### College Management Application

### Salesforce Project Ready

**Day 1 :**

### **Topic:** Setting Up Salesforce Developer Org To Get Started

### **Milestone / Activities**: Created Developer Account

**Detailed Description**: Steps Followed :-

Creating a developer org in salesforce.

1.Go to developers.salesforce.com

2.Click on sign up

3.On the sign up form, enter the following details :

a.First name & Last name

b.Email

c.Role : Developer

d.Company : College Name

e.Country : India

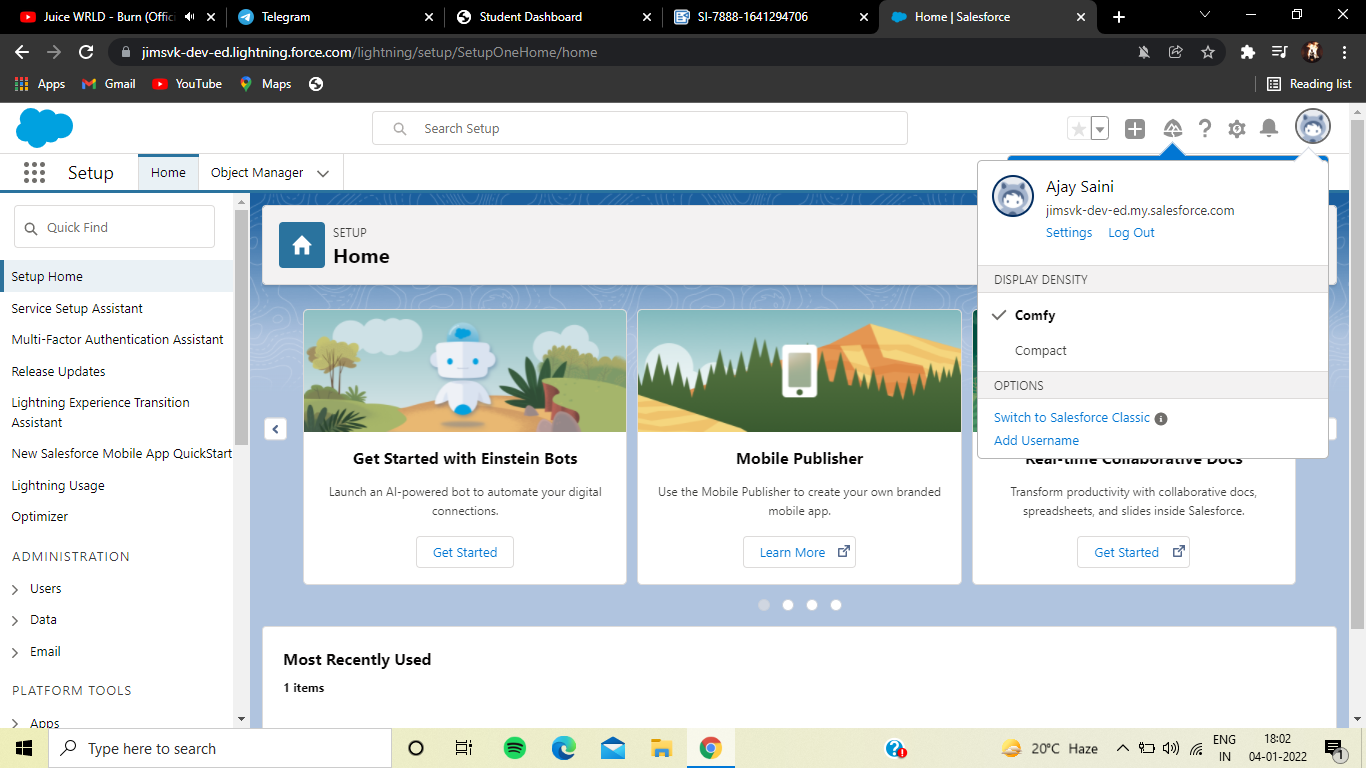
f.Postal Code : pin code

g.Username : should be a combination of your name and company

This need not be an actual email id, you can give anything in the format :

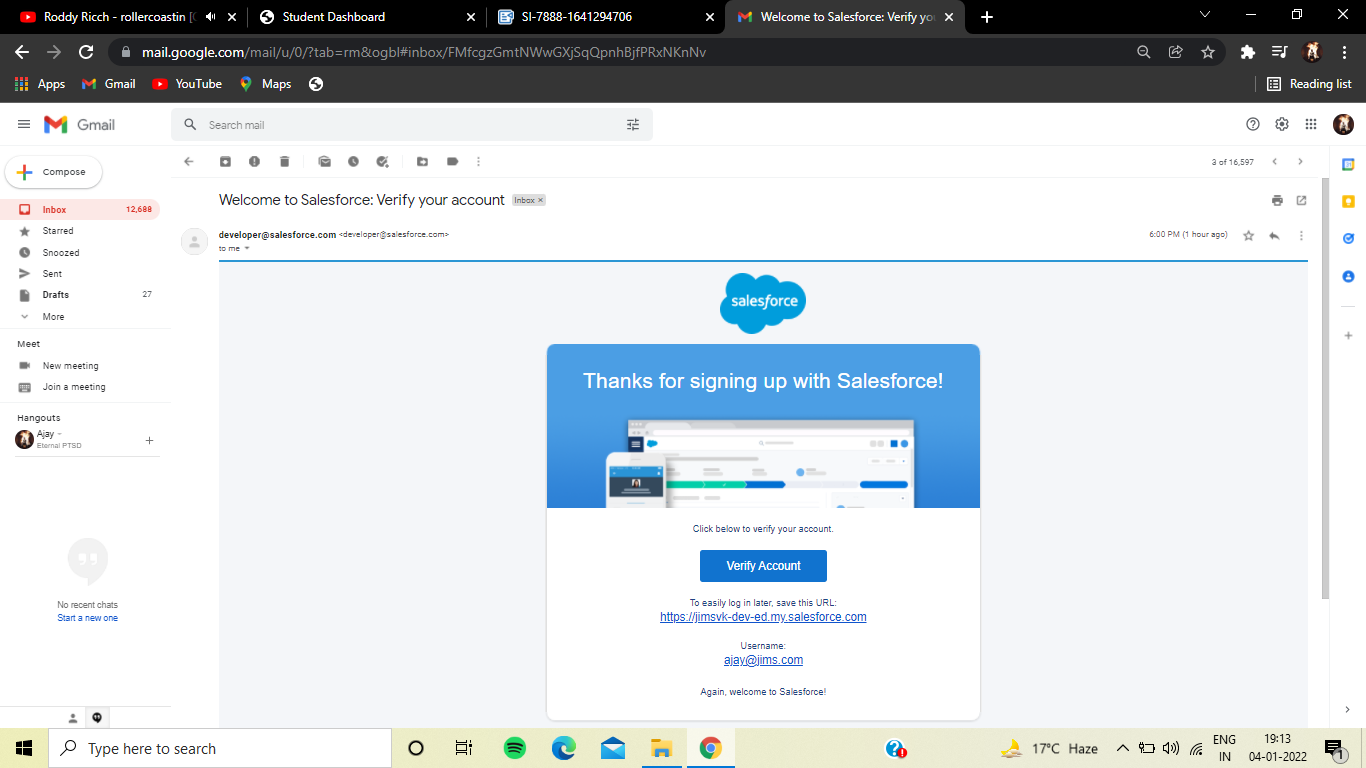
username@organization.com

Click on sign up



### **Account Activation**

### Activation Email

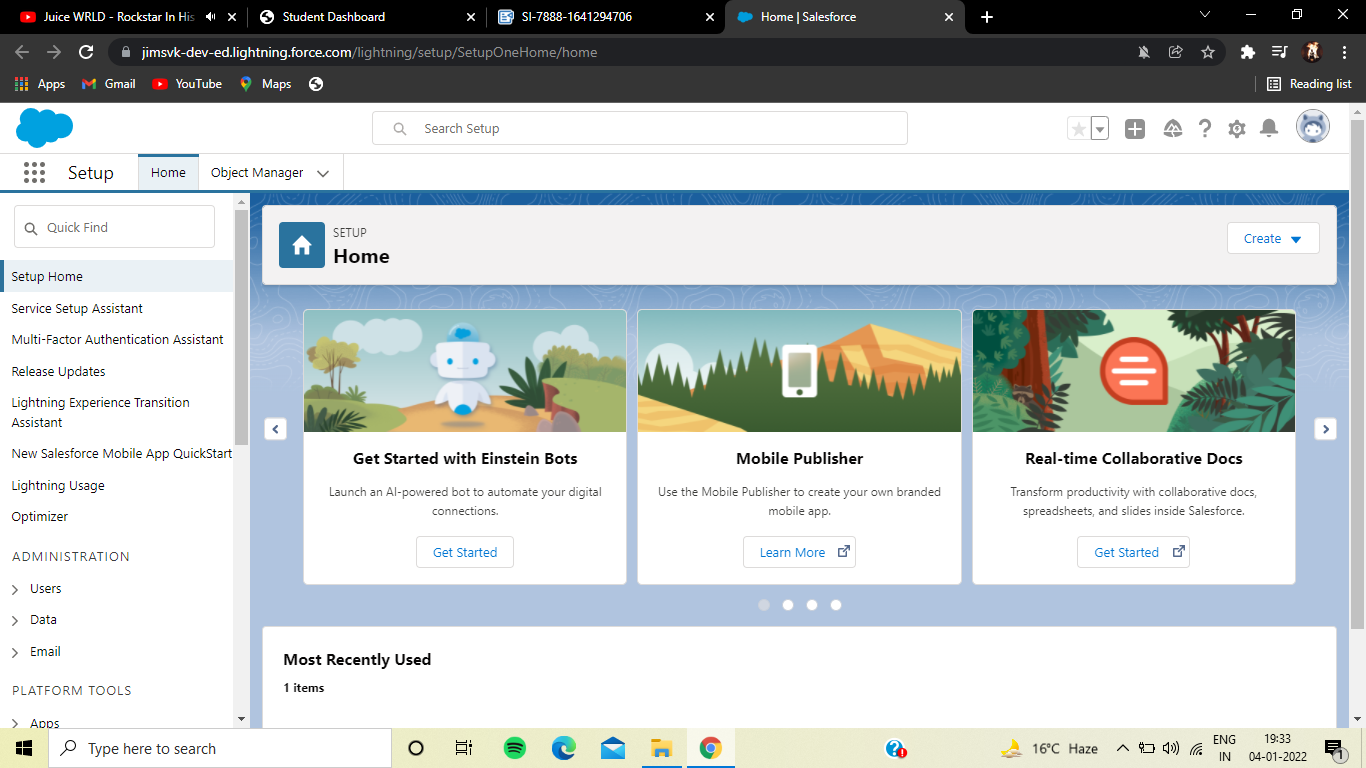


### **Login Into Salesforce Account**

1.Go to salesforce.com and click on login.

2.Enter the username and password.

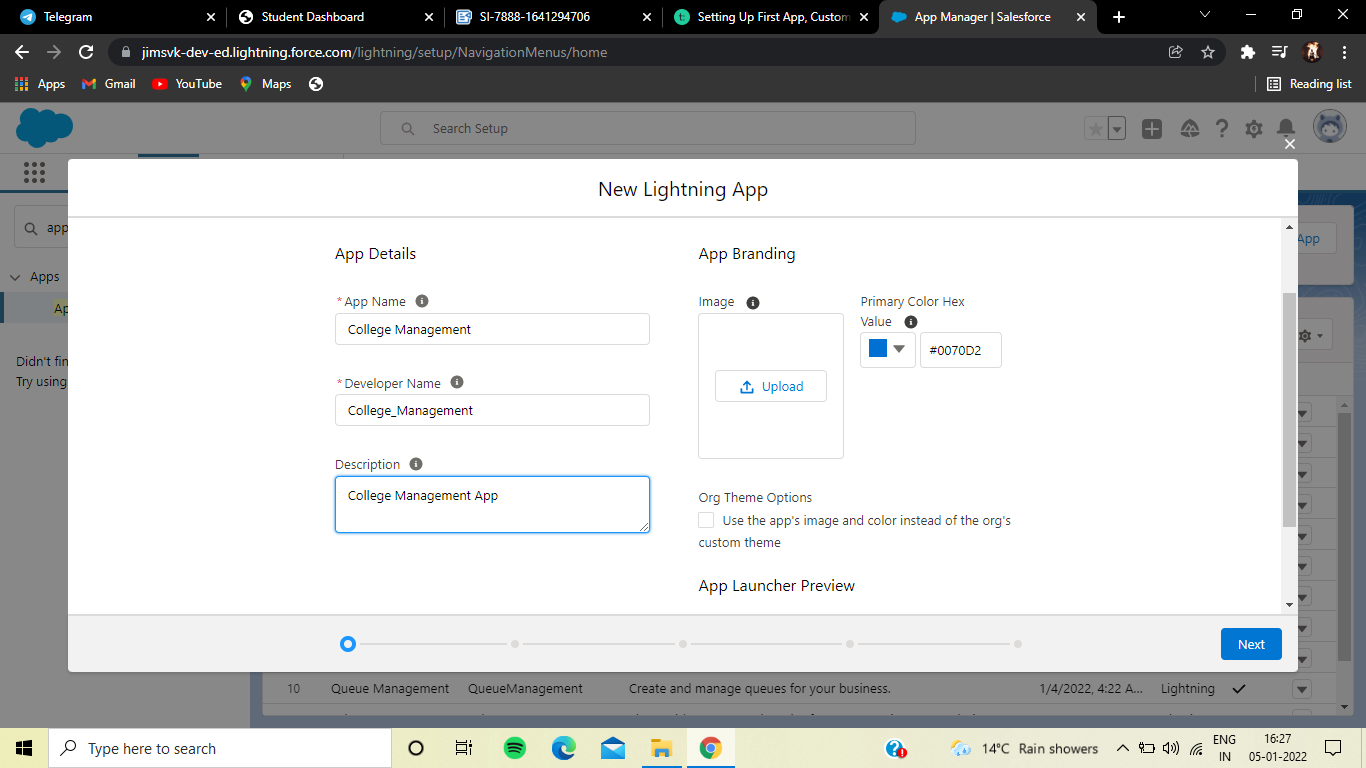
3.After login you'll see the home page.



