salesforce
DEVELOPERS

UNIT-4

By
Mrs. P.SHYAMALA

UNIT–IV

Salesforce User Interface:

Lightning Web Components Basics:

Discover Lightning Web Components,

Create Lightning Web Components,

Deploy Lightning Web Component Files,

Handle Events in Lightning Web Components,

Add Styles and Data to a Lightning

Web Component Secure Server-Side Development:

Write Secure Apex Controllers,

Mitigate SOQL Injection,

Request Forgery.

Lightning Web Components Basics:- Discover Lightning Web Components

Why Lightning Web Components (LWC)?

- **Modern browsers** are based on web standards, and evolving standards are constantly improving what browsers can present to a user. We want you to be able to take advantage of these innovations.

- **Lightning Web Components uses core Web Components standards** and provides only what's necessary to **perform well in browsers supported by Salesforce.**

- Because it's built on code that runs natively in browsers, **Lightning Web Components is lightweight and delivers exceptional performance**. Most of the code you write is **standard JavaScript and HTML.**

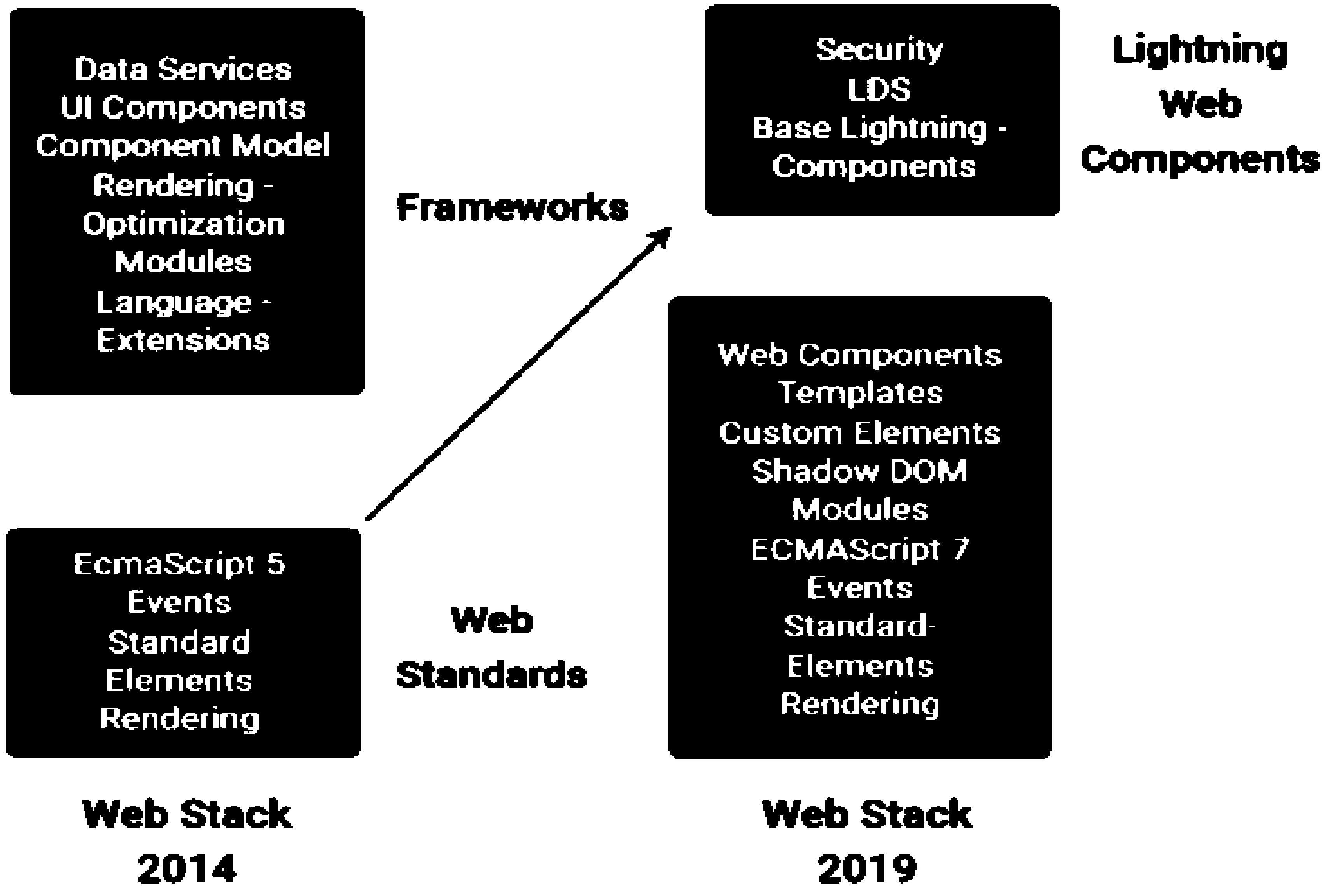
- Most of the **code we write to create component is standard JavaScript and HTML.**

We'll find it easier to:

- Find solutions in common places on the web.
- Find developers with necessary skills and experience.
- Use other developers' experiences (even on other platforms).
- Develop faster.
- Utilize full encapsulation so components are more versatile.

Lightning Web Components Basics:- Discover Lightning Web Components

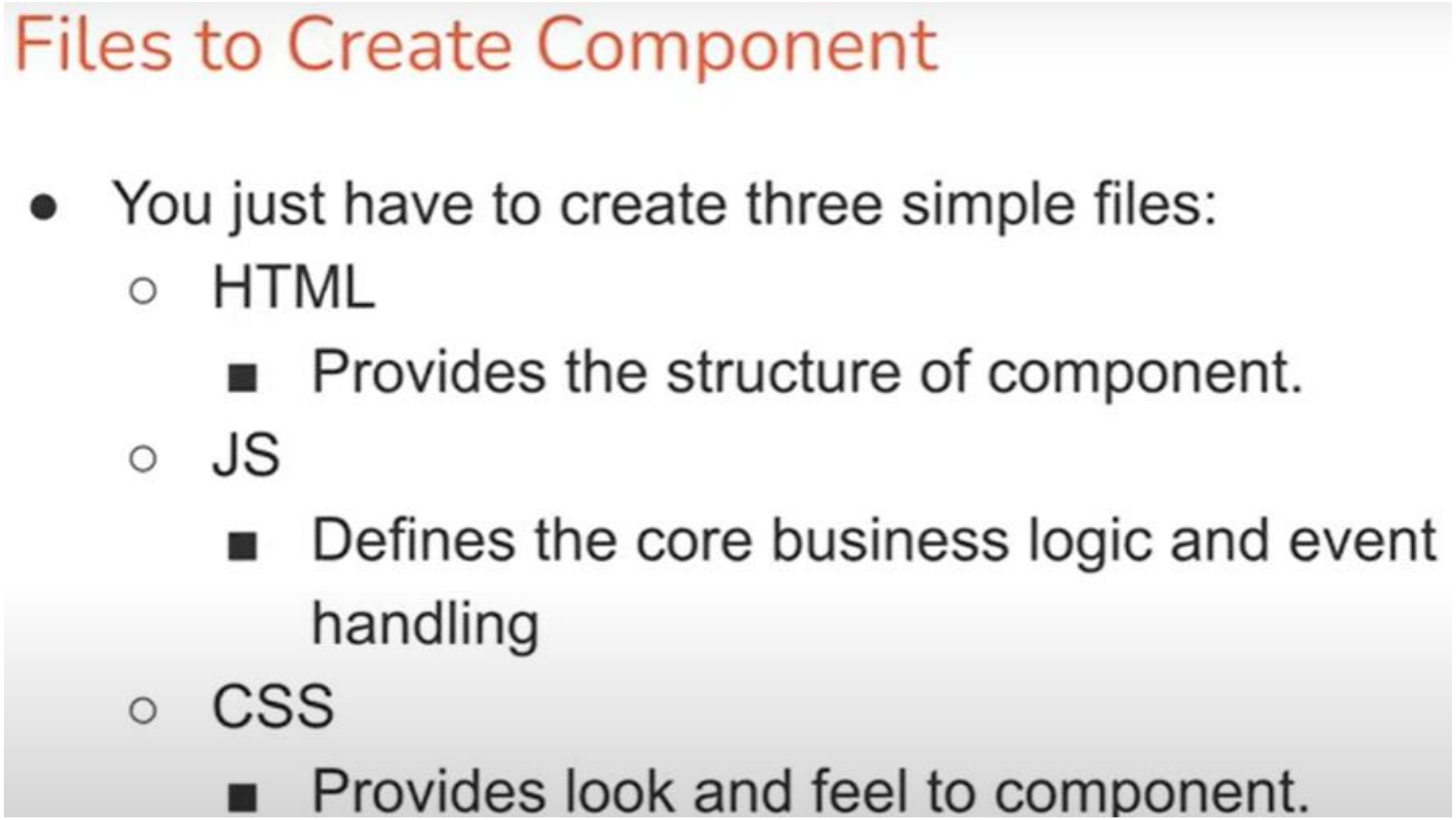
Why Lightning Web Components?



Lightning Web Components Basics:- Discover Lightning Web Components

Simple Component Creation

We simply create components using (1) a JavaScript file, (2) an HTML file, and optionally (3) a CSS file.



Files to Create Component

- You just have to create three simple files:
  - HTML
    - Provides the structure of component.
  - JS
    - Defines the core business logic and event handling
  - CSS
    - Provides look and feel to component.

Lightning Web Components Basics:- Discover Lightning Web Components

EXAMPLE:

Simple Lightning web component that displays “Hello World” in an input field.

HTML

```
<template>
  <input value={message}></input>
</template>
```

The **template tag is a fundamental building block** of a component's HTML. It allows you to store pieces of HTML.

JavaScript

```
import { LightningElement } from 'lwc';
export default class App extends LightningElement {
  message = 'Hello World';
}
```

CSS

```
Input { color : blue; }
```

We need an **HTML file and a JavaScript file with the same name in the same folder (also with a matching name). We deploy those to an org with some metadata . Salesforce compiles** our files and takes care of the component construction for us automatically.

LWC Module, Lifecycle Hooks and Decorators

The LWC Module:

Lightning Web Components uses modules (built-in modules were introduced in ECMAScript 6) to bundle core functionality and make it accessible to the JavaScript in your component file. The core module for Lightning web components is lwc.
Begin the module with the import statement and specify the functionality of the module that your component uses.
The import statement indicates the JavaScript uses the LightningElement functionality from the lwc module.

```
// import module elements
import { LightningElement} from 'lwc';
// declare class to expose the component
export default class App extends LightningElement {
  ready = false;
  // use lifecycle hook
  connectedCallback() {
    setTimeout(() => {
      this.ready = true;
    }, 3000);  }}
```

LightningElement is the base class for Lightning web components, which allows us to use connectedCallback().
The connectedCallback() method is one of our lifecycle hooks.

