* Lightning data service
  + How we use salesforce data with out LWC
  + Have to deploy to org to actually get records since server will not help you
  + There are several ways LWC can work with data
    - Lightning data service (LDS)
      * Easiest way to work with salesforce data in LWC
      * LWC have lightning components build on top of LDS
      * Records loaded this way are cached and shared across components if a page is composed of several components
        + This means they all share the same version
        + This increases the performance because the record only needs to be loaded in once
      * To perform operations on data and access metadata using LDS we can use
        + Components: lightning-record-edit-form, lightning-record-form, and lightning-record-view-form
        + Wire adapters and function in the lightning/UI API modules
      * If LDS detects a record to a record or any data or metadata it supports all components using a relevant @wire adapter receive the new value
      * If you need more flexibility you can use a LDS wire adapter directly
      * Build in LDS components
        + Provide the UI to view, create and edit a record
      * LDS Wire Adapters and Functions
        + To access raw record data so that you can perform business logic or create a form that needs more customization than the base components
        + Use @wire to specify the getRecord LDS wire adaptor

The wire service provides an immutable stream of data to the component

Each value in the stream is a newer version of the value that precedes it

We call the wire service reactive in part since it supports reactive variables which are prefixed with $

If a reactive variable changes, the wire service provisions new data

* + - * + Wire service syntax

Import a wire adapter using names import syntax

Decorate a property of function with the @wire decorator

Specify the record adaptor

Each wire adaptor defines a data type

Import {adapterId} from ‘adapterModule’

@wire(adaptorId, adapterConfig)

Property or function

adapterId (identifier) is the id of the wire adaptor

adapterModule (string) is the id of the module that contains the wire adapter function

adaptorConfig(object) is a configuration object specific to the wire adaptor

Config object property values can either be strings or references to objects and fields imported from @salesforce/schema

Properties in the adapterConfig object cant be undefined

propertyorFunction is a private property of function that receives the stream of data from the wire service

to access object and field API names, use an import statement

All objects and field imports come from @salesforce/schema

Import objectName from ‘@salesforce/schema/object’

Import objectName from ‘@salesforce/schemanamespace\_\_/object’

Import POSITION\_OBJECT from ‘@salesforce/schema/Position\_\_c’

Import FIELD\_NAME from ‘@salesforce/schema/object.field’

To display a lust of filed values you can use getListUI

* + - * + Import limitations

Salesforce supports many object or fold suffixes to represent different types of data

* Calling apex methods from LDS
  + Caking apex methods with wiring
    - Apex limits are applied per invocation of an apex method
    - Use default import statements to import apex methods
    - Import apexMethodName from ‘@salesforce/apex/namespace.className.methodReference’;
    - The apex method must be annotated with @AuraEnabled(cacheable=true)
    - The method must also be static and public/global
    - @wire(apexMethodName, [apexMethodParams])
    - propertyOrFunction
  + Calling apex methods imperatively
    - To control when the method invocation occurs
    - You do this to call a method without the cacheable annotation
      * Anything that inserts, updates or deletes data
    - To work with objects not supported buy UI API
    - To call a method from an ES6 module that doesn’t extend LightningElement
    - The imported function returns a promise
      * This code provides a one time resolution given a set of parameters
* Events in LWC
  + LWC events are dispatched to standard DOM events
  + Can also create and dispatch custom events
  + These events are used to communicate up the component hierarchy
  + Are actually built on top of DOM events
  + DOM event system programming design pattern
    - Event name, called type
    - Config to initialize the event
    - A JS object that emits the event
  + Custom events
    - To create, we use the CustomEvent interface
    - In LWC customEvent provides a more consistent experience across browsers
    - It requires a no setup or boiler plate code
    - Can pass any kind of data using the detail property
    - To create, use the CustomEvent() constructor
      * It requires one parameter which is indicating the event type
    - As a component author, you name the event type when you create the event
    - Whatever you name your event type, the word on will be prefixed to the string you provide
    - Then you can trigger those custom events and handle them with logic that is needed
    - You can still attach event listeners programmatically with this.template.addEventListener(‘eventType’,function)
* Lightning message service
  + A way to communicate between components not in the same hierarchy
  + Allows for communication across the lightning message service
  + Can talk to about anything in the org
  + Any component in a lightning experience application that listens for events in a message channel updates when it receives a message
  + It also works across namespaces
  + To create a message channel
    - Use the LightningMessageChannel metadata type
    - Deploy it to your org
    - Include the xm definition in the firce-app/main/default/messageChannels directory
    - messageChannelName.messageChannel-meta.xml