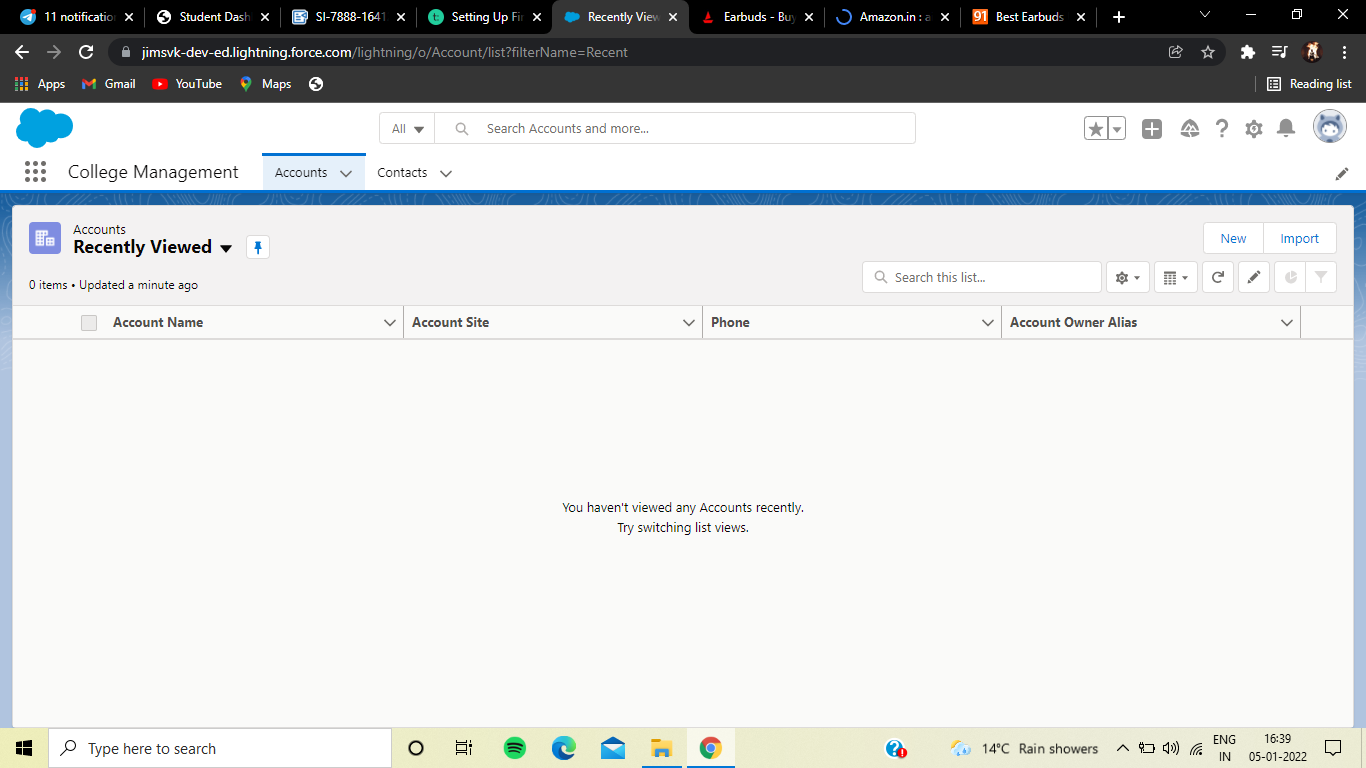
**Day 2 :**

## Topic: Setting Up First App, Custom Object and Fields Creation

First step is creating new lightning app.



