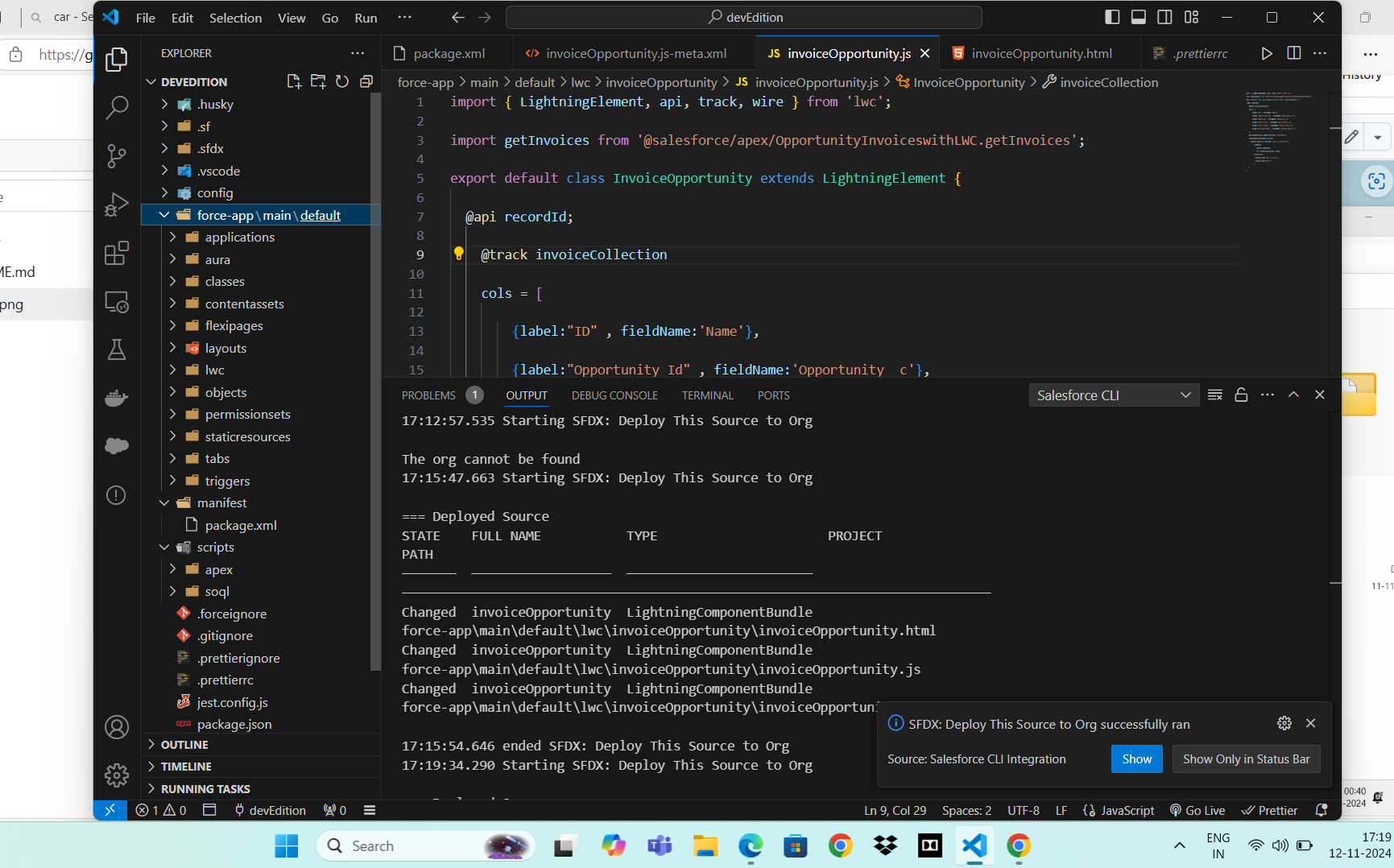
columns={cols}>

</lightning-datatable>

</lightning-card>

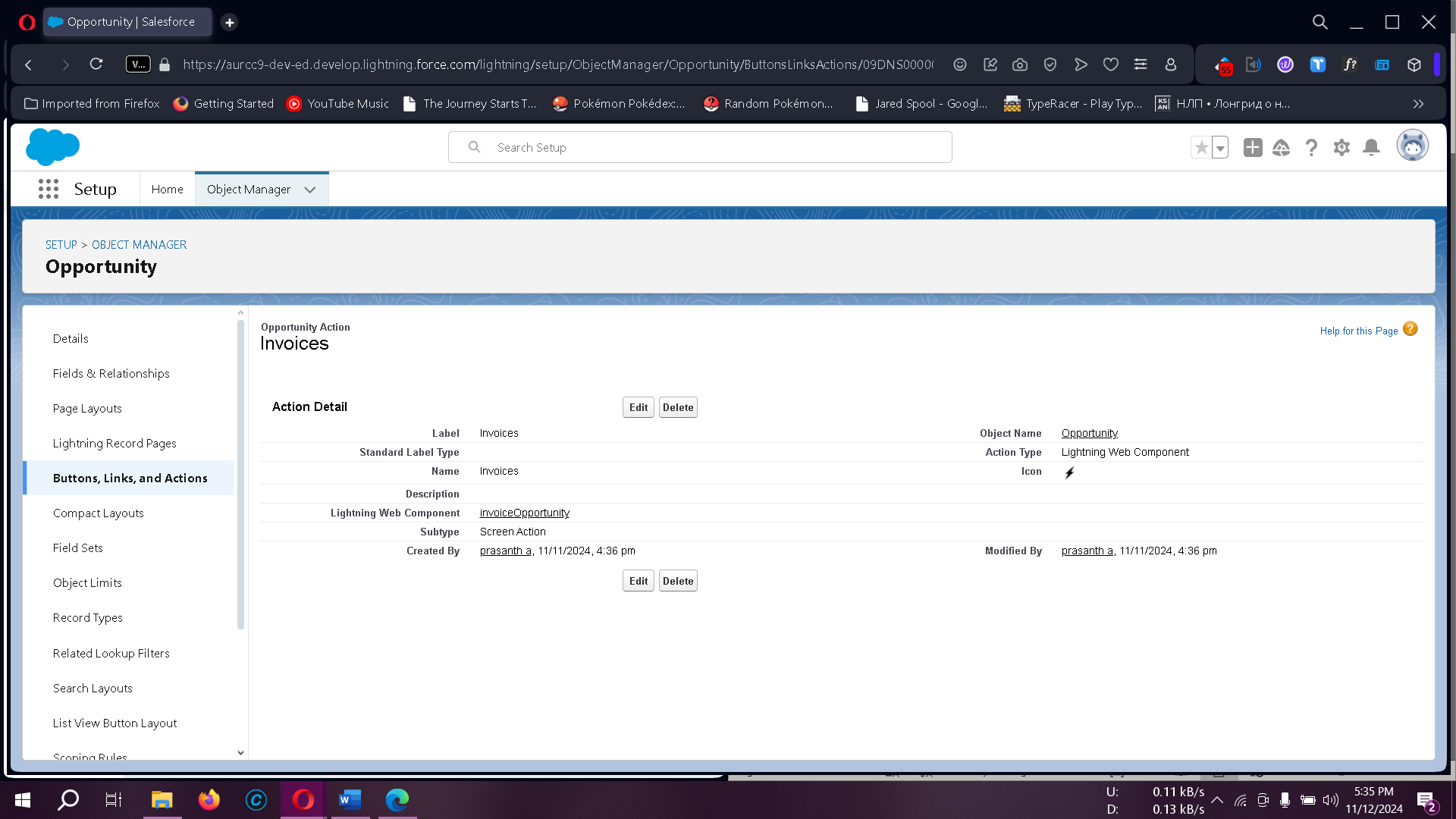
</template>

1. **Deploy Component**  
   Right-click on the component folder and select **SFDX: Deploy Source to Org**.