LWC Module, Lifecycle Hooks and Decorators

Lifecycle Hooks:

Lightning Web Components provides methods that allow you to “hook” your code up to critical events in a component's lifecycle. These events include when a component is:

- Created
- Added to the DOM
- Rendered in the browser
- Encountering errors
- Removed from the DOM

Respond to any of these lifecycle events using callback methods. For example, the connectedCallback() is invoked when a component is inserted into the DOM. The disconnectedCallback() is invoked when a component is removed from the DOM.

In the JavaScript file we used to test our conditional rendering, we used the connectedCallback() method to automatically execute code when the component is inserted into the DOM. The code waits 3 seconds, then sets ready to true.

```
import { LightningElement } from 'lwc';
export default class App extends LightningElement {
  ready = false;
  connectedCallback() {
    setTimeout(() => {
      this.ready = true;
    }, 3000);  }}
```

LWC Module, Lifecycle Hooks and Decorators

Decorators:

Decorators are often used in JavaScript to modify the behavior of a property or function.

To use a decorator, import it from the lwc module and place it before the property or function.

```
import { LightningElement, api } from 'lwc';
export default class MyComponent extends LightningElement{
  @api message; }
```

You can import multiple decorators, but a single property or function can have only one decorator. For example, a property can't have @api and @wire decorators.

Examples of Lightning Web Components decorators include:

@api: Marks a field as public. Public properties define the API for a component. An owner component that uses the component in its HTML markup can access the component's public properties. All public properties are reactive, which means that the framework observes the property for changes. When the property's value changes, the framework reacts and rerenders the component.

LWC Module, Lifecycle Hooks and Decorators

@track: Tells the framework to observe changes to the properties of an object or to the elements of an array. If a change occurs, the framework rerenders the component. All fields are reactive. If the value of a field changes and the field is used in a template—or in the getter of a property used in a template—the framework rerenders the component. You don't need to decorate the field with @track. Use @track only if a field contains an object or an array and if you want the framework to observe changes to the properties of the object or to the elements of the array. If you want to change the value of the whole property, you don't need to use @track.

@wire: Gives you an easy way to get and bind data from a Salesforce org.

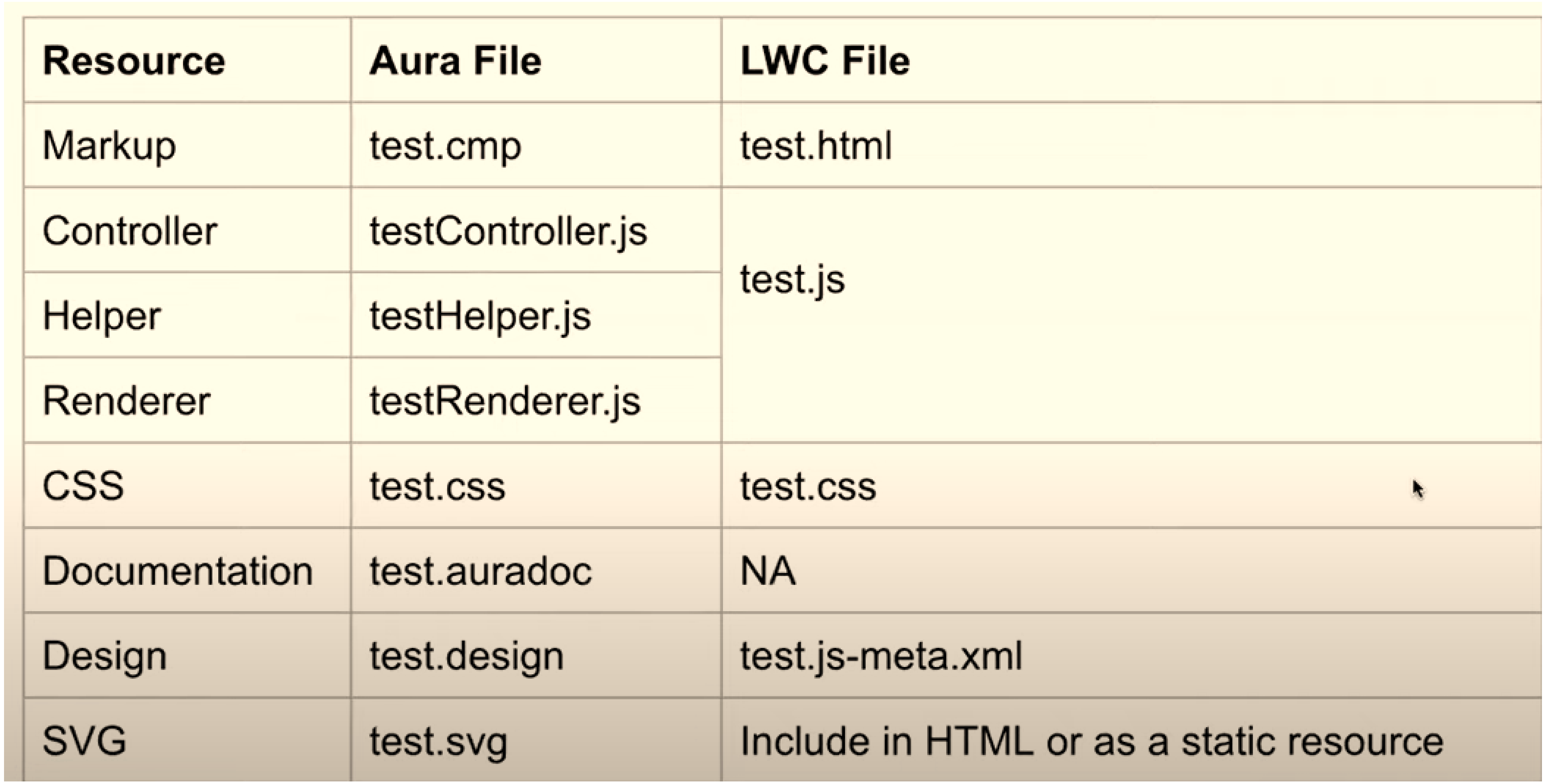
Lightning Web Components Basics:- Discover Lightning Web Components

Lightning Web Components and Aura Components Do Work Together?

- We can build Lightning components using **two programming models: Lightning Web Components and the original model, Aura Components.**
- **Yes we can! we can use Lightning web components without giving up our existing Aura components**
- Aura components and Lightning web components can **coexist and interoperate**, and **they share the same high-level services:**
- Aura components and Lightning web components **can coexist on the same page**
- **Aura components can include Lightning web components but Lightning web components cannot contain Aura components**

Lightning Web Components Basics - Discover Lightning Web Components

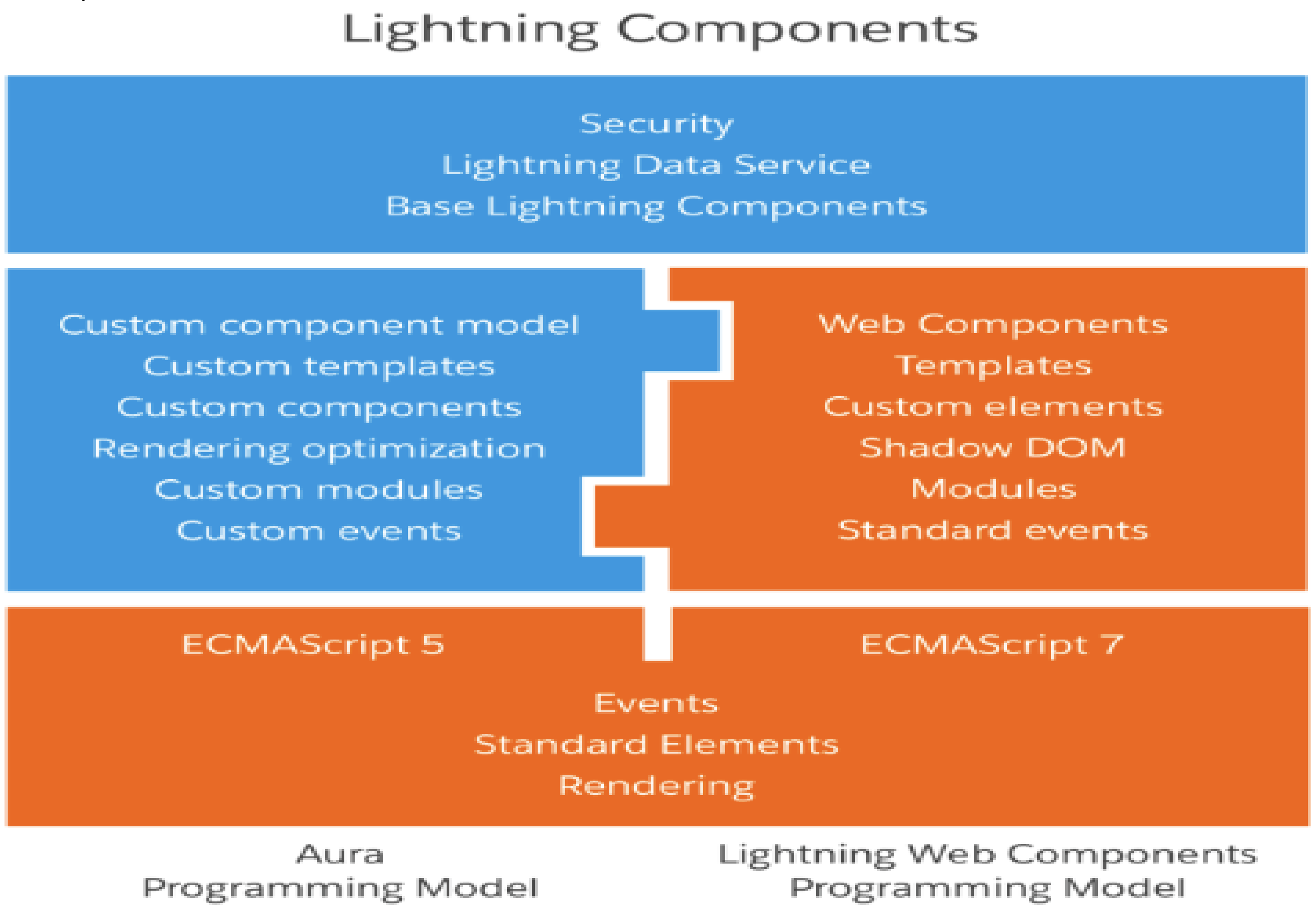
Lightning Aura Vs LWC Component Bundles:



Resource	Aura File	LWC File
Markup	test.cmp	test.html
Controller	testController.js	test.js
Helper	testHelper.js	
Renderer	testRenderer.js	
CSS	test.css	test.css
Documentation	test.auradoc	NA
Design	test.design	test.js-meta.xml
SVG	test.svg	Include in HTML or as a static resource