this is how our app gonna look alike

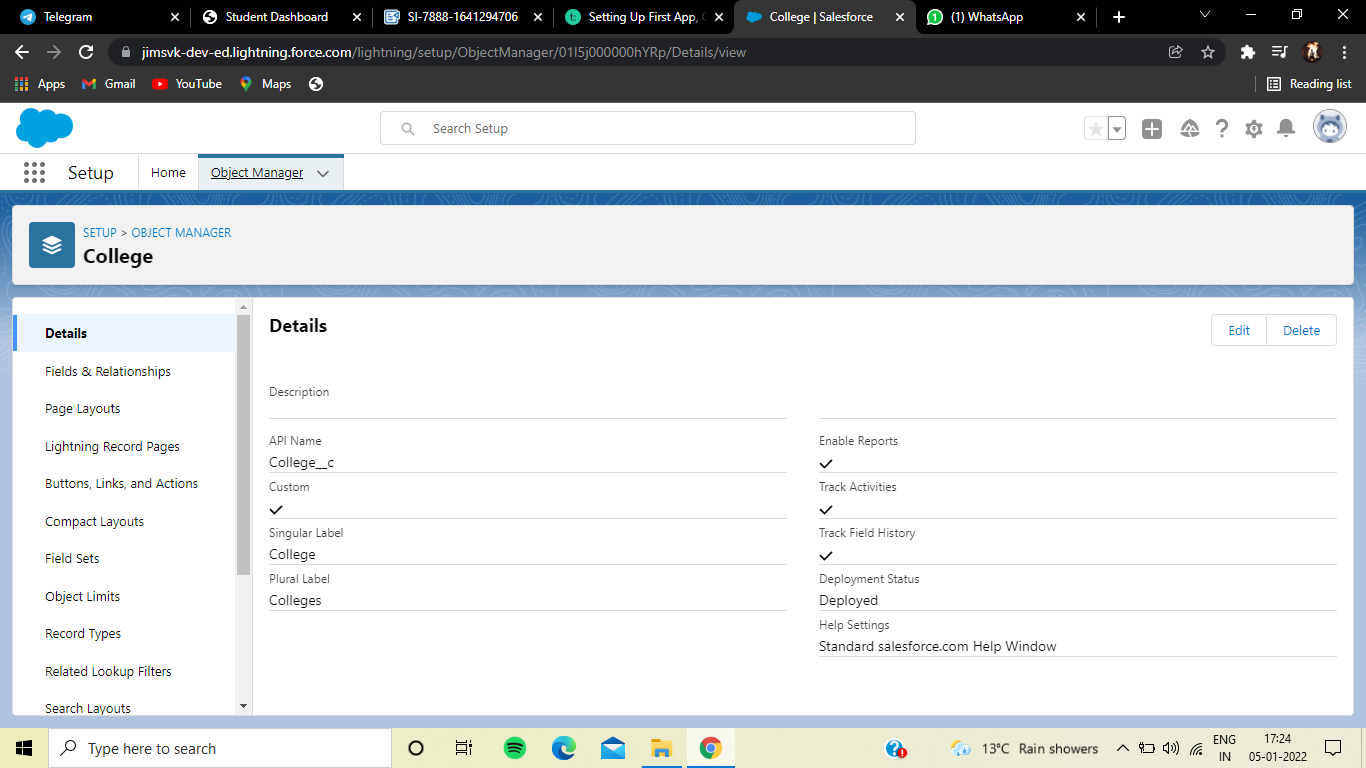


### **Milestone:** Creating The Custom Objects For The Application

**Custom objects are the objects which are created manually by the users**

### Custom objects created :

**1. College**



### Creating Fields On College Object

Field Name Data Type Required Values

College Name Picklist Yes

College Email Picklist blr@mit.co.in

hyd@mit.co.in

mum@mit.co.in

maa@mit.co.in

ccu@mit.co.in

del@mit.co.in

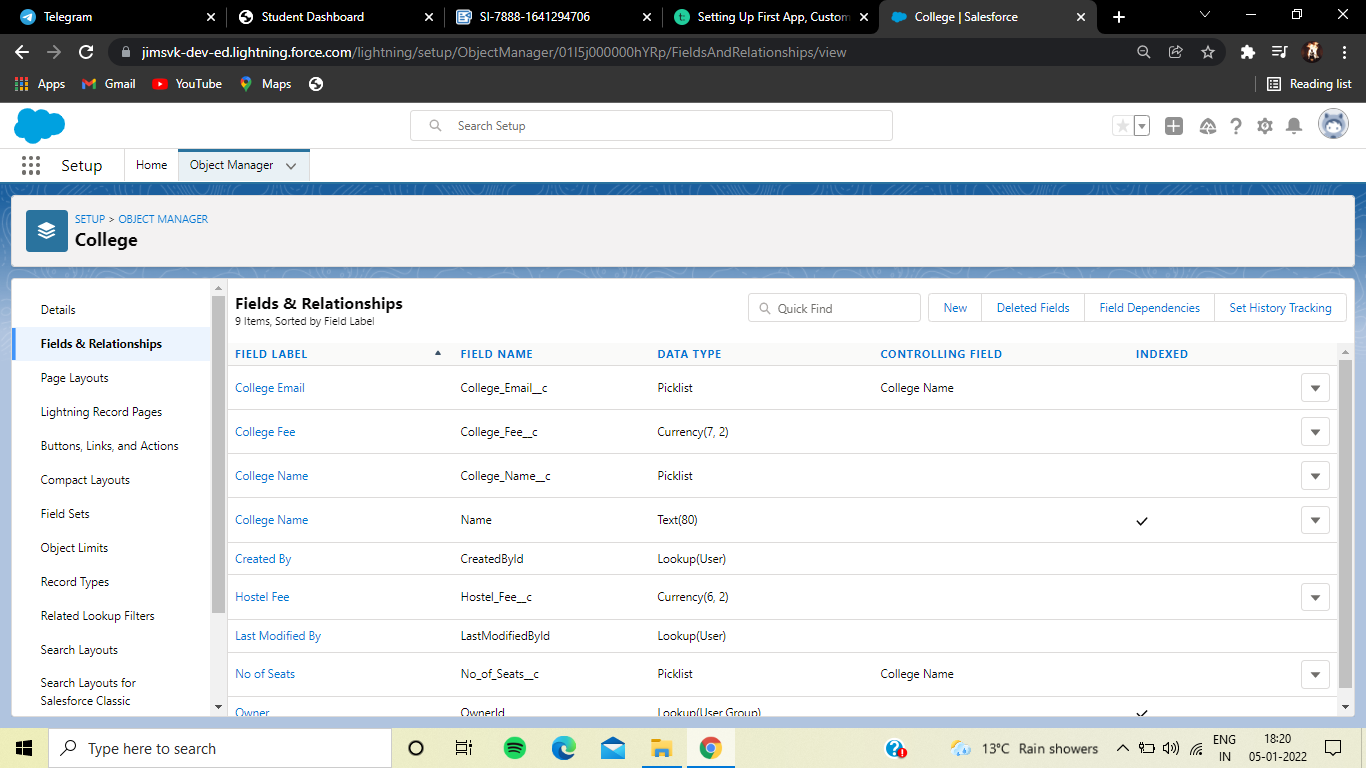
No of Seats Picklist

500- 1000, 1000-2500,

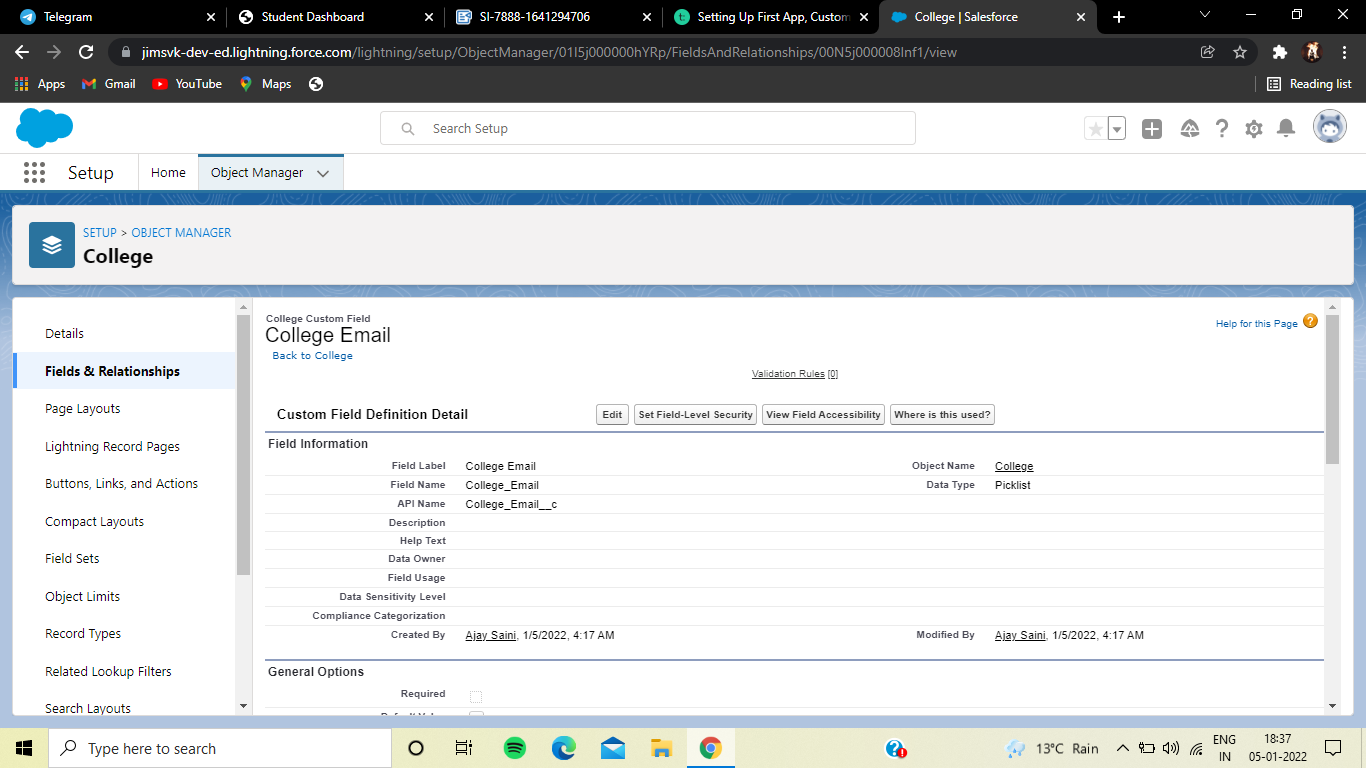
2000-6000, 6000 - 10000

College Fee Currency Yes

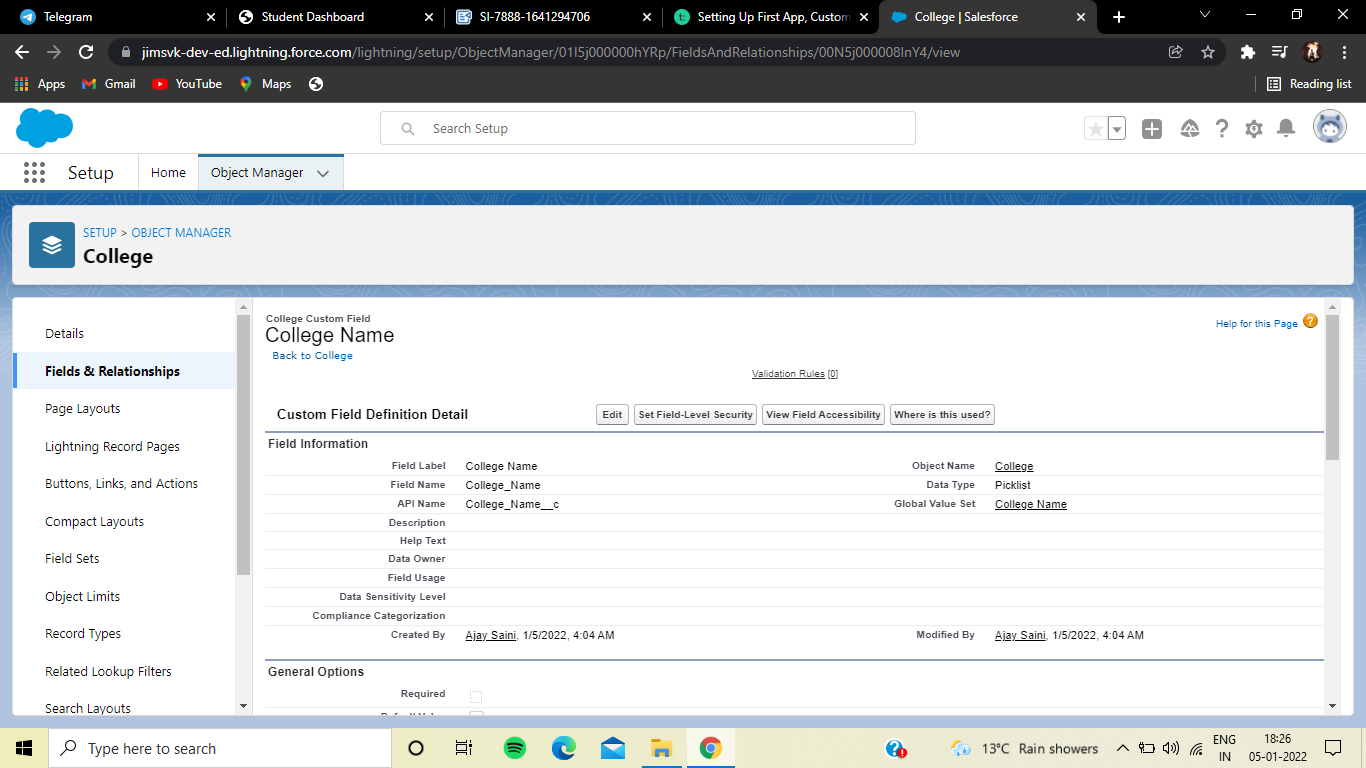
Hostel Fee Currency Yes



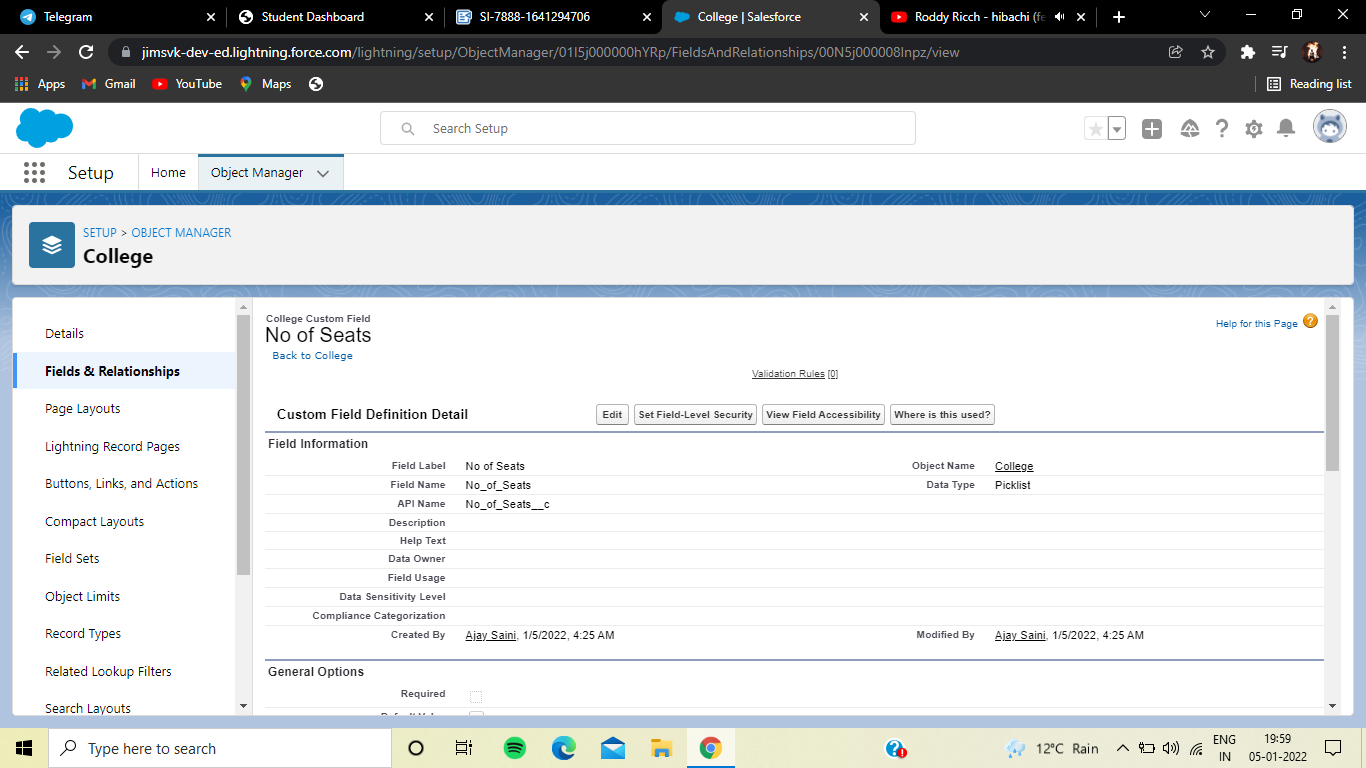
College Email :



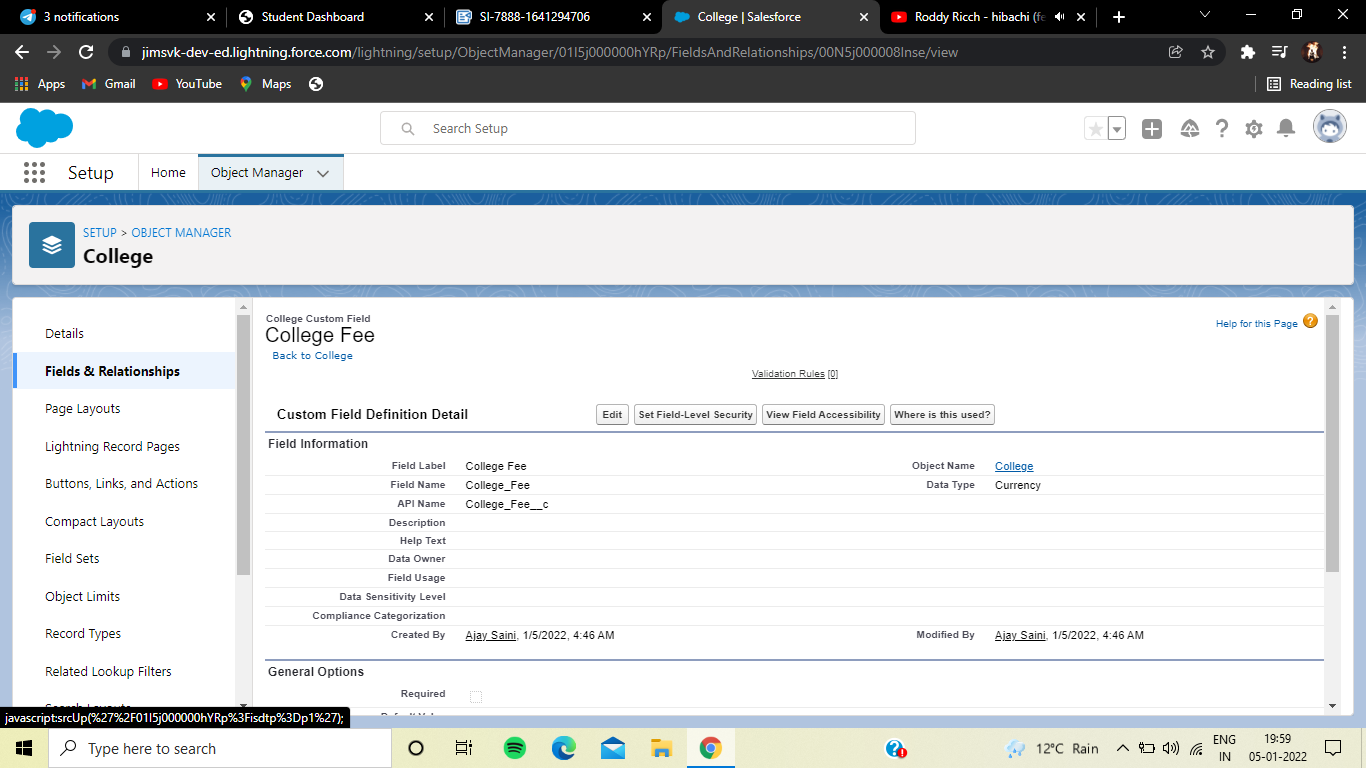
College Name:



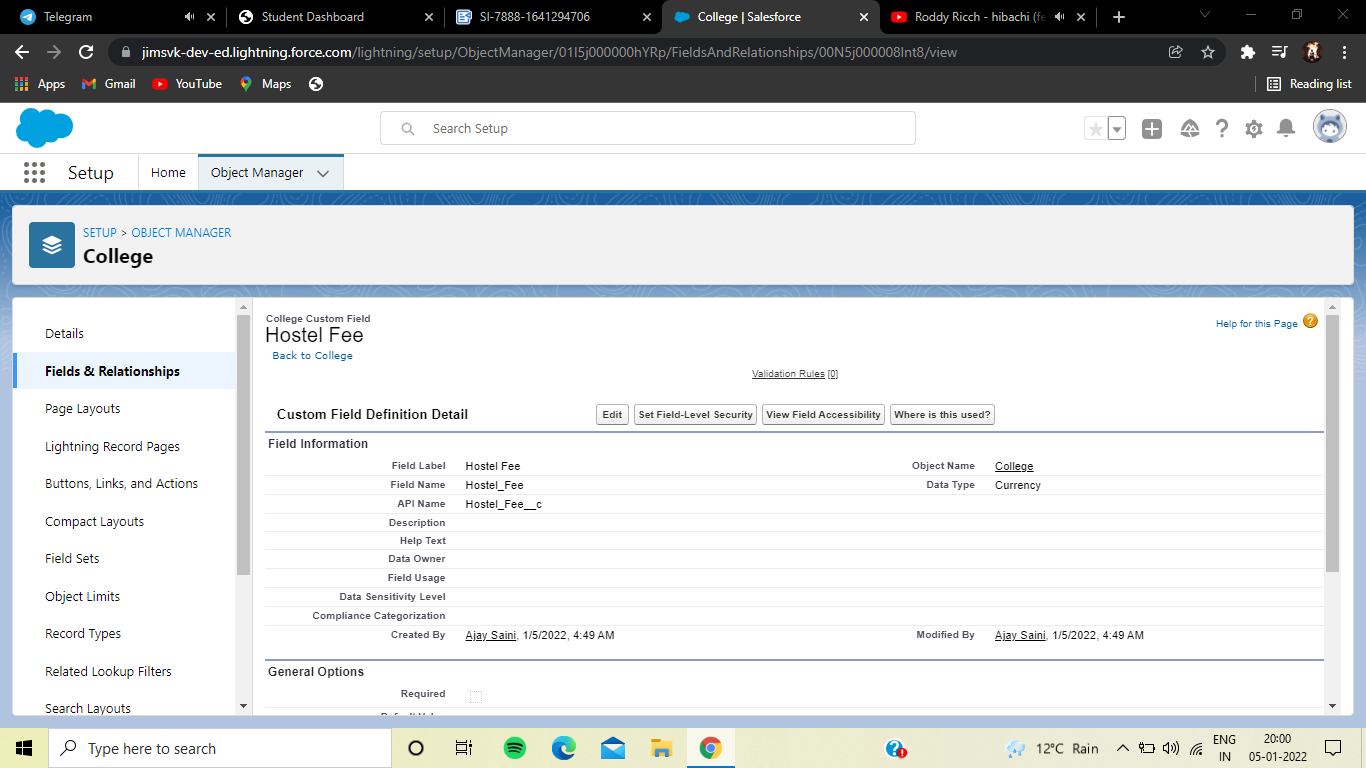
No of Seats:



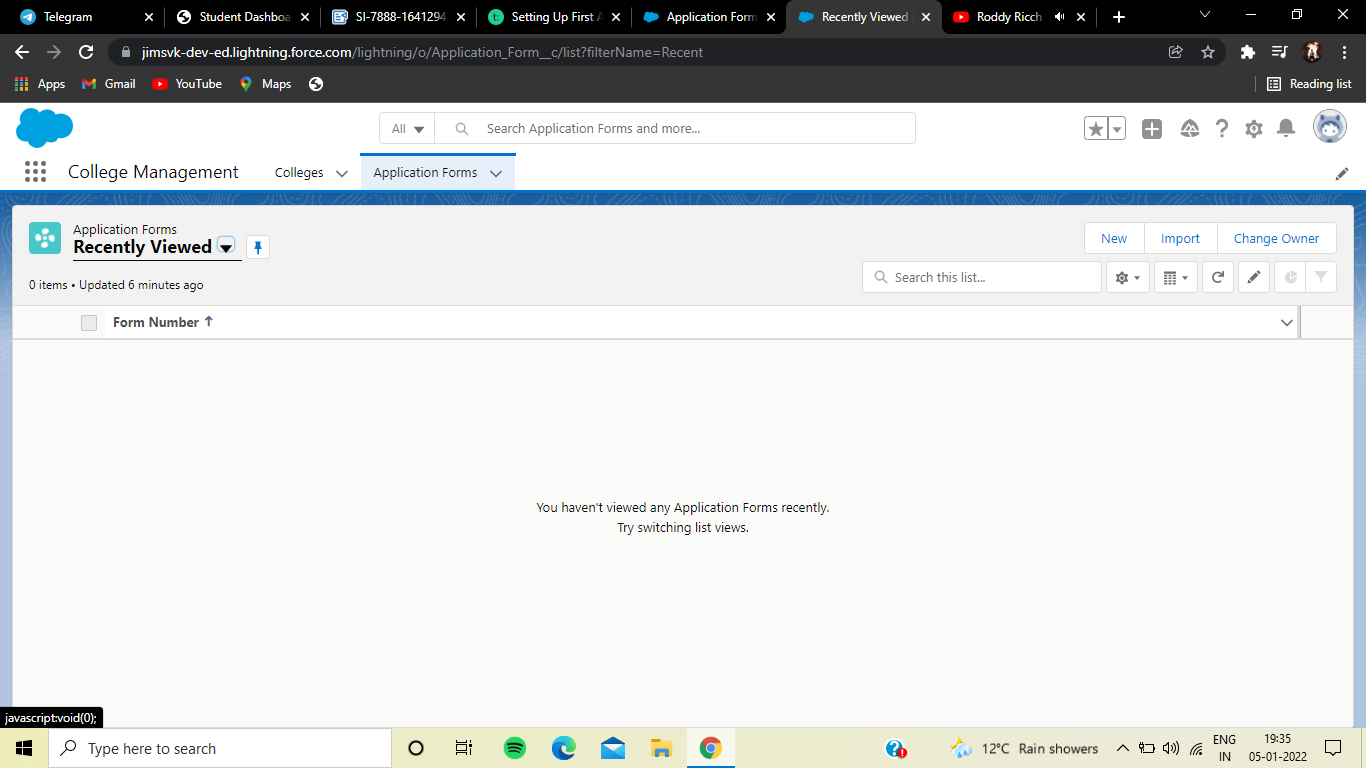
College Fee:

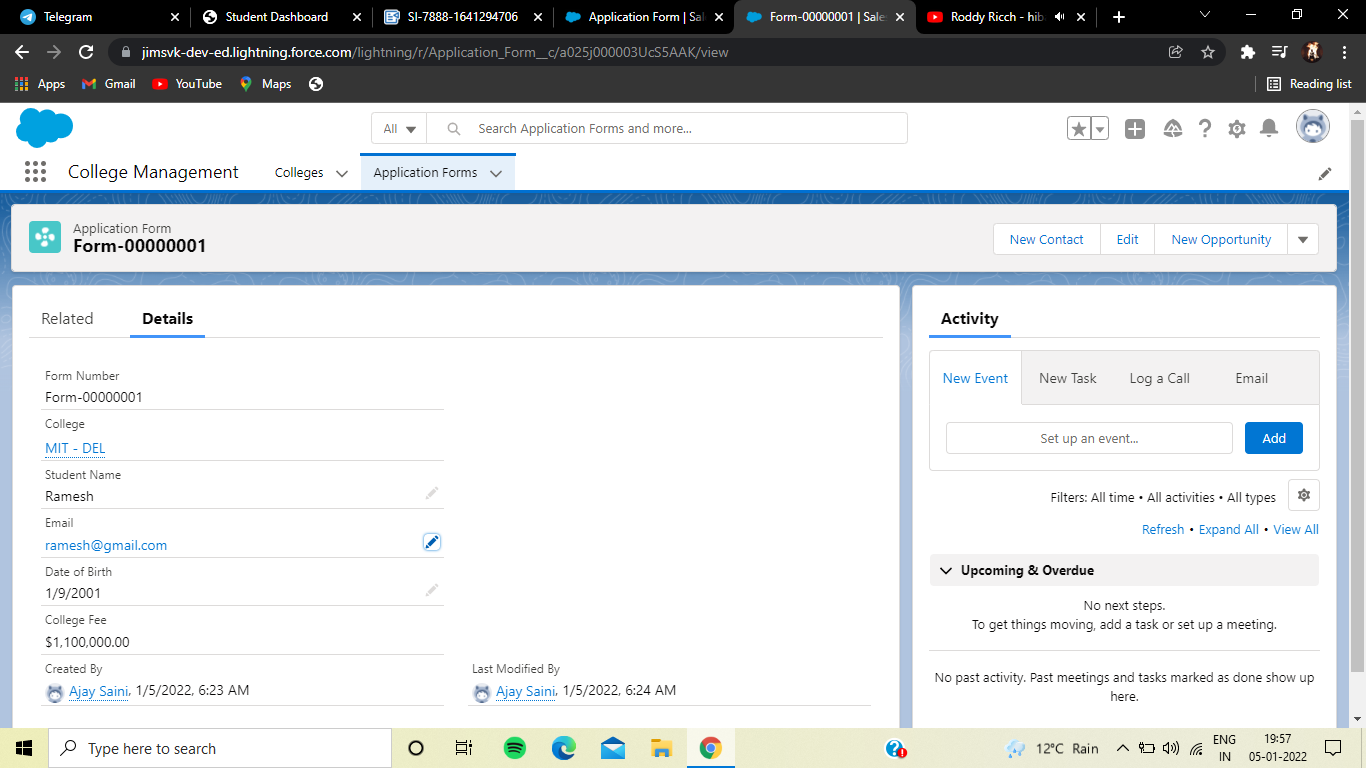


Hostel Fee:



**2. Application Form**





### **Creating Fields On Application Form Object**

### Field Name Data Type Required Values

College Master-Detail(College) Yes

College Fees Formula(Currency)

Date of Birth Date Yes

Email Email(Unique) Yes

Student Name Text(30) Yes

Phone Phone Yes

Ready To Join Checkbox(default=Uncheck)

Looking For Hostel

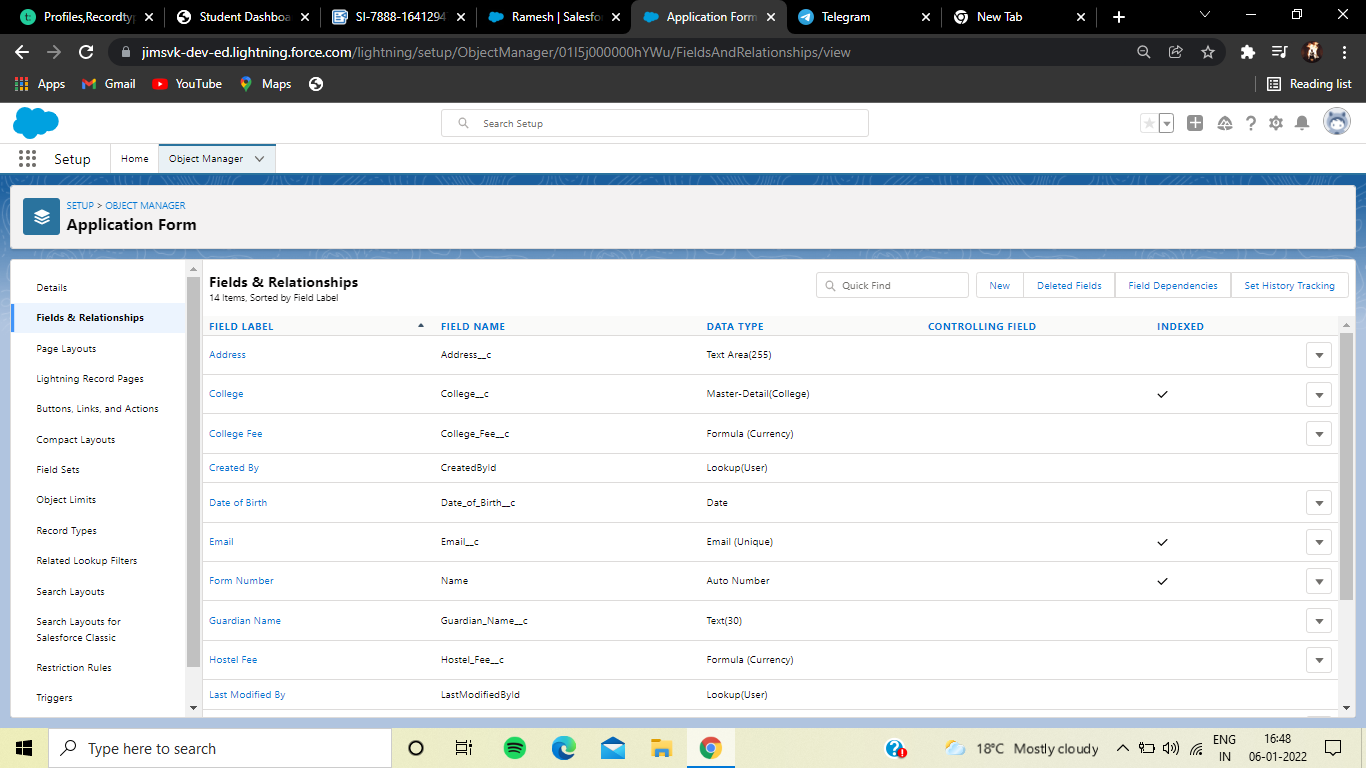
Stay Checkbox(default=Uncheck)

Address Text (255) Yes

Hostel Fee Formula(Currency)

Guardian Name Text(30)

Form Number Autonumber F-{000000} Starting Number=1



**3. Student**



### **Creating Fields On Student Object**

Field Name Data Type Required Value

Phone Phone Yes

Guardian Name Text(30) Yes

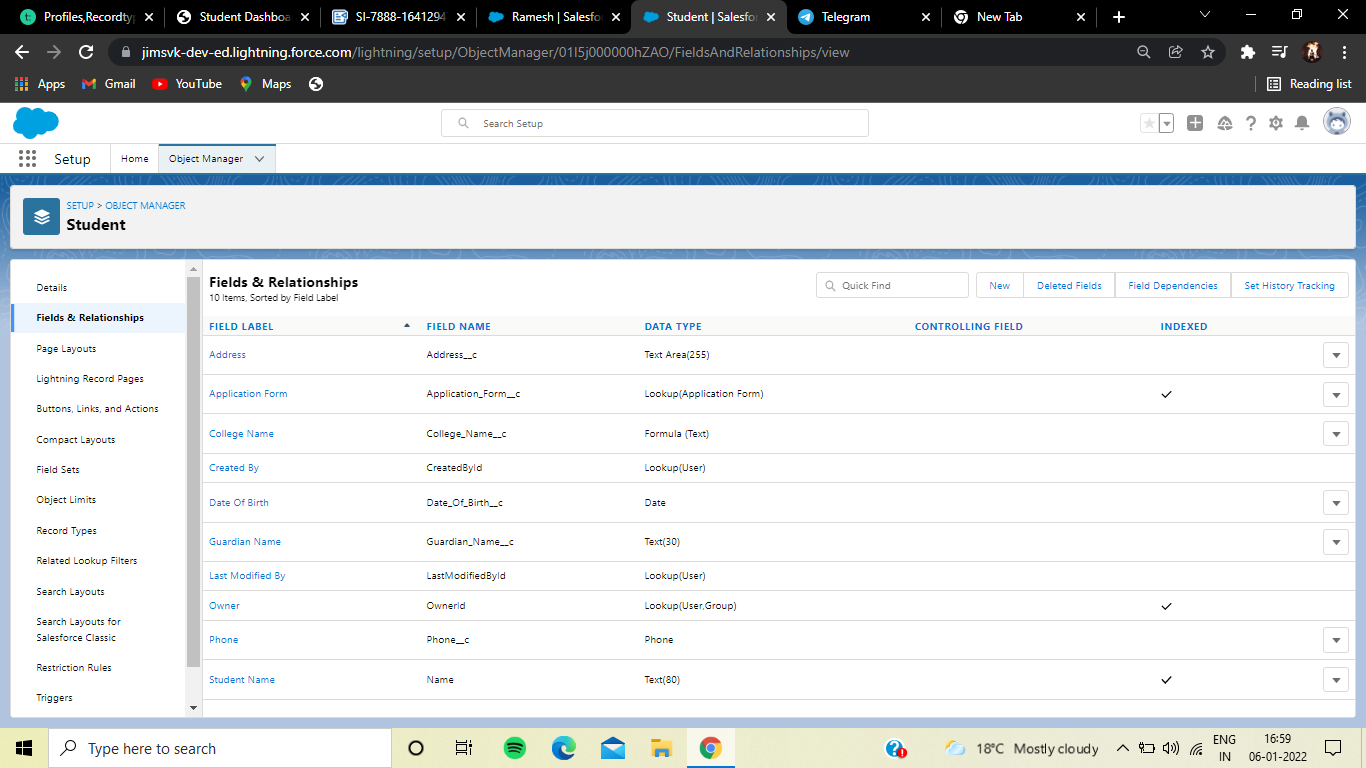
Date Of Birth Date Yes

College Name Formula(Text)