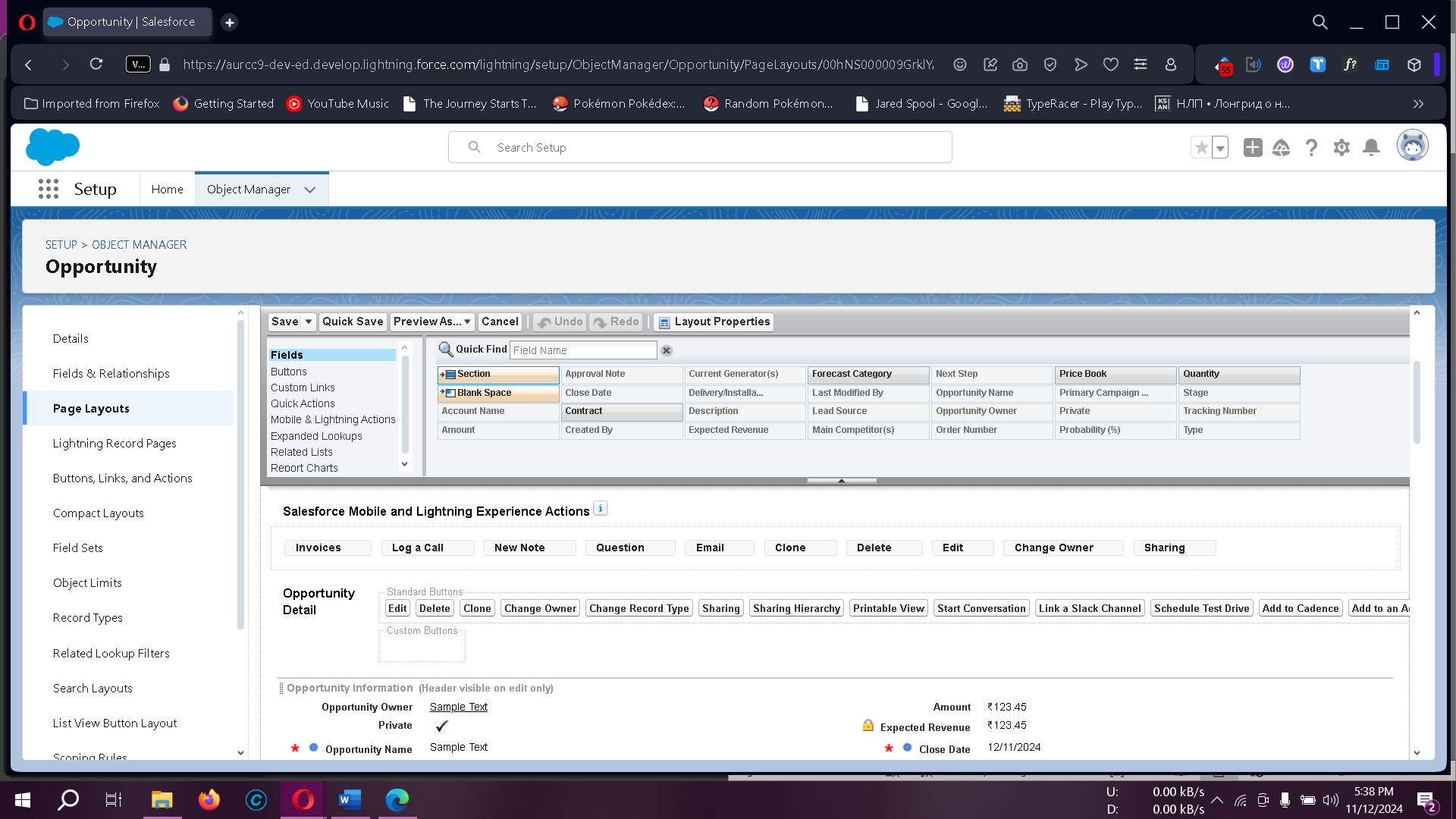
### ****Adding Component to Opportunity Record Page****

1. **Add Action Button**  
   In Salesforce Setup, go to **Object Manager > Opportunity > Buttons, Links, and Actions**. Click **New Action**, select **Lightning Web Component**, and choose InvoiceOpportunity.



### Adding Invoice Opportunity to Opportunity Record Page

1. In **Opportunity Object Manager**, go to **Page Layouts** and select **Opportunity Layout**.
2. Under **Mobile and Lightning Actions**, search for **Invoice** in **Quick Find**.
3. Drag and drop **Invoice** into the Salesforce Mobile and Lightning Experience actions section.
4. Click **Save**.



### ****Scheduling Apex for Closed Opportunities****

1. **Create Apex Class for Scheduler**  
   In Developer Console, create a new Apex class called DeleteClosedLostOpportunities.