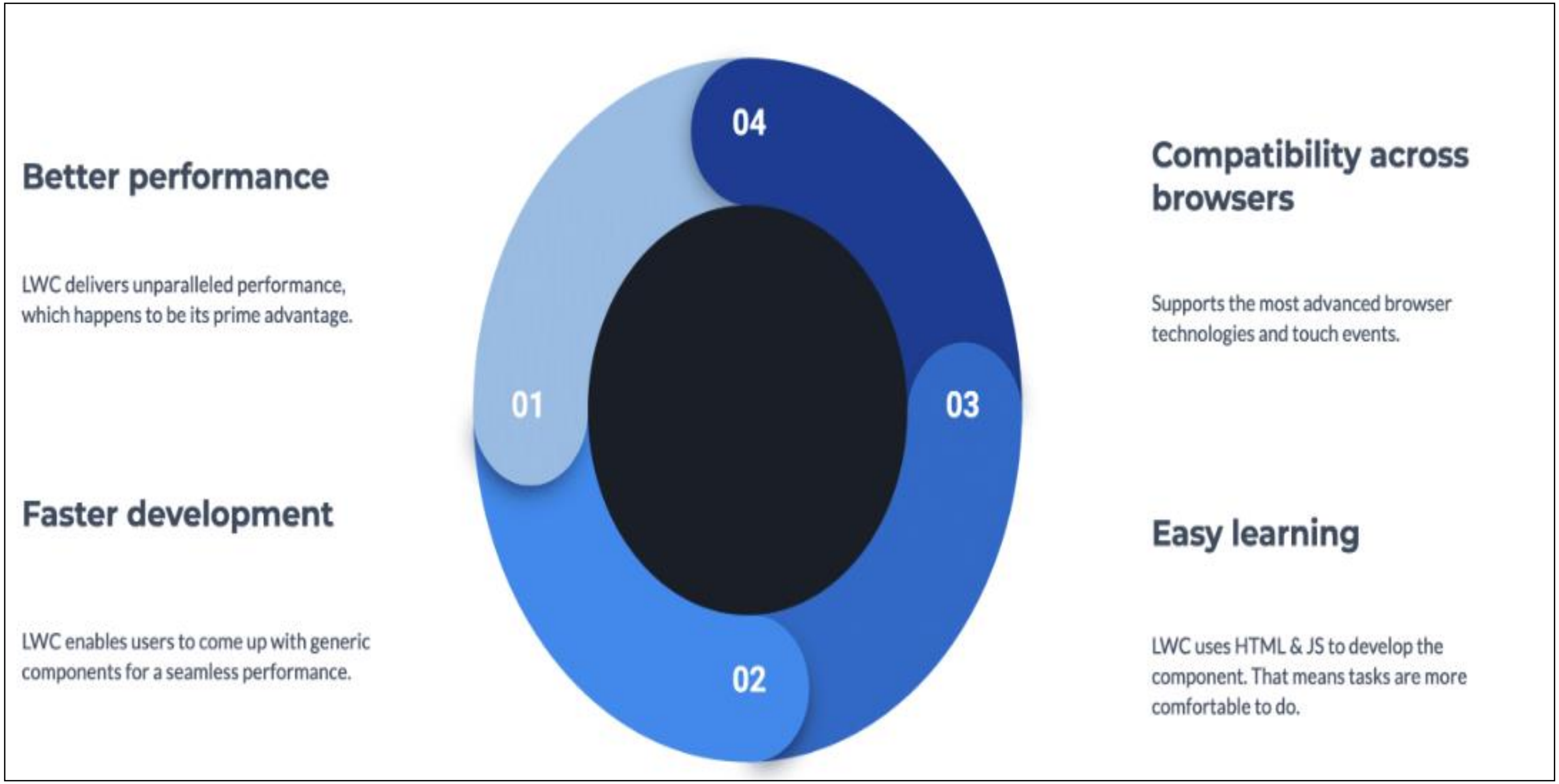
Lightning Web Components Basics:- Discover Lightning Web Components

- Aura components and Lightning web components **share the same base Lightning components**. Base Lightning components have already been implemented as Lightning web components.
- Aura components and Lightning web components **share the same underlying services** (Lightning Data Service, User Interface API,



Lightning Web Components Basics:- Discover Lightning Web Components

Advantages of using LWC



Better performance

LWC delivers unparalleled performance, which happens to be its prime advantage.

Faster development

LWC enables users to come up with generic components for a seamless performance.

Compatibility across browsers

Supports the most advanced browser technologies and touch events.

Easy learning

LWC uses HTML & JS to develop the component. That means tasks are more comfortable to do.

Lightning Web Components Basics:- Discover Lightning Web Components

To develop Lightning web components efficiently, use the following tools and environments.

The tools used in Lightning Web Components (LWC) development include:

1.Visual Studio Code Salesforce Extension Pack:

A **popular code editor with Salesforce Extensions for LWC development**. We've focused on Visual Studio as a development tool, providing an **integrated environment** for you to build your components. The Salesforce Extension Pack for Visual Studio Code provides code-hinting, lint warnings, and built-in commands:

2. Salesforce CLI:

Command-line interface for managing Salesforce development tasks, including LWC.

3. Code Builder

Salesforce Code Builder is a **web-based integrated development environment** that has all the power and flexibility of Visual Studio Code, Salesforce Extensions for VS Code, and Salesforce CLI in your web browser.

NOTE: VS Code is a powerful IDE but it requires you to **install the app and related tools** on your computer. Code Builder enables you to use VS Code in your **browser without the need to install anything**.

Lightning Web Components Basics - Create Lightning Web Components

To develop Lightning web components efficiently, use the following tools and environments.

We develop a Lightning web component using Visual Studio Code with the Salesforce extension.

What You Need:

- Visual Studio Code(VS Code) with the Salesforce Extension Pack
- Salesforce CLI

Install the Command Line Interface (CLI):

Use the Salesforce CLI to control the full application and with it you can easily create environments for development and testing, synchronize source code between your orgs and version control system (VCS), and execute tests.

Lightning Web Components Basics - Create Lightning Web Components

Steps to install Salesforce CLI :

1. Install the CLI from Salesforce CLI.
2. Confirm the CLI is properly installed and on the latest version by running the following command from the command line.

```
sf update
```

3. You should see output like Updating CLI....
4. In our next step, we set up our local development environment with Visual Studio Code.

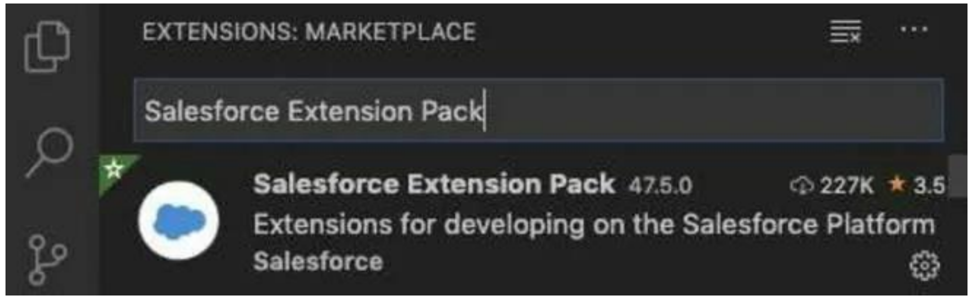
Lightning Web Components Basics - Create Lightning Web Components

Install Salesforce Extensions for Visual Studio Code:

Visual Studio Code is the go-to code editor for Salesforce developers. It's free, open-source, and available for Windows, Linux, and macOS. This editor has easy-to-install extensions for syntax highlighting, code completion, and more.

Install Visual Studio Code and the recommended Salesforce Extension Pack:

1. Download and install the latest version of Visual Studio Code for your operating system.
2. Launch Visual Studio Code.
3. On the left toolbar, click the Extensions icon
4. Search for Salesforce Extension Pack and click Install.

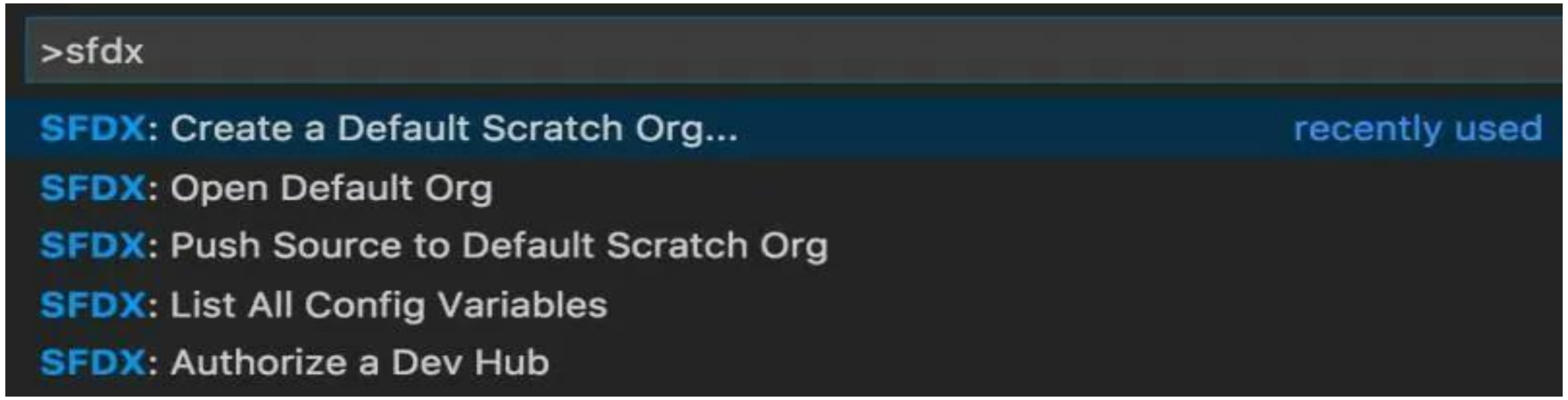

EXTENSIONS: MARKETPLACE
Salesforce Extension Pack
Salesforce Extension Pack 47.5.0
227K ★ 3.5
Extensions for developing on the Salesforce Platform
Salesforce

Lightning Web Components Basics - Create Lightning Web Components

Ensure Your Development Environment Is Ready:

Now that you've installed Visual Studio Code and enabled the necessary extensions, you need to test them out.