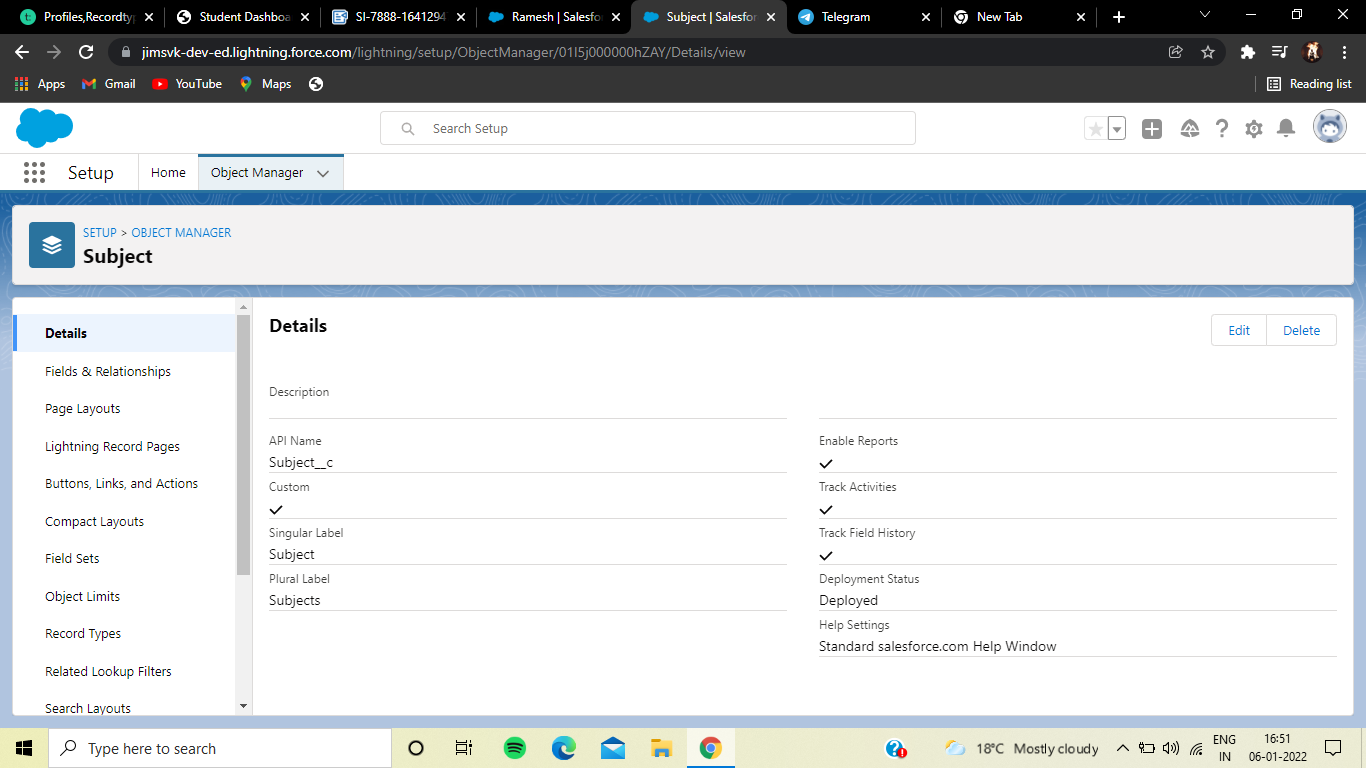
Application Form Lookup(Application Form)

Address Text(255) Yes

Student Name



**4. Subject**



### **Creating Fields On Subject Object**

Field Name Data Type Required Value

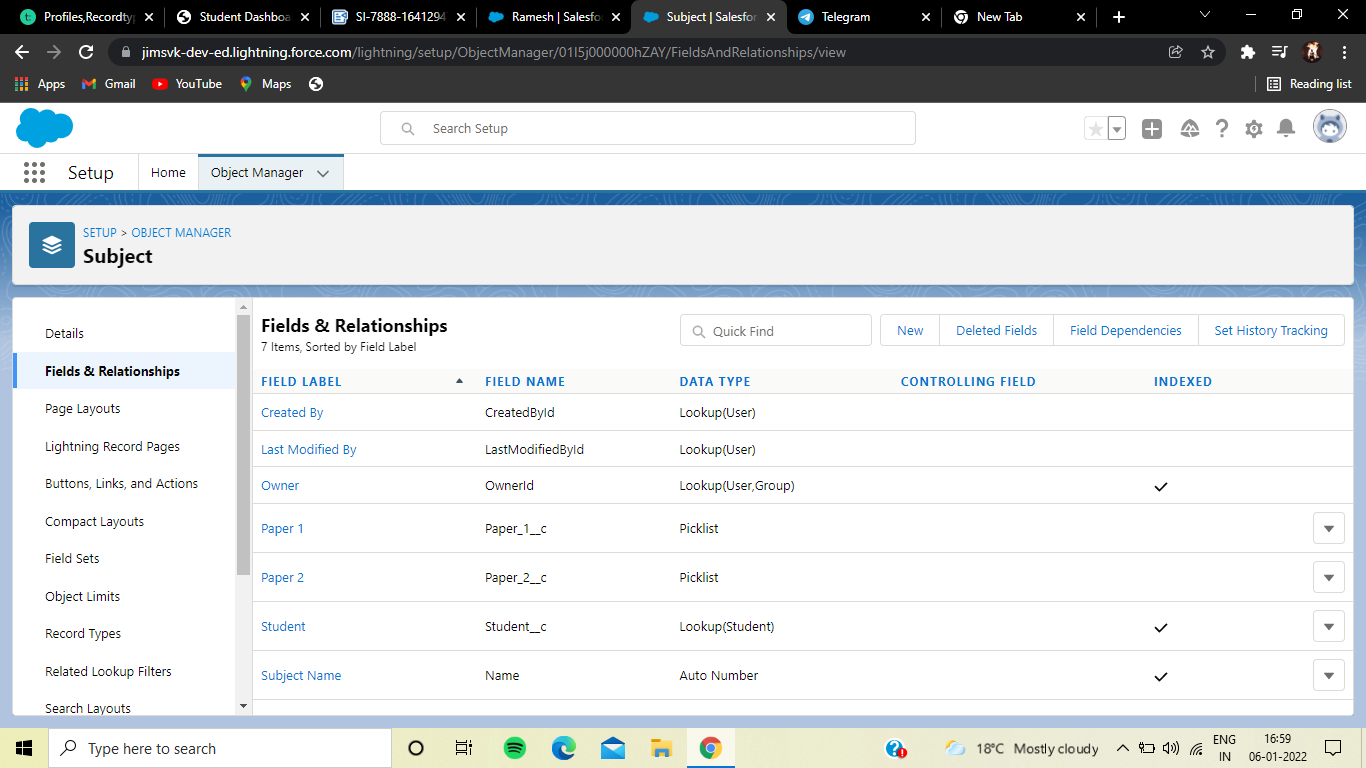
Student Lookup(Student)

Paper2 Picklist(Refer Business logic Milestone)

Paper 1 Picklist(Refer Business Logic Milestone)

Subject ID AutoNumber S-{000000}

Starting Number=1



### Adding Business Logic To Application

Business logic is the brain of the application; it makes sense of the data and responds to the button clicks in the UI.

A controller in the Model-View-Controller architecture forms the bridge between a Model (Data) and View (Display).

We can write the business logic in the Apex code as well as in a declarative syntax.

(Validations, Triggers, Formula Fields.)

### Creating Global Picklist Value Sets

Creating Global Picklist Value Sets

Created the following global picklist value sets for the application.