**Apex Code**:

apex

Copy code

public class DeleteClosedLostOpportunities implements Schedulable {

public void execute(SchedulableContext sc) {

List<Opportunity> lostOpps = [

SELECT Id FROM Opportunity WHERE StageName = 'Closed Lost'

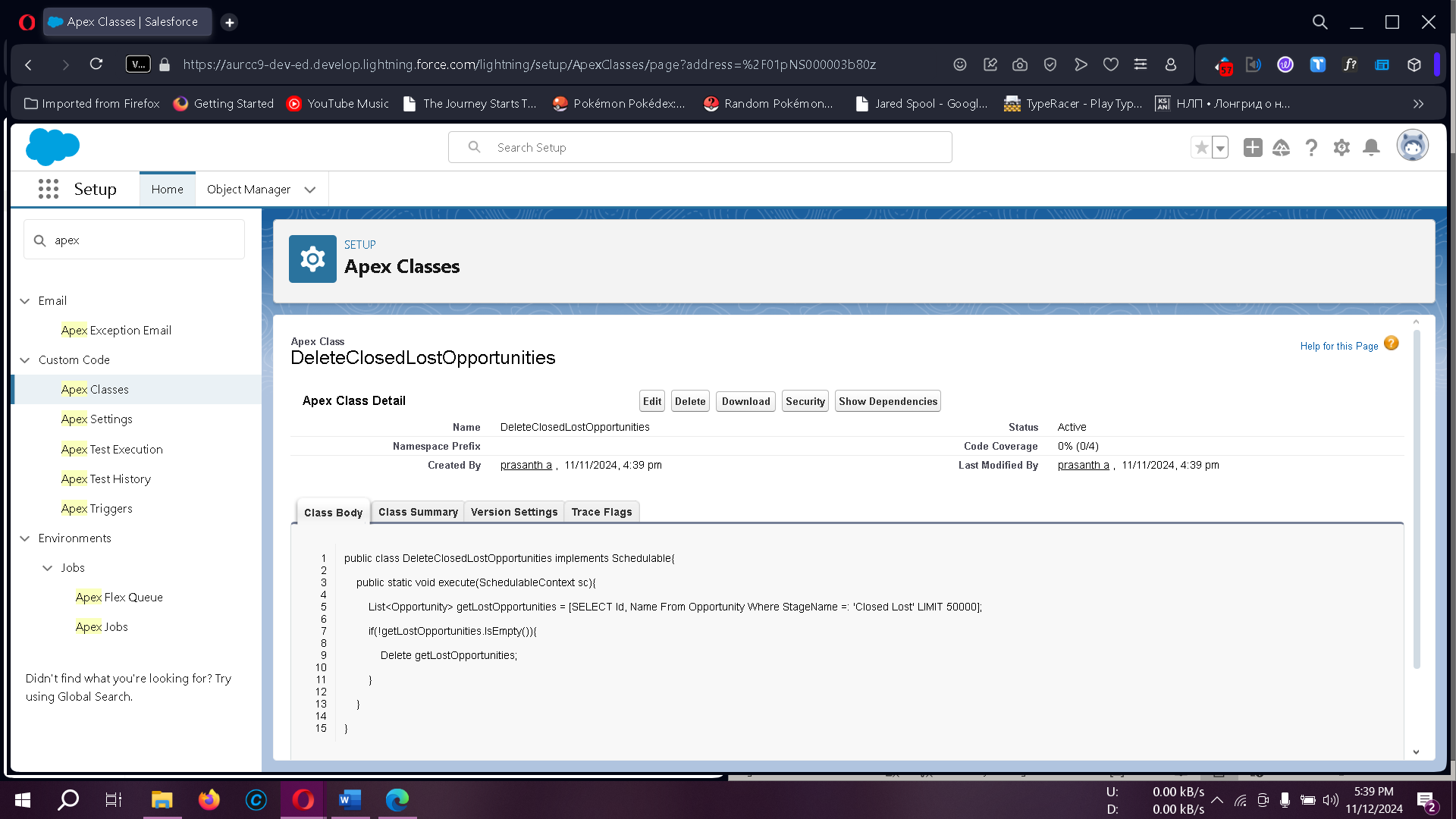
];

delete lostOpps;

}

}

1. **Schedule the Class**  
   Go to **Setup > Apex Classes > Schedule Apex** and set it to run every Monday at 10:00 AM.



### ****Creating Salesforce Reports****

## Reports give you access to your Salesforce data. You can examine your Salesforce data in almost infinite combinations, display it in easy-to-understand formats, and share the resulting insights with others. Before building, reading, and sharing reports, review these reporting basics.