1. In Visual Studio Code, open the Command Palette by pressing Ctrl+Shift+P (Windows).
2. Enter sfdx to filter for commands provided by the Salesforce Extensions.



```
>sfdx
SFDX: Create a Default Scratch Org...                recently used
SFDX: Open Default Org
SFDX: Push Source to Default Scratch Org
SFDX: List All Config Variables
SFDX: Authorize a Dev Hub
```

3. In the final step, you create your first Lightning web component and add it to your org's home page.

Lightning Web Components Basics - Create Lightning Web Components

Create a Hello World Lightning Web Component:

Create a Salesforce DX Project:

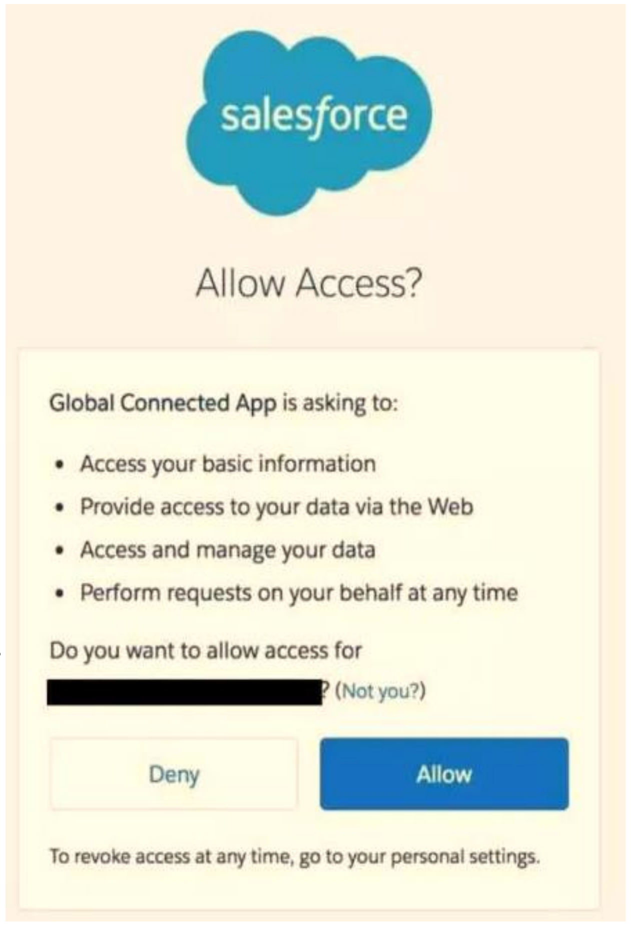
1. In Visual Studio Code, open the Command Palette by pressing Ctrl+Shift+P (Windows). Type SFDX.
2. Select SFDX: Create Project.
3. Press Enter to accept the standard option.
4. Enter **HelloWorldLightningWebComponent** as the project name.
5. Press Enter.
6. Select a folder to store the project.
7. Click Create Project.

Lightning Web Components Basics - Create Lightning Web Components

Authorize Your Org:

1. In Visual Studio Code, open the Command Palette by pressing Ctrl+Shift+P (Windows).
2. Type SFDX.
3. Select SFDX: Authorize an Org.
4. Press Enter to accept the Project Default login URL option.
5. Press Enter to accept the default alias.

   This opens the Salesforce login in a separate browser window.
6. Log in using your Salesforce credentials.
7. If prompted to allow access, click Allow.



Allow Access?

Global Connected App is asking to:

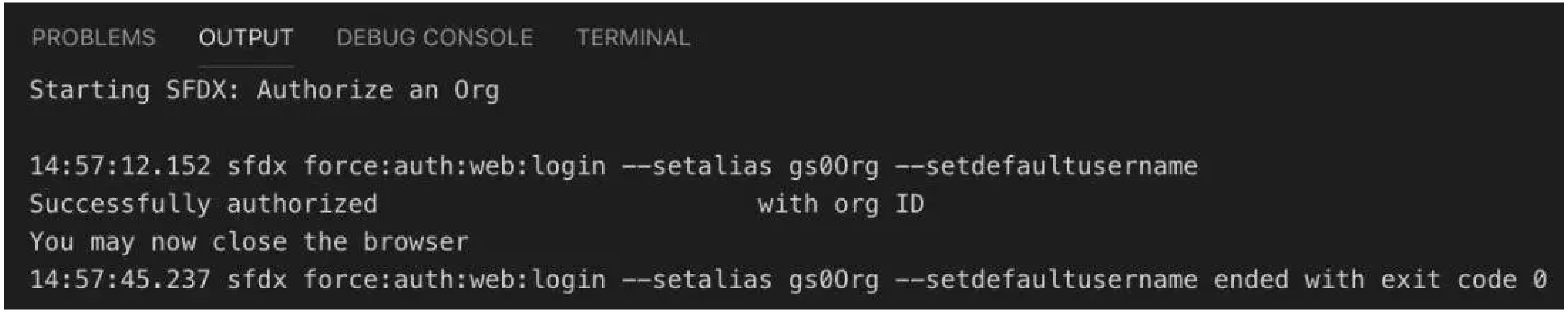
- Access your basic information
- Provide access to your data via the Web
- Access and manage your data
- Perform requests on your behalf at any time

Do you want to allow access for [redacted]? (Not you?)

To revoke access at any time, go to your personal settings.

Lightning Web Components Basics - Create Lightning Web Components

8. After you authenticate in the browser, the CLI remembers your credentials. The success message should look like this:



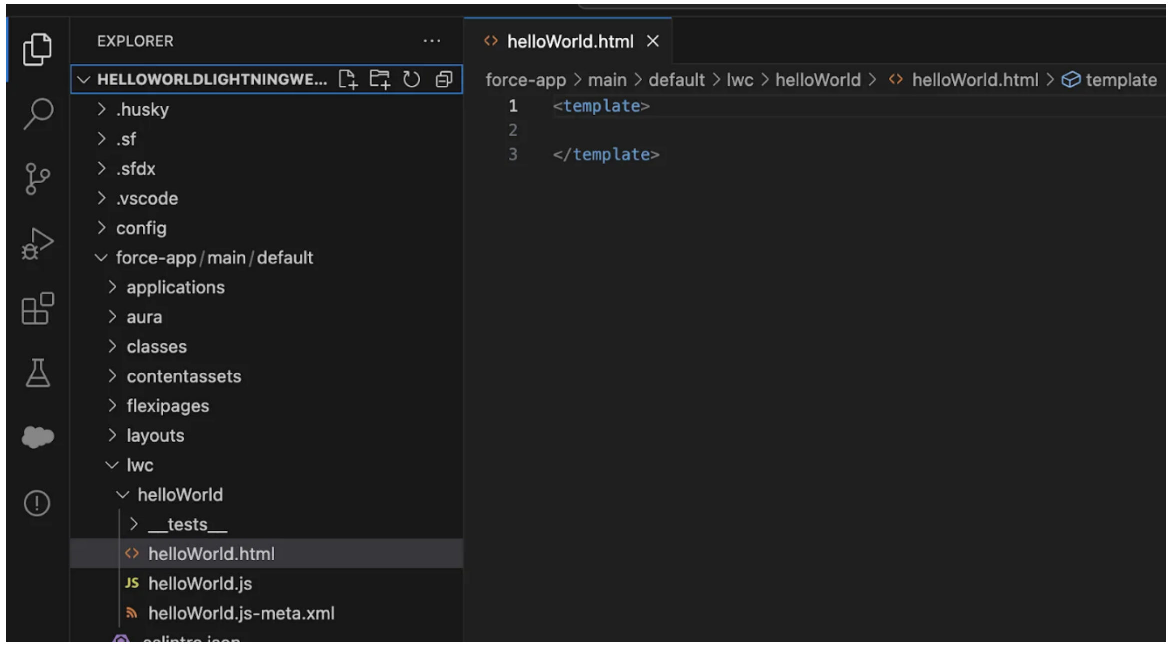
```
Starting SFDX: Authorize an Org

14:57:12.152 sfdx force:auth:web:login --setalias gs0Org --setdefaultusername
Successfully authorized                              with org ID
You may now close the browser
14:57:45.237 sfdx force:auth:web:login --setalias gs0Org --setdefaultusername ended with exit code 0
```

Lightning Web Components Basics - Create Lightning Web Components

Create a Lightning Web Component:

1. In Visual Studio Code, open the Command Palette by pressing Ctrl+Shift+P (Windows).
2. Type SFDX.
3. Select SFDX: Create Lightning Web Component.
4. Enter **helloWorld** for the name.
5. Press Enter to accept the default force-app/main/default/lwc.
6. Press Enter.
7. View the newly created files in Visual Studio Code.



```
<template>

</template>
```

Lightning Web Components Basics - Create Lightning Web Components

8. In the **HTML file**, **helloWorld.html**, copy and paste the following code.

```
<template>
  <lightning-card title="HelloWorld" icon-name="custom:custom14">
    <div class="slds-m-around_medium">
      <p>Hello, {greeting}!</p>
      <lightning-input label="Name" value={greeting} onchange={changeHandler}></lightning-input>
    </div>
  </lightning-card>
</template>
```

9. Save the file.
10. In the **JavaScript file**, **helloWorld.js**, copy and paste the following code

```
import { LightningElement } from 'lwc';
export default class HelloWorld extends LightningElement {
        greeting = 'World';
        changeHandler(event) {
        this.greeting = event.target.value;        }    }
```

Lightning Web Components Basics - Create Lightning Web Components

11. Save the file.

12. In the **XML file** **helloWorld.js-meta.xml**, copy and paste the following code.

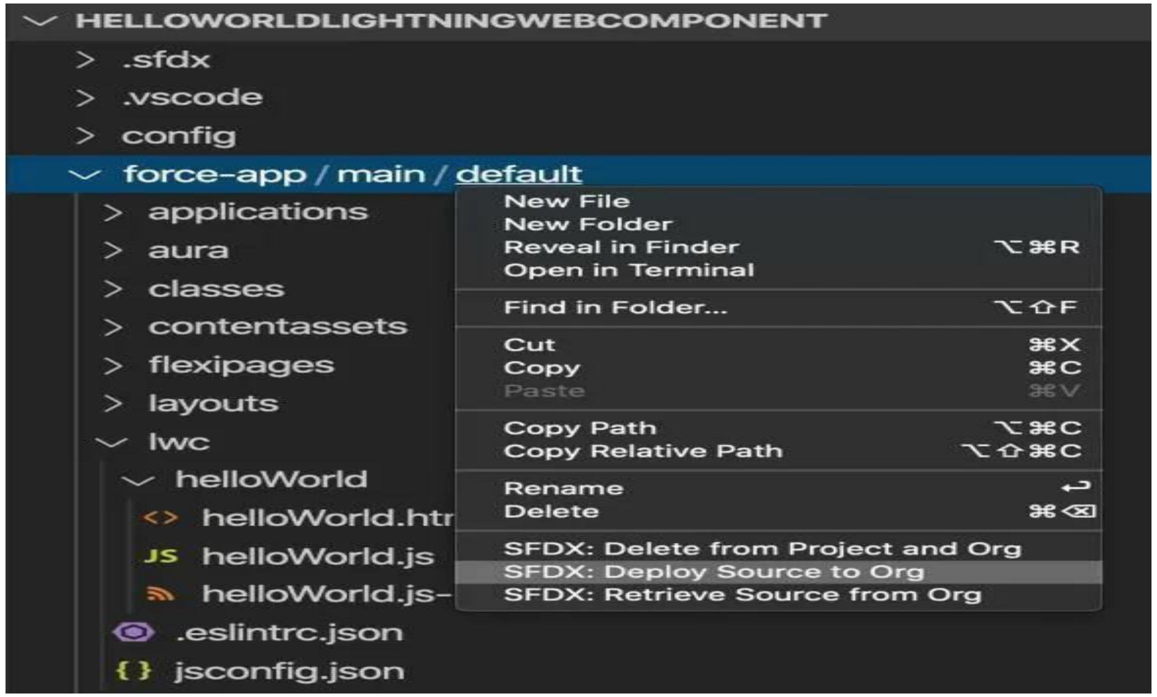
```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata" fqn="helloWorld">
   <apiVersion>58.0</apiVersion>
   <isExposed>true</isExposed>
   <targets>
   <target>lightning__AppPage</target>
   <target>lightning__RecordPage</target>
   <target>lightning__HomePage</target>
   </targets>
</LightningComponentBundle>
```