a)College

Picklist Value Name Values

College MIT-HYD

MIT-BLR

MIT-MUM

MIT-MAA

MIT-DEL

MIT-CCU

b)Paper1

Picklist Value Name Values

Paper 1 APEX

JAVA C

C++

c)Paper2

Picklist Value Name Values

Paper2

MATHEMATICS

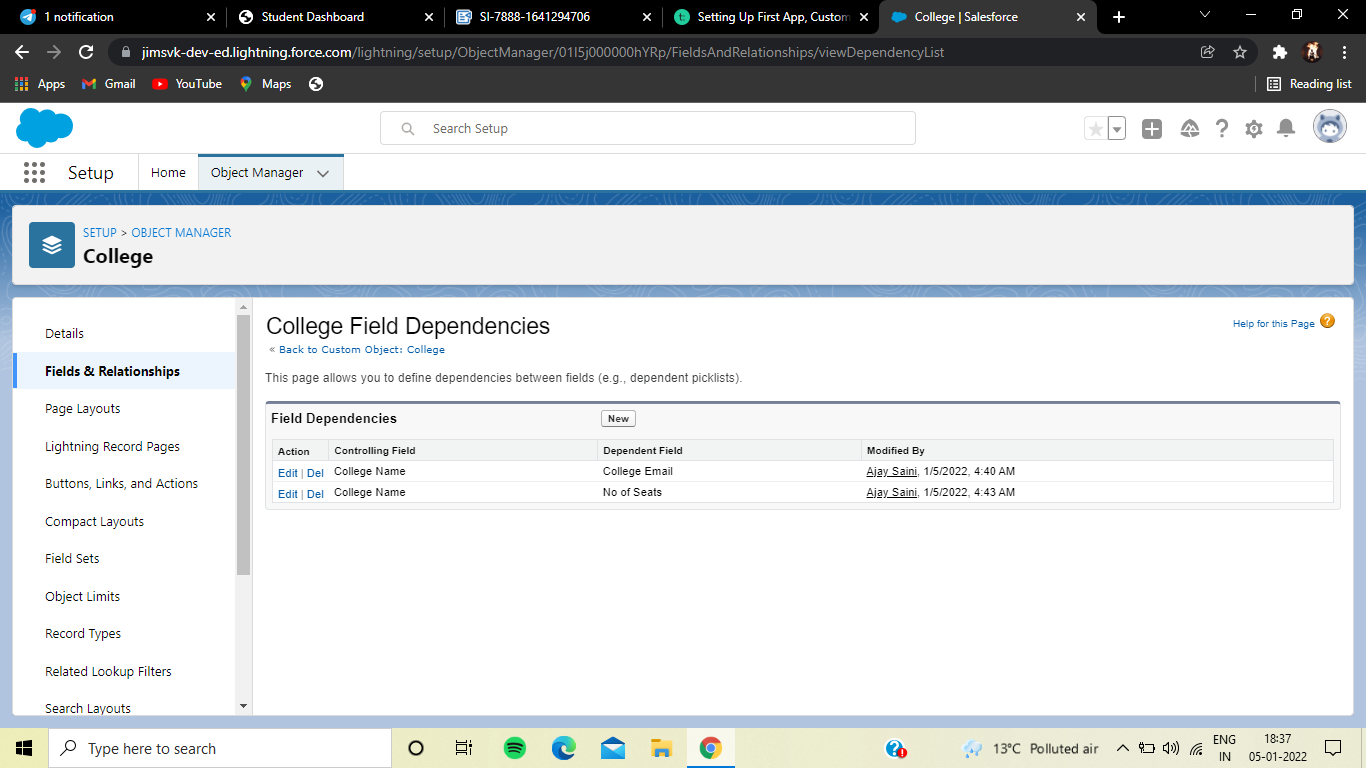
ENGLISH

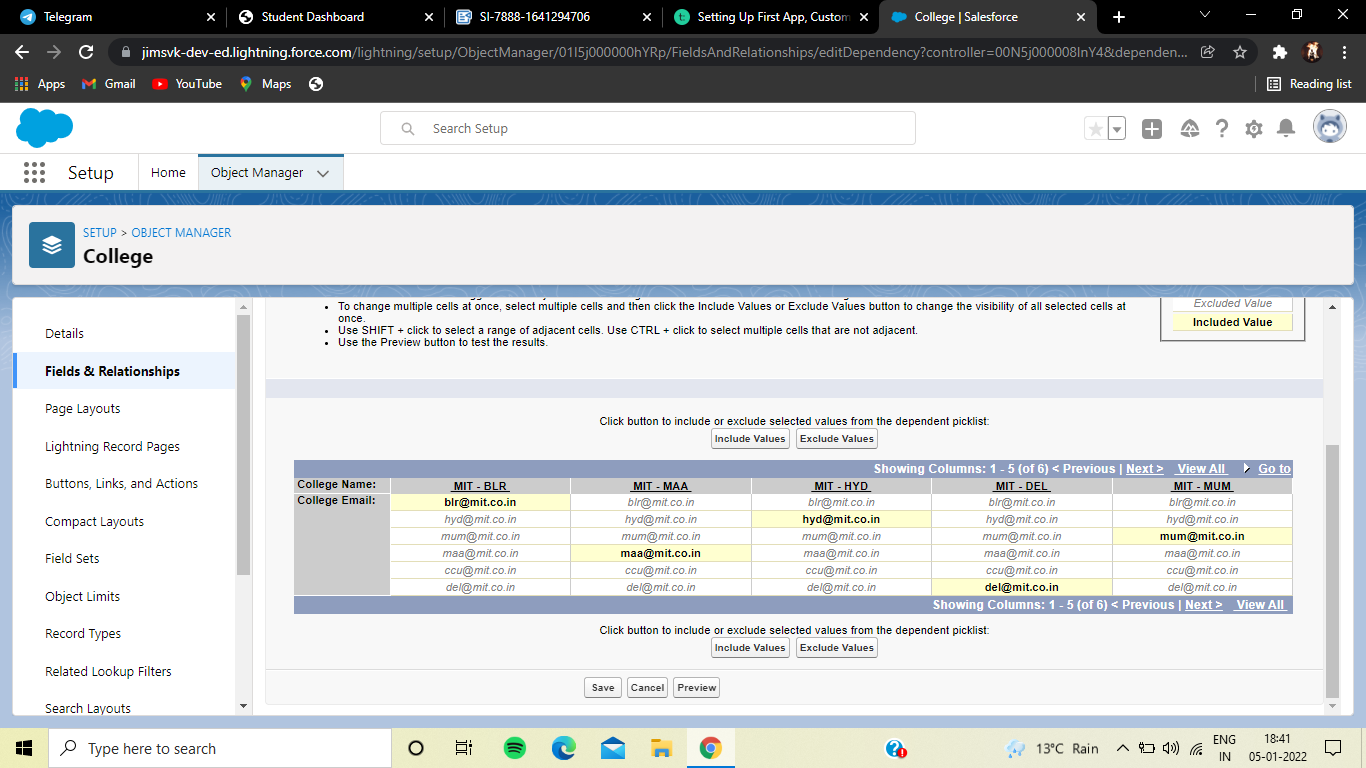
STATISTICS



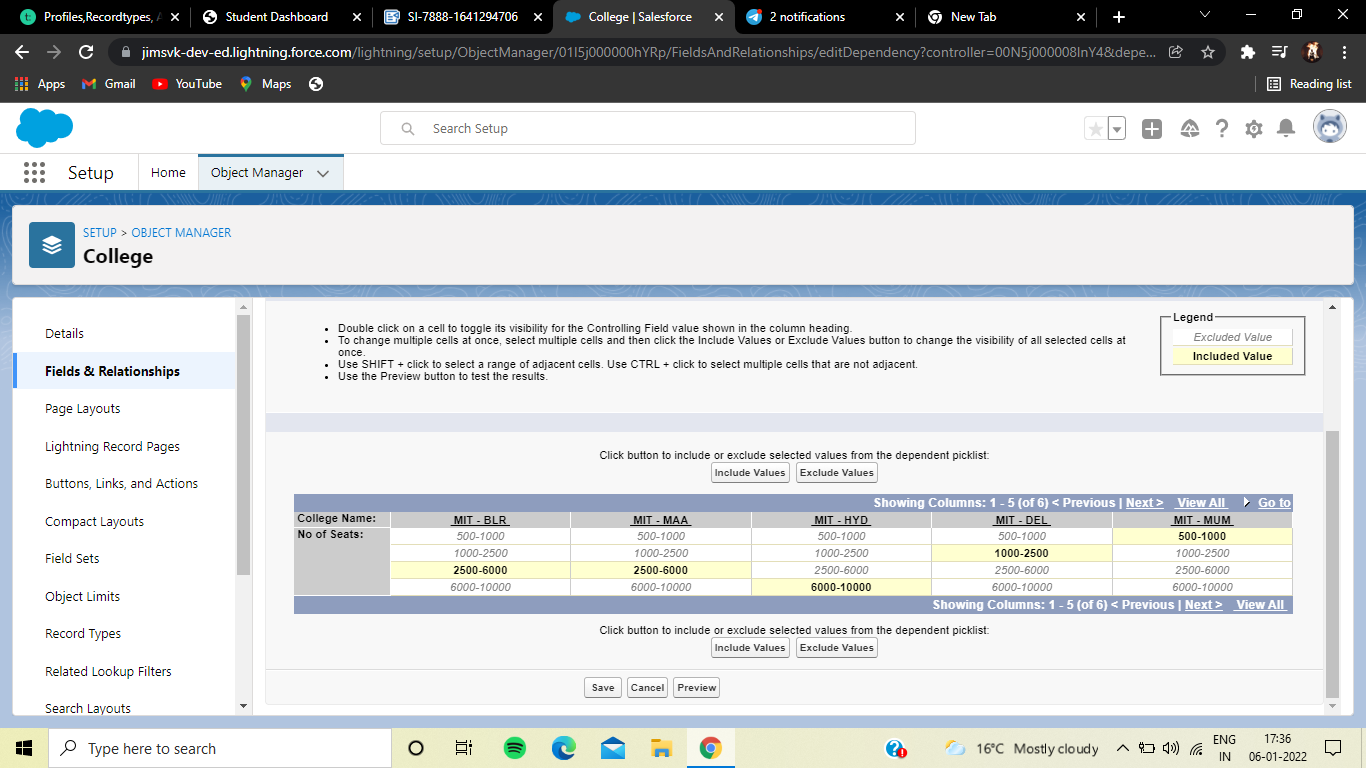
### Creating Field Dependencies

Created field dependency between college Name and Email, where the controlling field is college Name and dependent field is Email. Selected the email ids according to the college names.





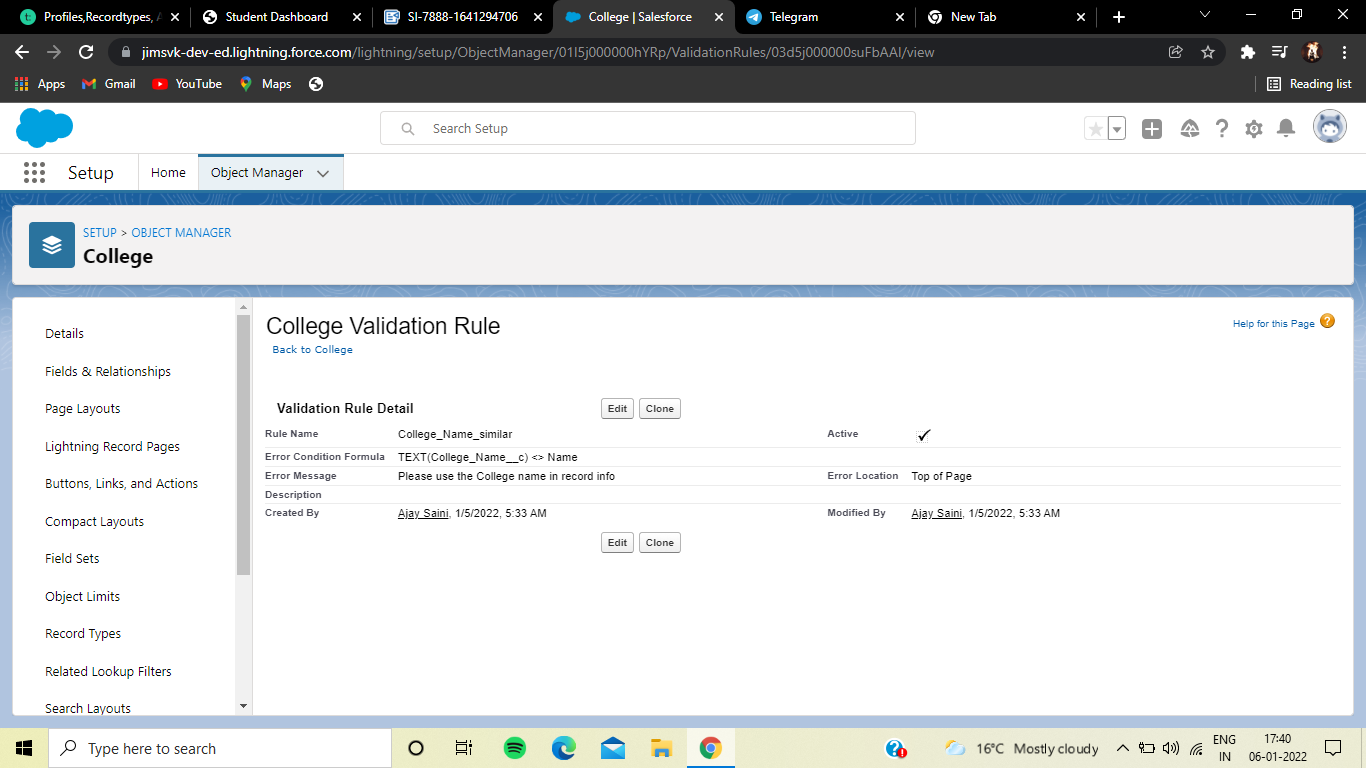
Created field dependency between college Name and capacity of students, where the controlling field is college Name and dependent field is Capacity of Students.



### Creating Validation Rules

1. Created a validation rule on the college object such that the college name and record info should have the same name.

TEXT(College\_Name\_\_c) <> Name.



2. Created a validation rule on the application form object to stop any modification on the application form once a student record is created.

AND( Ready\_To\_Join\_\_c == true,

OR( ISCHANGED( Address\_\_c ) ,

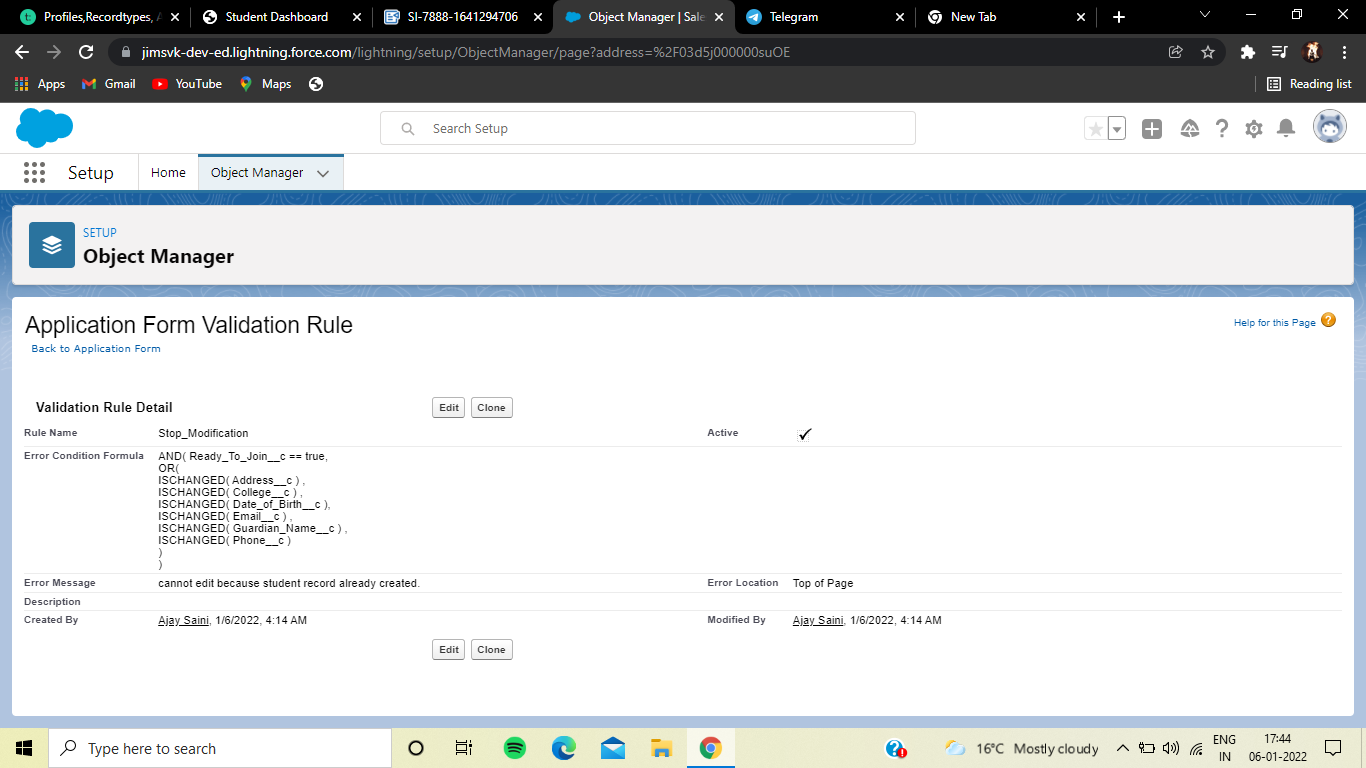
ISCHANGED( College\_\_c ) ,

ISCHANGED( Date\_Of\_Birth\_\_c ),

ISCHANGED( Email\_\_c ) ,

ISCHANGED( Guardian\_Name\_\_c ) ,

ISCHANGED( Phone\_\_c ) ) )