**Types of Reports in Salesforce**

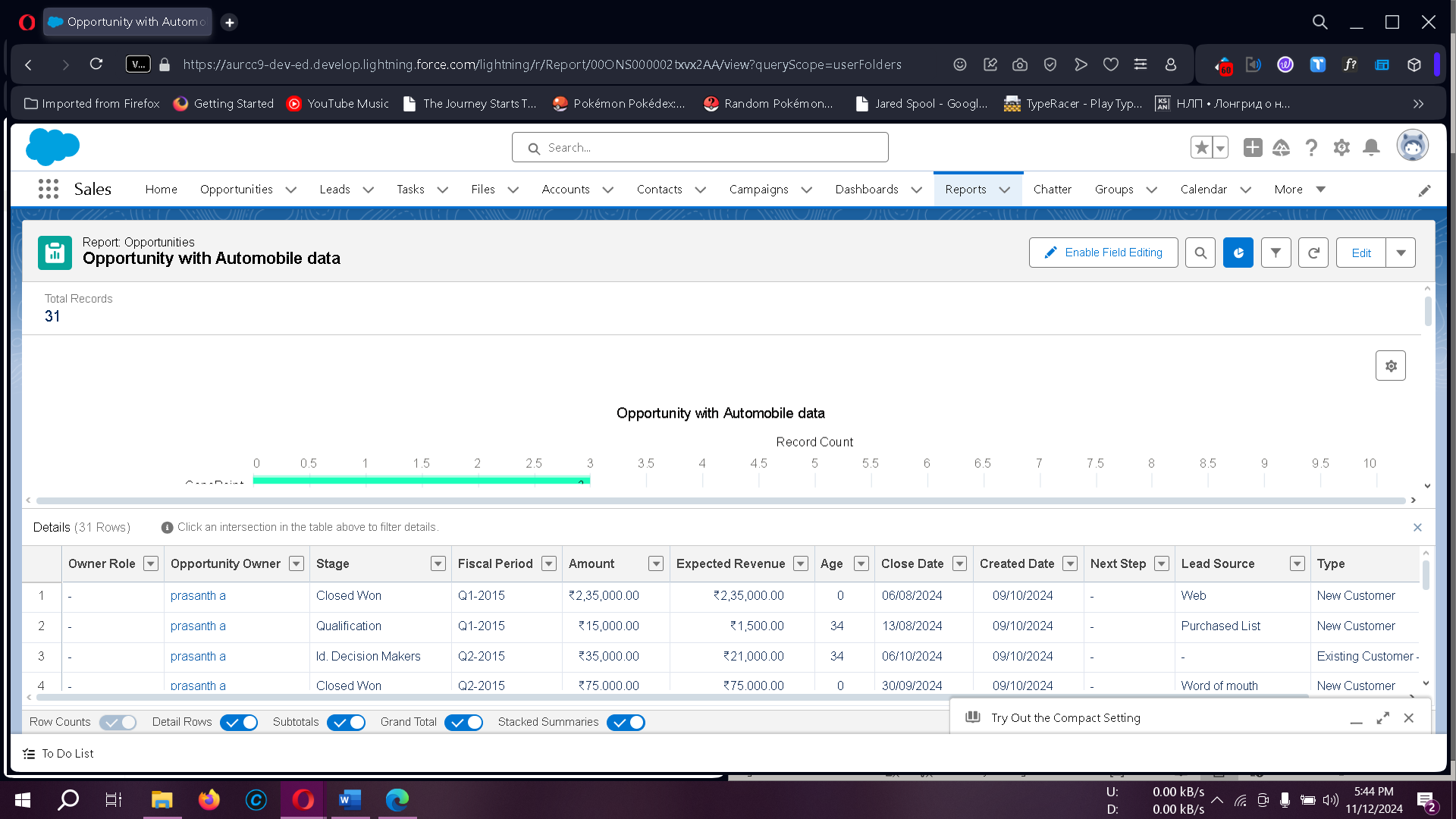
1. **Tabular**
2. **Summary**
3. **Matrix**
4. **Joined Reports**

#### Steps to Create a Report on Opportunity:

1. Navigate to the **Reports** tab and click **New Report**.
2. Select the report type and click **Start Report**.
3. Customize the report by adding fields and applying filters.
4. Save or run the report.