13. Save the file.

Lightning Web Components Basics - Deploy Lightning Web Component Files

Deploy to Your Org:

1. Right-click the default folder under force-app/main.



2. Click SFDX: Deploy Source to Org.
3. In the Output tab of the integrated terminal, view the results of your deployment. If the command ran successfully, a Deployed Source message lists the three files that were uploaded to the org.

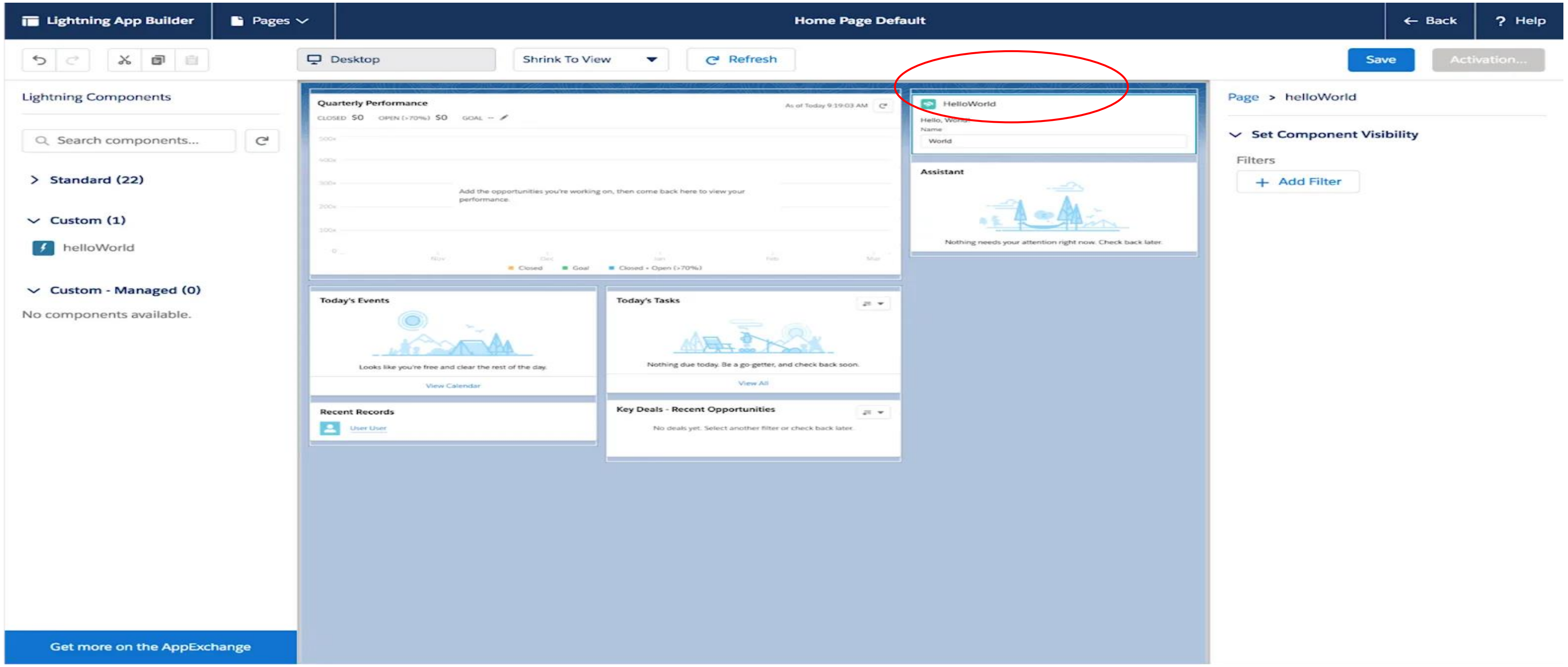
Lightning Web Components Basics - Deploy Lightning Web Component Files

Add Component to App in Lightning Experience:

1. In Visual Studio Code, open the Command Palette by pressing Ctrl+Shift+P.
2. Type SFDX.
3. Select SFDX: Open Default Org.

   This opens your Salesforce login in a separate browser.
4. Click then select Setup.
5. In Quick Find, enter Home, then select Home in the Feature Settings section.
6. For Advanced Seller Home, toggle the setting to Inactive.
7. From the App Launcher ( ), find and select Sales.
8. Click then select Edit Page.
9. Drag the **helloWorld** Lightning web component from the Custom area of the Lightning Components list to the top of the Page Canvas.

Lightning Web Components Basics - Deploy Lightning Web Component Files



Add the opportunities you're working on, then come back here to view your performance.

Nothing needs your attention right now. Check back later.

Looks like you're free and clear the rest of the day.

Nothing due today. Be a go-getter, and check back soon.

No deals yet. Select another filter or check back later.

10. Click Save, Activate, Assign as Org Default and Save.
11. Click Save again, then click ← to return to the page.
12. Refresh the page to view your new component.

Lightning Web Components Basics - Handle Events in Lightning Web Components

There are two ways to listen for an event:

- declaratively from the component’s HTML template, or
- programmatically using an imperative JavaScript API.

It’s better to listen from the HTML template since it reduces the amount of code you have to write.

To handle events, define methods in the component’s JavaScript class.

Lightning Web Components Basics - Handle Events in Lightning Web Components

Attach an Event Listener Declaratively:

Declare the listener in markup in the template of the owner component, in this example, c-parent.

```
<!-- parent.html -->
<template>
  <c-child onnotification={handleNotification}> </c-child>
</template>
```

Define the handler function, in this example handleNotification, in the c-parent JavaScript file.

```
import { LightningElement } from "lwc";                    // parent.js
export default class Parent extends LightningElement {
handleNotification() {     // Code runs when event is received    }   }
```

Lightning Web Components Basics - Handle Events in Lightning Web Components

Listen for Changes to Input Fields:

To listen for changes from an element in your template that accepts input, such as a text field (<input> or <lightning-input>), use the onchange event.

```
<!-- form.html -->
<template>
  <input type="text" value={myValue} onchange={handleChange} />
</template>
```

// form.js

```
import { LightningElement } from "lwc";
export default class Form extends LightningElement {
  myValue = "initial value";
  handleChange(evt) {
    console.log("Current value of the input: " + evt.target.value);  }}
```

In this example, the handleChange() method in the JavaScript file is invoked every time the value of the input changes.

Lightning Web Components Basics - Handle Events in Lightning Web Components

For Example **Adding of Two Numbers**

addNumbers.html

```
<template>
    <lightning-card title="Add Two Numbers" icon name="custom:custom63">
<div>
     <lightning-input type="number" label="First Number" value={firstNumber} onchange={handleFirstNumberChange}>
    </lightning-input>
   <lightning-input type="number" label="Second Number" value={secondNumber} onchange={handleSecondNumberChange}>
     </lightning-input>
      <lightning-button label="Add" onclick={addNumbers}></lightning-button>
     <div>
         <p>Result: {result}</p>
      </div>
 </div>
    </lightning-card>
</template>
```

Lightning Web Components Basics - Handle Events in Lightning Web Components

For Example **Adding of Two Numbers**

addNumbers.js

```
import { LightningElement, track } from 'lwc';
export default class AddNumbers extends LightningElement {
   @track firstNumber = 0;
   @track secondNumber = 0;
   @track result;

   handleFirstNumberChange(event) {
      this.firstNumber = parseFloat(event.target.value);
   }
   handleSecondNumberChange(event) {
      this.secondNumber = parseFloat(event.target.value);
   }
   addNumbers() {
      this.result = this.firstNumber + this.secondNumber;
   }
}
```

addNumbers.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle
xmlns="http://soap.sforce.com/2006/04/metadata"
fqn="addNumbers">
   <apiVersion>57.0</apiVersion>
   <isExposed>true</isExposed>

   <targets>
      <target>lightning__AppPage</target>
      <target>lightning__RecordPage</target>
      <target>lightning__HomePage</target>
   </targets>

</LightningComponentBundle>
```

Lightning Web Components Basics - Add Styles and Data to a Lightning Web Component

Lightning Design System Styles(LDSS) - CSS and Component Styling:

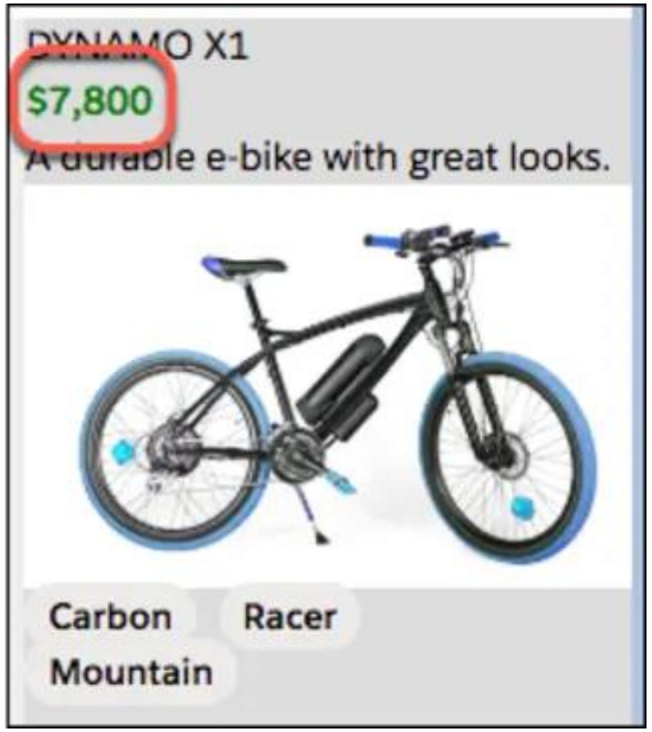
- The implementation of CSS for Lightning Web Components adheres to the W3C standard. You can create a style sheet in the CSS file, and it's automatically applied to the corresponding HTML file.
- Lightning Web Components encapsulates components, keeping them separate from the global DOM. We do this through a mechanism called Shadow DOM.
- Shadow DOM is a common implementation that allows a component's elements to live in a "sub tree" of the DOM.
- The component can keep its appearance and behavior within other apps or as a child of another component.

Lightning Web Components Basics - Add Styles and Data to a Lightning Web Component

For example:

- Add the following .price entry to the **detail.css** file.