**Day 3 :**

### Topic: Profiles, Recordtypes, Automation

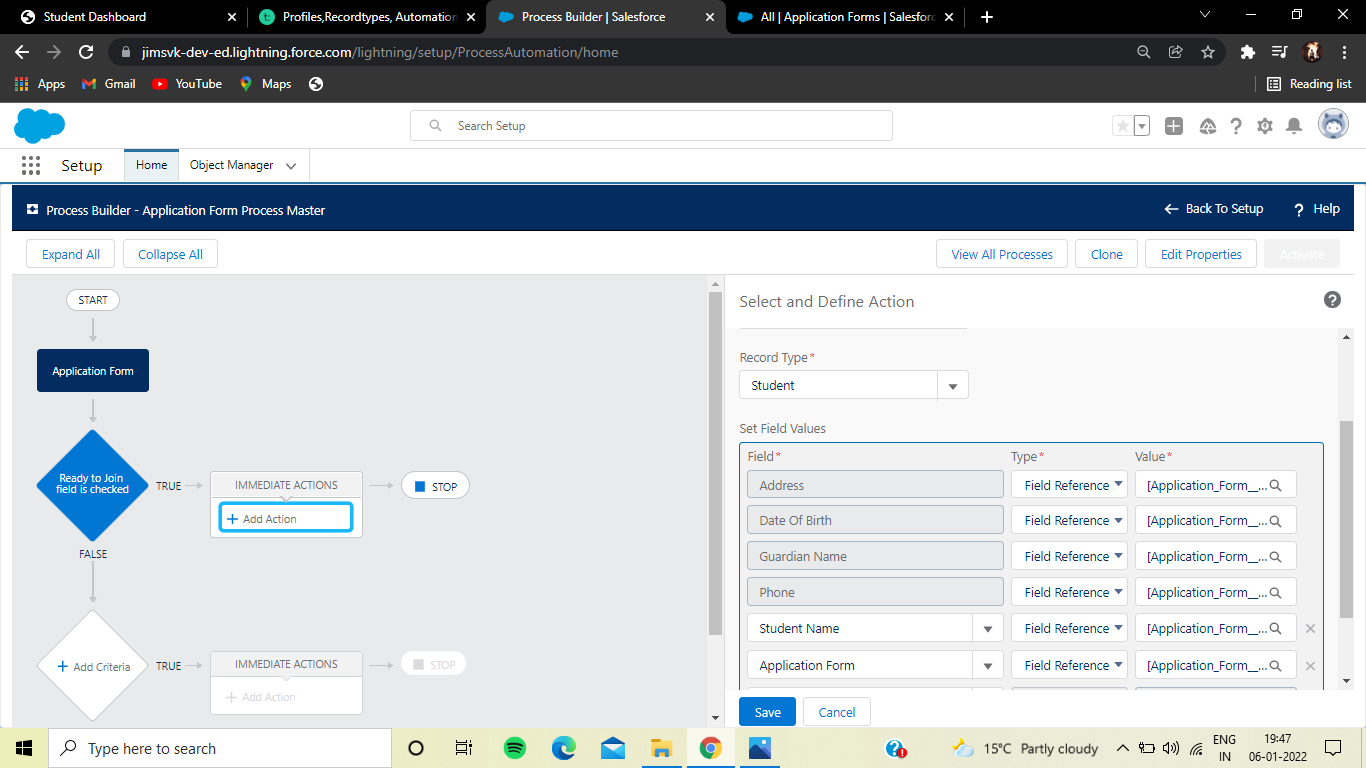
### Milestones: Creating an automation process

### 1. Create an automation process such that when the "ready to join" field is checked on the application form object we need to create the student record automatically with the information specified in the application form record.

### 2. Go to Setup -> select “Process Builder” from quick find. Create a Process Builder on the “Application Form” object with a condition as “When a record change”. And select “When a record is created or edited”.

### a. In the diamond shape box(called nodes), select the criteria which trigger the Process builder to fire. In our example, it is “When Ready to Join field is checked.”

b Once the node is setup, click on the adjacent box called “Immediate action”. And select create a record on the student object. Please follow the below screenshot.



### Day 4 :

### Topic: Automation, Security and sharing, overview of apex

### **Milestones:** Create The Student Record Using Flow:

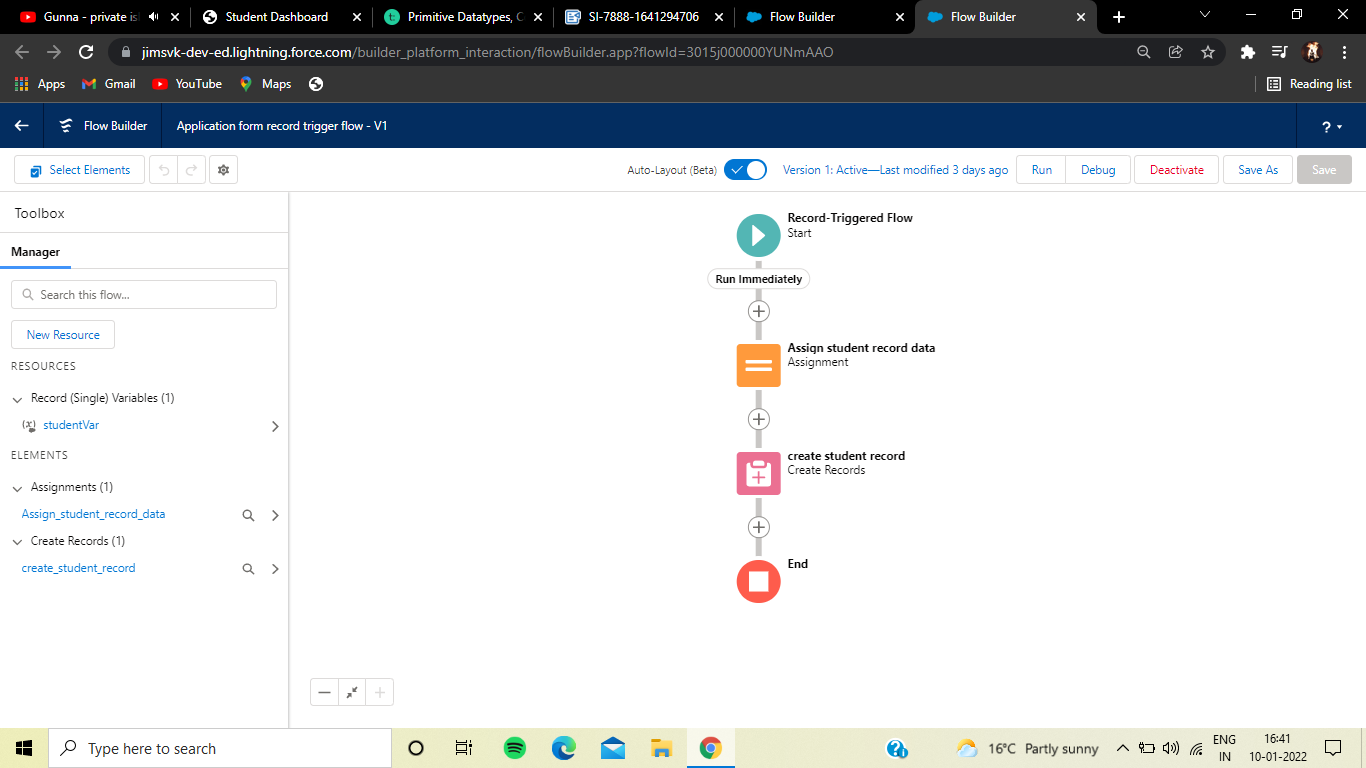
1. First deactivate the process builder which we created earlier.

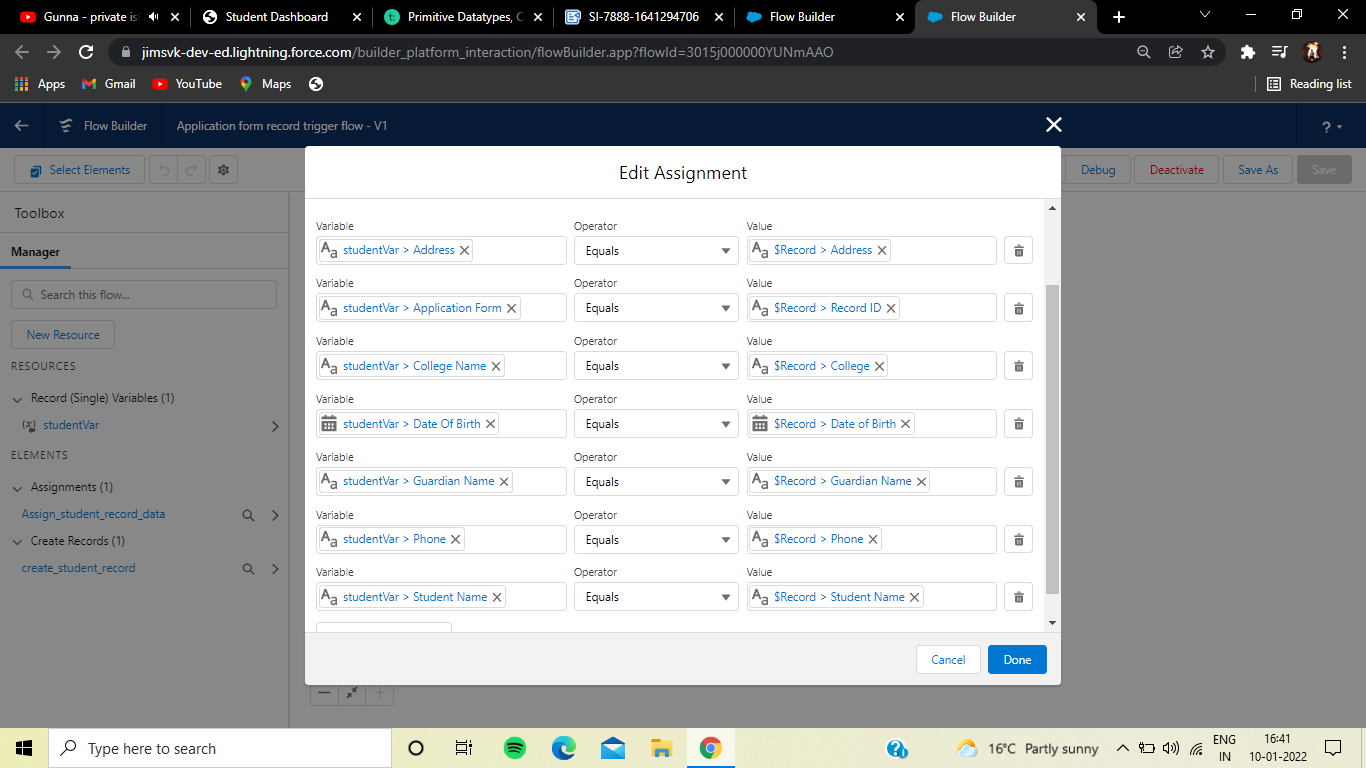
2. Now search for flows and select new flow ->record triggered flow

3. For object select application form, in configure trigger select when the record is updated for entry criteria select as ready to join equals to true.

4. Now create a variable named student in the resource section.

5.added the assignment as follows.





6. Now add create records.

### 

### Apex Overview

Apex is a strongly typed, object-oriented programming language that allows developers to execute flow and transaction control statements on the Lightning platform server in conjunction with calls to the Lightning Platform API. Using syntax that looks like Java and acts like database stored procedures, Apex enables developers to add business logic to most system events, including button clicks, related record updates, and Visualforce pages. Apex code can be initiated by Web service requests and from triggers on objects.

Apex can be stored on the platform in two different forms:

* A ***class*** is a template or blueprint from which Apex objects are created. Classes consist of other classes, user-defined methods, variables, exception types, and static initialization code. From Setup, enter Apex Classes in the Quick Find box, then select **Apex Classes**. See [Manage Apex Classes](https://help.salesforce.com/s/articleView?id=code_manage_packages.htm&type=5&language=en_US).
* A ***trigger*** is Apex code that executes before or after specific data manipulation language (DML) events occur, such as before object records are inserted into the database, or after records have been deleted. Triggers are stored as metadata in Salesforce. A list of all triggers in your organization is located on the Apex Triggers page in Setup. See [Manage Apex Triggers](https://help.salesforce.com/s/articleView?id=code_manage_triggers.htm&type=5&language=en_US).

Apex generally runs in system context; that is, the current user's permissions and field-level security aren’t taken into account during code execution. Sharing rules, however, are not always bypassed: the class must be declared with the without sharing keyword in order to ensure that sharing rules are not enforced.

You must have at least 75% of your Apex covered by unit tests before you can deploy your code to production environments. In addition, all triggers must have some test coverage. See [Apex Unit Tests](https://help.salesforce.com/s/articleView?id=code_run_tests.htm&type=5&language=en_US).

After creating your classes and triggers, as well as your tests, replay the execution using the [Developer Console](https://help.salesforce.com/s/articleView?id=code_system_log.htm&type=5&language=en_US).

You can add, edit, or delete Apex using the Salesforce user interface only in a Developer Edition organization, a Salesforce Enterprise Edition trial organization, or sandbox organization. In a Salesforce production organization, you can change Apex only by using the Metadata API deploy call, the Salesforce Extensions for Visual Studio Code, or the Ant Migration Tool. The Salesforce Extensions for Visual Studio Code and Ant Migration Tool are free resources provided by Salesforce to support its users and partners, but are not considered part of our Services for purposes of the Salesforce Master Subscription Agreement.