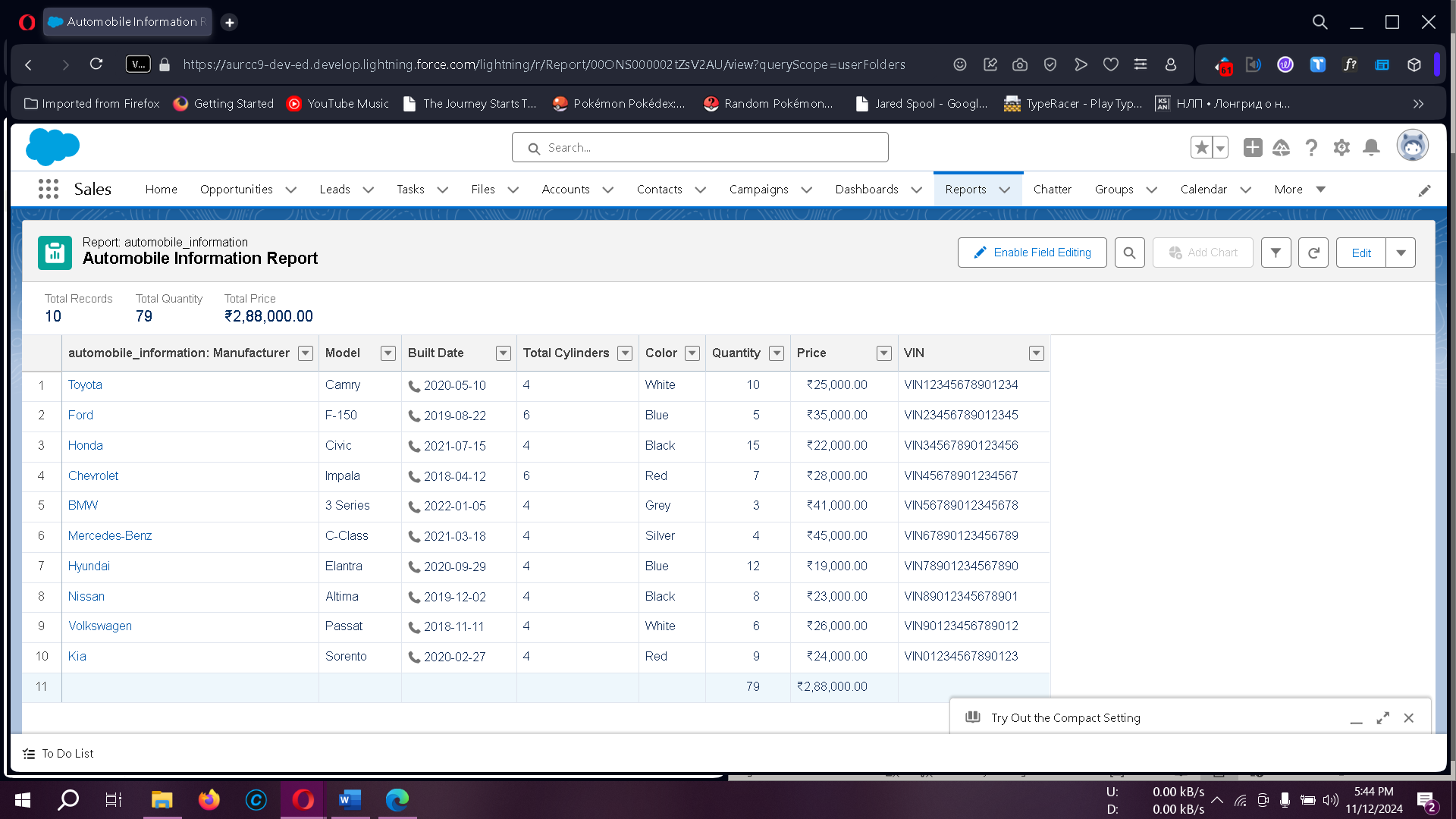
#### Create a Report on Automobile Information:

* Choose the **Automobile Information** report type and apply filters.
* **Opportunity Sales Report**  
  Go to **Reports > New Report**, select **Opportunity** as the report type, and customize the fields and filters.



#### Create a Report on Opportunity Automobiles:

* Use the **Opportunities with Opportunity Automobiles and Automobile** report type.
* Select the report type: **Automobile Information**.
* Apply filters as needed.



### Creating a Dashboard in Salesforce

Dashboards provide a visual summary of real-time data, enabling users to quickly understand business trends, monitor performance, and make informed decisions. They allow easy access to report data through visual components.

#### Steps to Create a Sales Dashboard:

1. **Automobile Sales Dashboard**  
   Go to **Dashboards > New Dashboard**, name it "Automobile Sales," add components, and save it.

The created dashboard will display the selected report's data in a visual format.



### 10. Conclusion

The Salesforce CRM setup for automobile sales enables a structured and accessible system for managing sales data. By establishing a developer account, creating custom objects, configuring fields and relationships, and setting up Lightning Apps, businesses can efficiently handle data management and streamline their sales processes. This system provides a robust, scalable solution for any organization involved in automobile sales, leveraging Salesforce’s CRM capabilities for optimal productivity and data control.