```
body{
  margin: 0;
}
.price{
  color: green;
  font-weight: bold;
font-family: serif;
}
```



DYNAMO X1

$7,800

A durable e-bike with great looks.

- Save and deploy your files.

Secure Server-Side Development - Write Secure Apex Controllers

Apex Security and Sharing:

When you use Apex, the security of your code is critical. By default, Apex classes code executes in system mode and has the ability to read and update all data within an organization. Therefore, you must enforce sharing rules, set object and field permissions, and protect against CRUD and FLS violations.

Two ways to provide security

1. Enforcing Sharing Rules
2. Enforcing Object and Field Permissions

- You have to explicitly set this keyword for the class because Apex code runs in system context

Secure Server-Side Development - Write Secure Apex Controllers

1. Enforcing Sharing Rules:

- Apex generally runs in system context.
- In system context, Apex code has access to all objects and fields—object permissions, field-level security, and sharing rules aren’t applied for the current user.

There are three keywords to remember for sharing rules,

- With Sharing
- Without sharing
- Inherited sharing

Secure Server-Side Development - Write Secure Apex Controllers

With Sharing:

- The with sharing keyword lets you specify that the sharing rules for the current user are enforced for the class.
- You have to explicitly set this keyword for the class because Apex code runs in system context.

Without Sharing

- You use the without sharing keywords when declaring a class to ensure that the sharing rules for the current user are not enforced.
- For example, you can explicitly turn off sharing rule enforcement when a class is called from another class that is declared using with sharing

Secure Server-Side Development - Write Secure Apex Controllers

Inherited Sharing

- Apex class without a sharing declaration is insecure by default.
- Designing Apex classes that can run in either with sharing or without sharing mode at runtime is an advanced technique.
- Such a technique can be difficult to distinguish from one where a specific sharing declaration is accidentally omitted.
- An explicit inherited sharing declaration clarifies the intent, avoiding ambiguity arising from an omitted declaration or false positives from security analysis tooling.
- An Apex class withinherited sharing runs as with sharing when used as:
  - An Aura component controller
  - A Visualforce controller
  - An Apex REST service
  - Any other entry point to an Apex transaction

Secure Server-Side Development - Write Secure Apex Controllers

2. Enforcing Object and Field Permissions

User Mode Operations

Data operations (SOQL, DML, and SOSL) in Apex run in system mode by default and have full CRUD access to all objects and fields in general.

In Spring 2023, Apex introduced new access levels allowing developers to select the mode for executing data operations.

- User mode
- System mode

Access Records in User Mode:

Access Records in User mode ensures the enforcement of sharing rules, CRUD/FLS, and Restriction Rules.

By utilizing SOQL queries with the USER_MODE keyword, such as in this example

List<Account> acc = [SELECT Id FROM Account WITH USER_MODE];

System mode privileges are temporarily dropped, ensuring retrieval of only the records accessible to the user.

Secure Server-Side Development - Write Secure Apex Controllers

Insert Records in User Mode:

In User mode, Insert Records ensures that the insert operation executes only if the user has permission for both creating a new record and edit permission on the field Opportunity.Amount (FLS check).

```
Opportunity o = new Opportunity();
// specify other fields
o.Amount=500;
insert as user o;
```

Another way to execute User mode operations:

```
Opportunity o = new Opportunity();
// specify other fields
o.Amount=500;
database.insert(o,AccessLevel.USER_MODE);
```

Secure Server-Side Development - Write Secure Apex Controllers

Update Records in User Mode

To update Records in User mode:

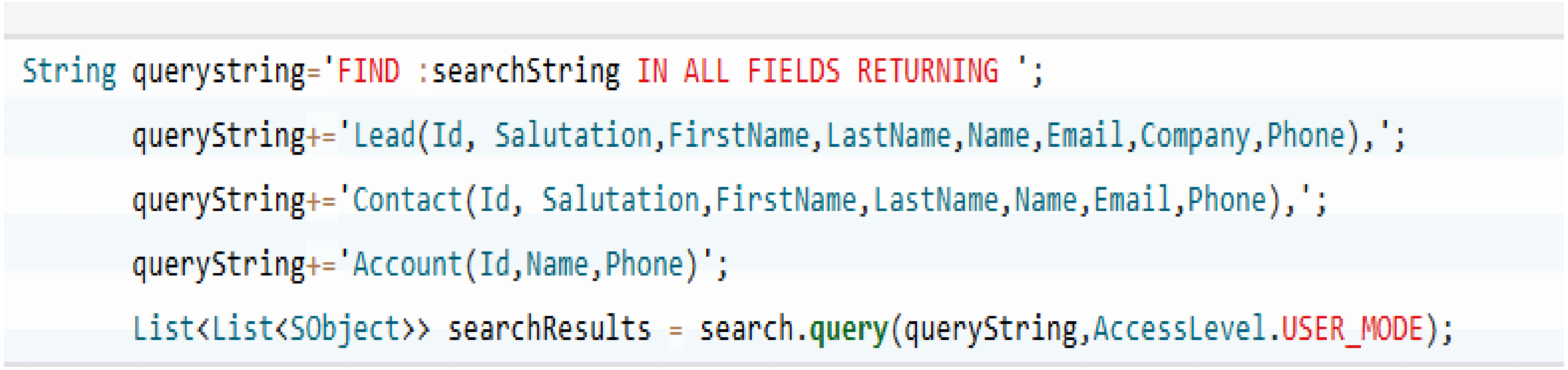
Account a = [SELECT Id,Name,Website FROM Account WHERE Id=:recordId];

a.Website='https://example.com';

update as user a;

SOSL in User Mode

To execute SOSL in User mode:



```
String querystring='FIND :searchString IN ALL FIELDS RETURNING ';
       queryString+='Lead(Id, Salutation,FirstName,LastName,Name,Email,Company,Phone),';
       queryString+='Contact(Id, Salutation,FirstName,LastName,Name,Email,Phone),';
       queryString+='Account(Id,Name,Phone)';
       List<List<SObject>> searchResults = search.query(queryString,AccessLevel.USER_MODE);
```

Secure Server-Side Development - Write Secure Apex Controllers

Using WITH SECURITY_ENFORCED:

You can integrate the **WITH SECURITY_ENFORCED clause into your SOQL SELECT queries** within Apex code to validate field- and object-level security permissions automatically.

Here's how to use **WITH SECURITY_ENFORCED,**

Strategic Placement:

1. **Insert the clause after the WHERE clause** (if present) or after the FROM clause if no WHERE clause exists.

2. **Place it before ORDER BY, LIMIT, OFFSET, or aggregate function** clauses.

Example:

List<Account> act1 = [SELECT Id, (SELECT LastName FROM Contacts) FROM Account **WHERE Name like 'Acme' WITH SECURITY_ENFORCED]**

This returns Id and LastName for the Acme account entry if the user has field access for LastName.

Secure Server-Side Development - Write Secure Apex Controllers

Using CRUD/FLS Check Methods

You can also enforce object-level and field-level permissions in your code by explicitly calling the **sObject describe result methods** and the **field describe result methods** that check the current user's access permission levels.

- **Schema.DescribeSObjectResult**
- **Schema.DescribeFieldResult**

For example, you can call the isAccessible, isCreateable, or isUpdateable methods of **Schema.DescribeSObjectResult** to verify whether the current user has read, create, or update access to an sObject, respectively. Similarly, **Schema.DescribeFieldResult** exposes these access control methods that you can call to check the current user’s read, create, or update access for a field. In addition, you can call the **isDeletable()** method provided by **Schema.DescribeSObjectResult** to check if the current user has permission to delete a specific sObject.

Secure Server-Side Development - Write Secure Apex Controllers

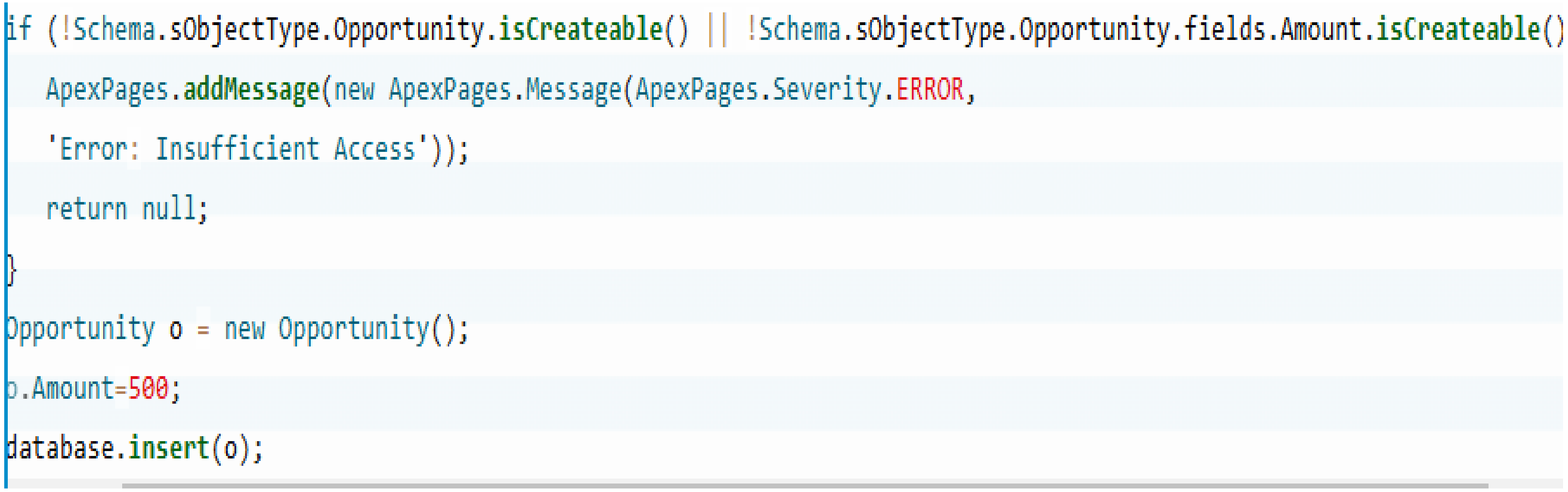
Let’s walk through the **DescribeSObjectResult** class **helper functions** that you can use to verify a user’s level of access. These include:

- **IsCreateable()**
- **IsAccessible()**
- **IsUpdateable()**
- **IsDeleteable()**

Secure Server-Side Development - Write Secure Apex Controllers

IsCreateable():

Before your code inserts a record in the database, you have to check that the logged-in user has both Edit permission on the field and Create permission on the object. You can check both permissions by using the **isCreateable()** method on the particular object.

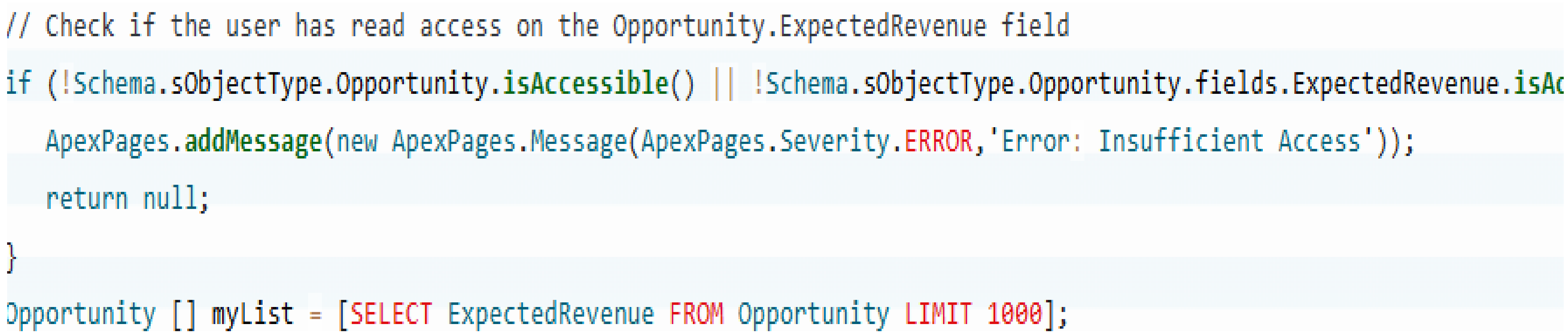


```
if (!Schema.sObjectType.Opportunity.isCreateable() || !Schema.sObjectType.Opportunity.fields.Amount.isCreateable()
   ApexPages.addMessage(new ApexPages.Message(ApexPages.Severity.ERROR,
   'Error: Insufficient Access'));
   return null;
}
Opportunity o = new Opportunity();
o.Amount=500;
database.insert(o);
```

Secure Server-Side Development - Write Secure Apex Controllers

isAccessible():

Before your code retrieves a field from an object, you want to verify that the logged-in user has permission to access the field by using the isAccessible() method on the particular object.

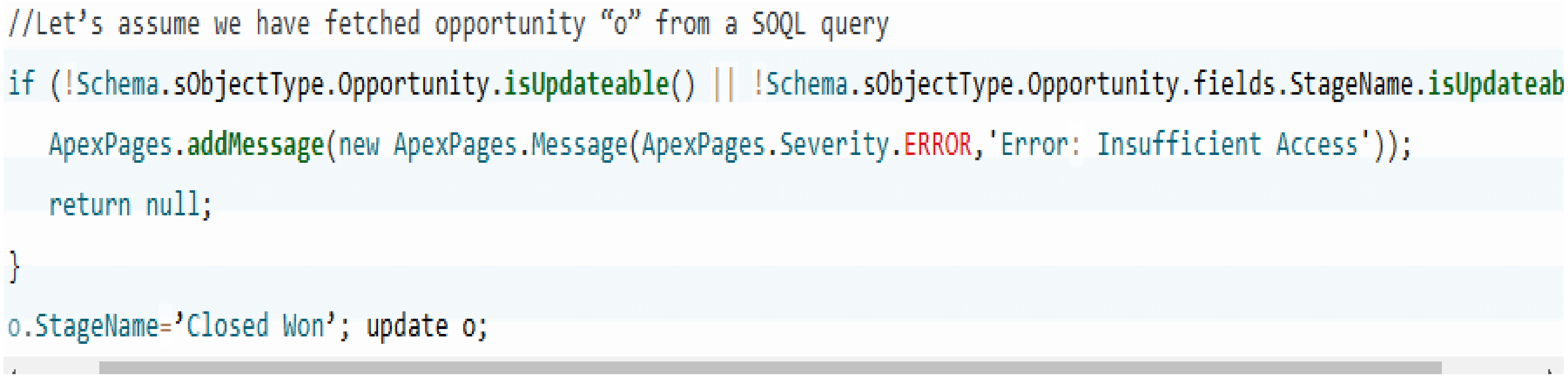


```
// Check if the user has read access on the Opportunity.ExpectedRevenue field
if (!Schema.sObjectType.Opportunity.isAccessible() || !Schema.sObjectType.Opportunity.fields.ExpectedRevenue.isAc
   ApexPages.addMessage(new ApexPages.Message(ApexPages.Severity.ERROR,'Error: Insufficient Access'));
   return null;
}
Opportunity [] myList = [SELECT ExpectedRevenue FROM Opportunity LIMIT 1000];
```

Secure Server-Side Development - Write Secure Apex Controllers

isUpdateable():

Similarly, before your code updates a record, you have to check if the logged-in user has Edit permission for the field and the object. You can check for both permissions by using theisUpdateable()method on the particular object.



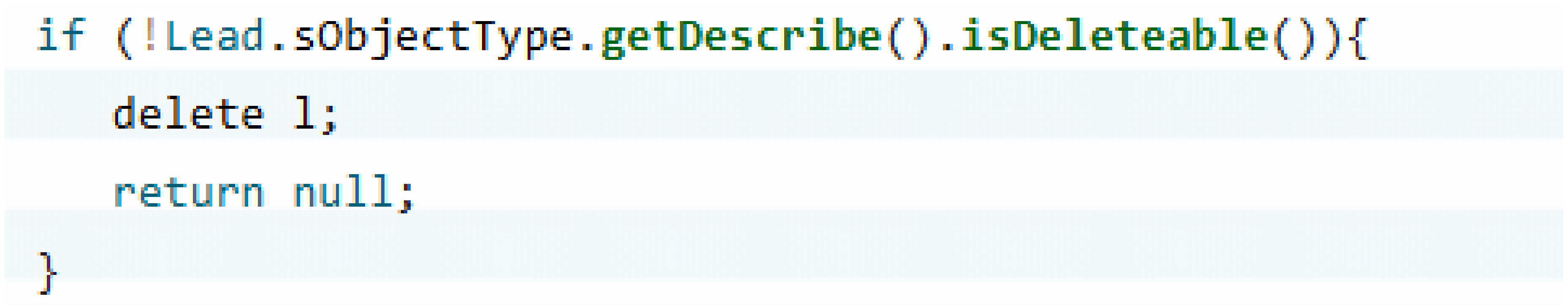
```
//Let’s assume we have fetched opportunity “o” from a SOQL query
if (!Schema.sObjectType.Opportunity.isUpdateable() || !Schema.sObjectType.Opportunity.fields.StageName.isUpdateab
   ApexPages.addMessage(new ApexPages.Message(ApexPages.Severity.ERROR,'Error: Insufficient Access'));
   return null;
}
o.StageName=’Closed Won’; update o;
```

Secure Server-Side Development - Write Secure Apex Controllers

isDeleteable():

Lastly, to enforce “delete” access restrictions, use the isDeleteable() function before your code performs a delete database operation.

Here’s how to configure this operation,



```
if (!Lead.sObjectType.getDescribe().isDeleteable()){
    delete l;
    return null;
}
```

Secure Server-Side Development - Mitigate SOQL Injection

Is SOQL Vulnerable to Injection Attacks?

A developer can still trust user input incorrectly, leading to an exposure of information via what is referred to as a SOQL injection attack.

SOQL Injection Prevention

1. Static queries with bind variables
2. String.escapeSingleQuotes()
3. Type casting
4. Replacing characters
5. Allowlisting

Secure Server-Side Development - Mitigate SOQL Injection

1. Static Query and Bind Variables

The first and most recommended method to prevent SOQL injection is to use static queries with bind variables. Consider the following query.

```
String query = ‘select id from contact where firstname =\”+var+’\”;
queryResult = Database.execute(query);
```

Using user input (the var variable) directly in a SOQL query opens the application up to SOQL injection. To mitigate the risk, translate the query into a static query like this:

```
queryResult = [select id from contact where firstname =:var]
```

Secure Server-Side Development - Mitigate SOQL Injection

2. Typecasting

Another strategy to prevent SOQL injection is to use typecasting.

By casting all variables as strings, user input can drift outside of expectation.

By typecasting variables as integers or Booleans, when applicable, erroneous user input is not permitted

The variable can then be transformed back to a string for insertion into the query using **string.valueOf()** method

```
public String textualAge {get; set;}
[...]
whereClause+='Age__c >'+textualAge+'';
whereclause_records = database.query(query+' where '+whereClause);
```

After Typecasting

```
whereClause+='Age__c >'+string.valueOf(textualAge)+'';
```

Secure Server-Side Development - Mitigate SOQL Injection

3. Escaping Single Quotes

Another cross-site scripting (XSS) mitigation option that is commonly used by developers who include user-controlled strings in their queries is the platform-provided escape function **string.escapeSingleQuotes()**.

```
String query = 'SELECT Id, Name, Title__c FROM Books';

String whereClause = 'Title__c like \'%'+textualTitle+'%\' ';

List<Bookswhereclause_records = database.query(query+' where '+whereClause);
```

In the example, replacing the where clause with the following code wrapping **textualTitle with String.escapeSingleQuotes()** will prevent an attacker from using SOQL injection to modify the query behavior.

```
String whereClause = 'Title__c like \'%'+String.escapeSingleQuotes(textualTitle)+ '%\' ';
```

Secure Server-Side Development - Mitigate SOQL Injection

4. Replacing Characters

A final tool in your tool belt is character replacement, also known as blocklisting.

This approach **removes "bad characters" from user input**.

```
String query = 'select id from user where isActive='+var;
```

Example:

before replacing

```
true AND ReceivesAdminInfoEmails=true
```

after replacing

```
trueANDRecievesAdminInfoEmails=true
```

The code to remove all spaces from a string can be written as follows:

```
String query = 'select id from user where isActive='+var.replaceAll('[^\\w]','');
```

Secure Server-Side Development - Mitigate SOQL Injection

5. Allowlisting

Another way to prevent SOQL injection without string.escapeSingleQuotes() is allowlisting.

Create a list of all “known good” values that the user is allowed to supply.

If the user enters anything else, you reject the response

Secure Server-Side Development - Request Forgery

Mitigate Cross-Site Request Forgery:

What Is CSRF?

CSRF is a common web application vulnerability where a malicious application causes a user’s client to perform an unwanted action on a trusted site for which the user is currently authenticated.

Let’s start with the idea that we have built an application that lists all of the current students in our network of schools. In this application, there are two important things to note.

Only the admin or the superintendent can access the page allowing users to promote students to the honor roll.

The page automatically refreshes if you click the Honor Roll link. If you’ve added a student, an alert will be noted that your student has been added to the honor roll.

Secure Server-Side Development - Request Forgery

Mitigate Cross-Site Request Forgery:

What Is CSRF?

What is happening behind the scenes is that the Honor Roll button makes a GET request to /**promote?UserId=<userid>**

As the page loads, it reads the URL parameter value and automatically changes the role of that student to the honor roll.

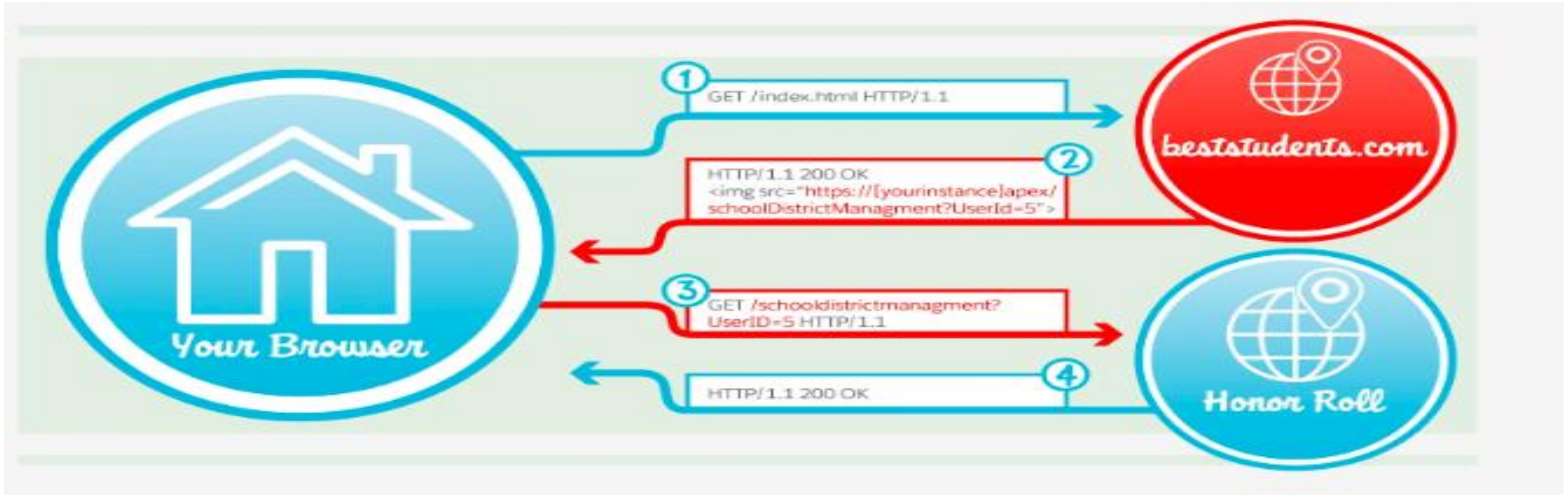
Secure Server-Side Development - Request Forgery

CSRF- Attack

This time, imagine that after logging in to your School District Management org, you decided to browse another website.

While on this website, you click a hyperlink. This hyperlink redirects to a link to www.beststudents.com/promote?user_id=123.

This malicious link is executed on behalf of the admin (your signed-in account), thereby promoting a student to the honor roll without you realizing it.



```
GET /index.html HTTP/1.1
```

```
HTTP/1.1 200 OK
<img src="https://[yourinstance]apex/
schoolDistrictManagment?UserId=5">
```

```
GET /schooldistrictmanagment?
UserID=5 HTTP/1.1
```

```
HTTP/1.1 200 OK
```

Secure Server-Side Development - Request Forgery

Prevent CSRF Attacks:

Consider a slightly different version of the page that has **two required URL parameters: userId and token**.

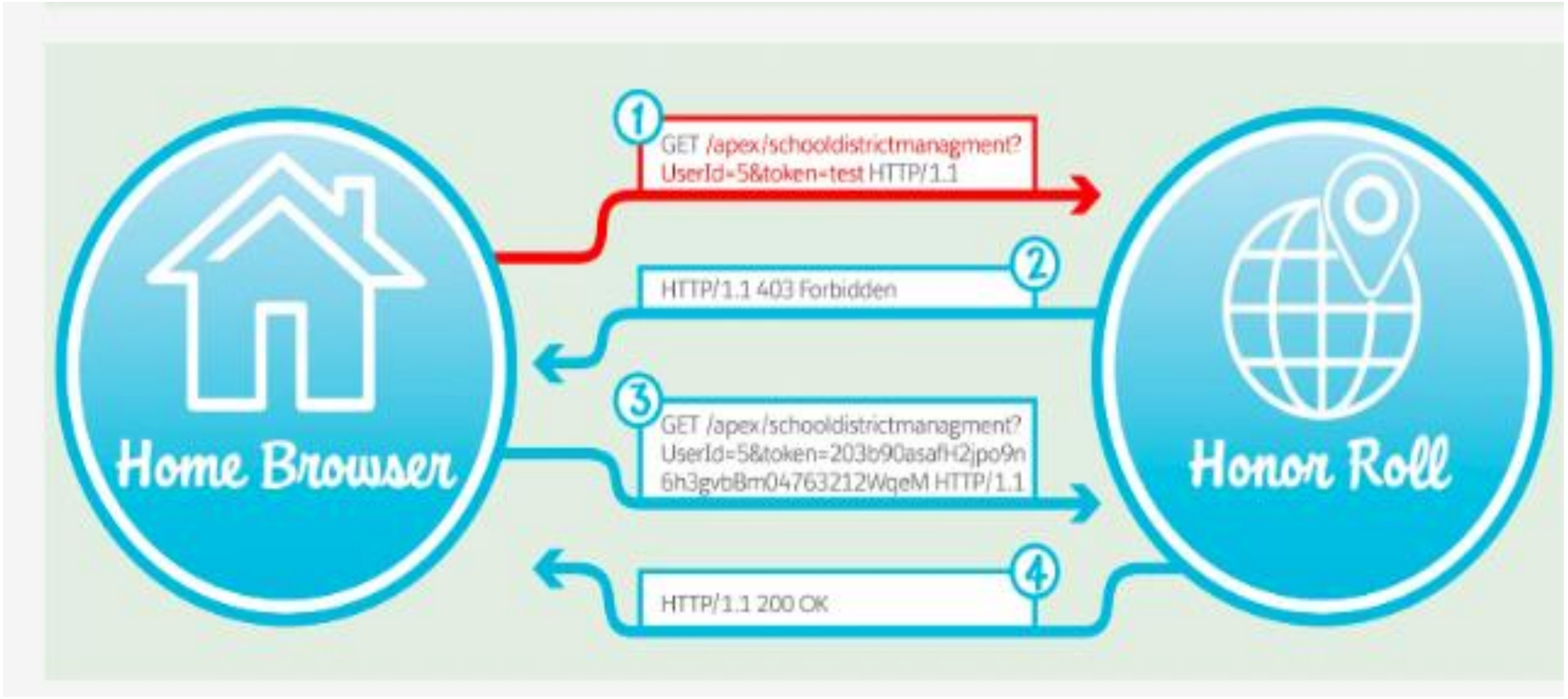
What if you made **the token parameter value a random, unique value that changed on every request**?

This would make it next to **impossible for an attacker to guess the current value, preventing the attack**.

This example is the most common prevention technique for CSRF.

Secure Server-Side Development - Request Forgery

Prevent CSRF Attacks:



```
GET /apex/schooldistrictmanagment?
UserId=5&token=test HTTP/1.1
```

```
HTTP/1.1 403 Forbidden
```

```
GET /apex/schooldistrictmanagment?
UserId=5&token=203b90asafH2jpo9n
6h3gvbBm04763212WqeM HTTP/1.1
```

```
HTTP/1.1 200 OK
```

Secure Server-Side Development - Request Forgery

For this prevention technique to be successful, four things must happen.

All sensitive state-changing requests (anything performing database operations) must include a token.

A token must be unique to the request or user’s session.

A token must be difficult to predict (long with advanced encryption).

The server must validate a token to ensure the request originated from the intended user

Secure Server-Side Development - Request Forgery

Mitigate Server Side Request Forgery:

What Is Server Side Request Forgery?

Server-side request forgery (SSRF) is a security vulnerability in web applications where an attacker can make unauthorized requests, both internal and external, on behalf of the server.

In an SSRF attack, the malicious application tricks the server into making requests to internal and external services or systems, potentially leading to unauthorized access or data exposure.

Secure Server-Side Development - Request Forgery

Mitigate Server Side Request Forgery:

The application is designed to make a GET request to the server whose address is contained in the API request to retrieve the student's details, for example

studentApi=https://192.168.0.1/student

An attacker would intercept the API call from the client, replace the endpoint value of the student service with a call to the metadata service and exfiltrate sensitive service configuration data from the internal metadata endpoint.

Preventing SSRF Attacks

Validate and Sanitize Inputs

Ensure that input values, such as the studentApivalue in our example, are properly validated and sanitized to prevent the injection of malicious URLs.

Implement Allowlisting

Restrict the allowed destinations for outgoing requests by enforcing the URL schema, port, and destination allowlist, disabling HTTP redirections. Only allow requests to specified, trusted endpoints.

Use URL Parsing Libraries

Utilize URL parsing libraries to parse and validate URLs before making requests. This helps ensure that the requested URLs conform to expected patterns.

Network Segmentation

Implement network segmentation to restrict the server's ability to make requests to internal resources, limiting the impact of any potential SSRF attacks.

Salesforce Platform Protections Against SSRF

Avoid GET Requests

Similar to CSRF prevention, developers should avoid using HTTP GET requests. Instead, prefer using POST or PUT requests to minimize risk of SSRF data exfiltration.

Validate Origin Headers

When integrating Salesforce Lightning applications with third-party APIs, validate the origin header in HTTP requests. Ensure that the request originates from a trusted source to prevent potential SSRF exploits.

Implement Anti-SSRF Tokens

Developers can add custom anti-SSRF tokens to XMLHttpRequests within Lightning by using setRequestHeader(). This adds an additional layer of protection against SSRF attacks