* **[Apex Developer Guide and Developer Tools](https://help.salesforce.com/s/articleView?id=code_apex_dev_guide_tools.htm&type=5&language=en_US)**  
  The Apex Developer Guide and Apex Reference Guide provide the complete reference for the Apex programming language. The Apex Developer Guide explains how to invoke Apex, how to work with limits, how to write tests, and more. The Apex Reference Guide provides reference information on Apex classes, interfaces, exceptions and so on. To write Apex code, you can choose from several Salesforce and third-party tools.
* **[Define Apex Classes](https://help.salesforce.com/s/articleView?id=code_define_package.htm&type=5&language=en_US)**  
  Salesforce stores Apex classes as metadata.
* **[Define Apex Triggers](https://help.salesforce.com/s/articleView?id=code_define_trigger.htm&type=5&language=en_US)**  
  Apex code can be invoked by using triggers. Apex triggers can be configured to perform custom actions before or after changes to Salesforce records, such as insertions, updates, or deletions.
* **[Executing Anonymous Apex Code](https://help.salesforce.com/s/articleView?id=code_dev_console_execute_anonymous.htm&type=5&language=en_US)**  
  The Developer Console allows you to execute Apex code as another way to generate debug logs that cover specific application logic.
* **[What Happens When an Apex Exception Occurs?](https://help.salesforce.com/s/articleView?id=code_apex_exceptions.htm&type=5&language=en_US)**  
  When an exception occurs, code execution halts. Any DML operations that were processed before the exception are rolled back and aren’t committed to the database. Exceptions get logged in debug logs. For unhandled exceptions, that is, exceptions that the code doesn’t catch, Salesforce sends an email that includes the exception information. The end user sees an error message in the Salesforce user interface.
* **[Handling Apex Exceptions in Managed Packages](https://help.salesforce.com/s/articleView?id=code_handling_managed_errors.htm&type=5&language=en_US)**
* **[Manage Apex Classes](https://help.salesforce.com/s/articleView?id=code_manage_packages.htm&type=5&language=en_US)**  
  An Apex class is a template or blueprint from which Apex objects are created. Classes consist of other classes, user-defined methods, variables, exception types, and static initialization code.
* **[Manage Apex Triggers](https://help.salesforce.com/s/articleView?id=code_manage_triggers.htm&type=5&language=en_US)**  
  A trigger is Apex code that executes before or after specific data manipulation language (DML) events occur, such as before object records are inserted into the database, or after records have been deleted.
* **[Managing Version Settings for Apex](https://help.salesforce.com/s/articleView?id=code_version_settings_apex.htm&type=5&language=en_US)**  
  To aid backwards-compatibility, classes are stored with the version settings for a specified version of Apex and the API.
* **[View Apex Classes](https://help.salesforce.com/s/articleView?id=code_view_class.htm&type=5&language=en_US)**  
  After you have created a class, you can view the code contained in the class, as well as the API against which the class was saved, and whether the class is valid or active.
* **[View Apex Trigger Details](https://help.salesforce.com/s/articleView?id=code_view_trigger.htm&type=5&language=en_US)**  
  Apex triggers are stored as metadata in the application under the object with which they are associated.
* **[Create an Apex Class from a WSDL](https://help.salesforce.com/s/articleView?id=code_wsdl_to_package.htm&type=5&language=en_US)**
* **[Monitoring the Apex Job Queue](https://help.salesforce.com/s/articleView?id=code_apex_job.htm&type=5&language=en_US)**
* **[Monitoring the Apex Flex Queue](https://help.salesforce.com/s/articleView?id=code_apex_flex_queue.htm&type=5&language=en_US)**  
  Use the Apex Flex Queue page to view and reorder all batch jobs that have a status of Holding. Or reorder your batch jobs programmatically using Apex code.
* **[Schedule Apex Jobs](https://help.salesforce.com/s/articleView?id=code_schedule_batch_apex.htm&type=5&language=en_US)**

Use the Apex scheduler and the Schedulable interface if you have specific Apex classes that you want to run on a regular basis, or to run a batch Apex job using the Salesforce user interface.

### Day 5 :

### Topic: Primitive Datatypes, Collections

### Data Types

In Apex, all variables and expressions have a data type, such as sObject, primitive, or enum.

* A primitive, such as an Integer, Double, Long, Date, Datetime, String, ID, or Boolean (see [Primitive Data Types](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_primitives.htm))
* An sObject, either as a generic sObject or as a specific sObject, such as an Account, Contact, or MyCustomObject\_\_c (see [Working with sObjects](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_SObjects.htm) in Chapter 4.)
* A collection, including:
  + A list (or array) of primitives, sObjects, user defined objects, objects created from Apex classes, or collections (see [Lists](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_collections_lists.htm))
  + A set of primitives (see [Sets](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_collections_sets.htm))
  + A map from a primitive to a primitive, sObject, or collection (see [Maps](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_collections_maps.htm))
* A typed list of values, also known as an *enum* (see [Enums](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_enums.htm))
* Objects created from user-defined Apex classes (see [Classes, Objects, and Interfaces](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_classes.htm))
* Objects created from system supplied Apex classes
* Null (for the null constant, which can be assigned to any variable)

Methods can return values of any of the listed types, or return no value and be of type Void.

Type checking is strictly enforced at compile time. For example, the parser generates an error if an object field of type Integer is assigned a value of type String. However, all compile-time exceptions are returned as specific fault codes, with the line number and column of the error

### Primitive Data Types

Apex uses the same primitive data types as SOAP API, except for higher-precision Decimal type in certain cases. All primitive data types are passed by value.

All Apex variables, whether they’re class member variables or method variables, are initialized to null. Make sure that you initialize your variables to appropriate values before using them. For example, initialize a Boolean variable to false.

Apex primitive data types include :

|  |  |
| --- | --- |
| Data Type | Description |
| Blob | A collection of binary data stored as a single object. You can convert this data type to String or from String using the toString and valueOf methods, respectively. Blobs can be accepted as Web service arguments, stored in a document (the body of a document is a Blob), or sent as attachments. For more information, see [Crypto Class](https://developer.salesforce.com/docs/atlas.en-us.232.0.apexref.meta/apexref/apex_classes_restful_crypto.htm). |
| Boolean | A value that can only be assigned true, false, or null. For example:  Boolean isWinner = true; |
| Date | A value that indicates a particular day. Unlike Datetime values, Date values contain no information about time. Always create date values with a system static method.  You can add or subtract an Integer value from a Date value, returning a Date value. Addition and subtraction of Integer values are the only arithmetic functions that work with Date values. You can’t perform arithmetic functions that include two or more Date values. Instead, use the [Date methods](https://developer.salesforce.com/docs/atlas.en-us.232.0.apexref.meta/apexref/apex_methods_system_date.htm). |
| Datetime | A value that indicates a particular day and time, such as a timestamp. Always create datetime values with a system static method.  You can add or subtract an Integer or Double value from a Datetime value, returning a Date value. Addition and subtraction of Integer and Double values are the only arithmetic functions that work with Datetime values. You can’t perform arithmetic functions that include two or more Datetime values. Instead, use the [Datetime methods](https://developer.salesforce.com/docs/atlas.en-us.232.0.apexref.meta/apexref/apex_methods_system_datetime.htm). |
| Decimal | A number that includes a decimal point. Decimal is an arbitrary precision number. Currency fields are automatically assigned the type Decimal.  If you do not explicitly set the number of decimal places for a Decimal, the item from which the Decimal is created determines the Decimal’s scale. *Scale* is a count of decimal places. Use the setScale method to set a Decimal’s scale.   * If the Decimal is created as part of a query, the scale is based on the scale of the field returned from the query. * If the Decimal is created from a String, the scale is the number of characters after the decimal point of the String. * If the Decimal is created from a non-decimal number, the number is first converted to a String. Scale is then set using the number of characters after the decimal point. |
| Double | A 64-bit number that includes a decimal point. Doubles have a minimum value of -263 and a maximum value of 263-1. For example:  Double d=3.14159;  Scientific notation (e) for Doubles is not supported. |
| ID | Any valid 18-character Lightning Platform record identifier. For example:  ID id='00300000003T2PGAA0';  If you set ID to a 15-character value, Apex converts the value to its 18-character representation. All invalid ID values are rejected with a runtime exception. |
| Integer | A 32-bit number that does not include a decimal point. Integers have a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647. For example:  Integer i = 1; |
| Long | A 64-bit number that does not include a decimal point. Longs have a minimum value of -263 and a maximum value of 263-1. Use this data type when you need a range of values wider than the range provided by Integer. For example:  Long l = 2147483648L; |
| Object | Any data type that is supported in Apex. Apex supports primitive data types (such as Integer), user-defined custom classes, the sObject generic type, or an sObject specific type (such as Account). All Apex data types inherit from Object.  You can cast an object that represents a more specific data type to its underlying data type. For example:  Object obj = 10; // Cast the object to an integer. Integer i = (Integer)obj; System.assertEquals(10, i);  The next example shows how to cast an object to a user-defined type—a custom Apex class named MyApexClass that is predefined in your organization.  Object obj = new MyApexClass(); // Cast the object to the MyApexClass custom type. MyApexClass mc = (MyApexClass)obj; // Access a method on the user-defined class. mc.someClassMethod(); |
| String | Any set of characters surrounded by single quotes. For example,  String s = 'The quick brown fox jumped over the lazy dog.';  **String size**: Strings have no limit on the number of characters they can include. Instead, the [heap size limit](https://developer.salesforce.com/docs/atlas.en-us.232.0.apexcode.meta/apexcode/apex_gov_limits.htm) is used to ensure that your Apex programs don't grow too large.  **Empty Strings and Trailing Whitespace**: sObject String field values follow the same rules as in SOAP API: they can never be empty (only null), and they can never include leading and trailing whitespace. These conventions are necessary for database storage.  Conversely, Strings in Apex can be null or empty and can include leading and trailing whitespace, which can be used to construct a message.  The Solution sObject field SolutionNote operates as a special type of String. If you have HTML Solutions enabled, any HTML tags used in this field are verified before the object is created or updated. If invalid HTML is entered, an error is thrown. Any JavaScript used in this field is removed before the object is created or updated. In the following example, when the Solution displays on a detail page, the SolutionNote field has H1 HTML formatting applied to it:  trigger t on Solution (before insert) {   Trigger.new[0].SolutionNote ='<h1>hello</h1>';  }  In the following example, when the Solution displays on a detail page, the SolutionNote field only contains HelloGoodbye:  trigger t2 on Solution (before insert) {  Trigger.new[0].SolutionNote =  '<javascript>Hello</javascript>Goodbye'; }  For more information, see “HTML Solutions Overview” in the Salesforce Help.  **Escape Sequences**: All Strings in Apex use the same escape sequences as SOQL strings: \b (backspace), \t (tab), \n (line feed), \f (form feed), \r (carriage return), \" (double quote), \' (single quote), and \\ (backslash).  **Comparison Operators**: Unlike Java, Apex Strings support using the comparison operators ==, !=, <, <=, >, and >=.  **String Methods**: As in Java, Strings can be manipulated with several standard methods.  long for the field. |
| Time | A value that indicates a particular time. Always create time values with a system static method. See [Time Class](https://developer.salesforce.com/docs/atlas.en-us.232.0.apexref.meta/apexref/apex_methods_system_time.htm). |

### Collections

Collections in Apex can be lists, sets, or maps.

* **[Lists](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_collections_lists.htm)**  
  A list is an ordered collection of elements that are distinguished by their indices. List elements can be of any data type—primitive types, collections, sObjects, user-defined types, and built-in Apex types.
* **[Sets](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_collections_sets.htm)**  
  A set is an unordered collection of elements that do not contain any duplicates. Set elements can be of any data type—primitive types, collections, sObjects, user-defined types, and built-in Apex types.
* **[Maps](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_collections_maps.htm)**  
  A map is a collection of key-value pairs where each unique key maps to a single value. Keys and values can be any data type—primitive types, collections, sObjects, user-defined types, and built-in Apex types.
* **[Parameterized Typing](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_parameterized_types.htm)**  
  Apex, in general, is a statically-typed programming language, which means users must specify the data type for a variable before that variable can be used.

### Day 6 :

### Topic: Overview Of Triggers, Batch Apex

### Milestones: Created A Batch apex For Application Form

### Apex Triggers

Apex triggers enable you to perform custom actions before or after events to records in Salesforce, such as insertions, updates, or deletions. Just like database systems support triggers, Apex provides trigger support for managing records.

Typically, you use triggers to perform operations based on specific conditions, to modify related records or restrict certain operations from happening. You can use triggers to do anything you can do in Apex, including executing SOQL and DML or calling custom Apex methods.

Use triggers to perform tasks that can’t be done by using the point-and-click tools in the Salesforce user interface. For example, if validating a field value or updating a field on a record, use validation rules and workflow rules instead.

Triggers can be defined for top-level standard objects, such as Account or Contact, custom objects, and some standard child objects. Triggers are active by default when created. Salesforce automatically fires active triggers when the specified database events occur.

### Batch Apex

Batch Apex is used to run large jobs (think thousands or millions of records!) that would exceed normal processing limits. Using Batch Apex, you can process records asynchronously in batches (hence the name, “Batch Apex”) to stay within platform limits. If you have a lot of records to process, for example, data cleansing or archiving, Batch Apex is probably your best solution.

Syntax Of BatchApex

public class MyBatchClass implements Database.Batchable<sObject> {

public (Database.QueryLocator | Iterable<sObject>) start(Database.BatchableContext bc) {

// collect the batches of records or objects to be passed to execute

}

public void execute(Database.BatchableContext bc, List<P> records){

// process each batch of records

}

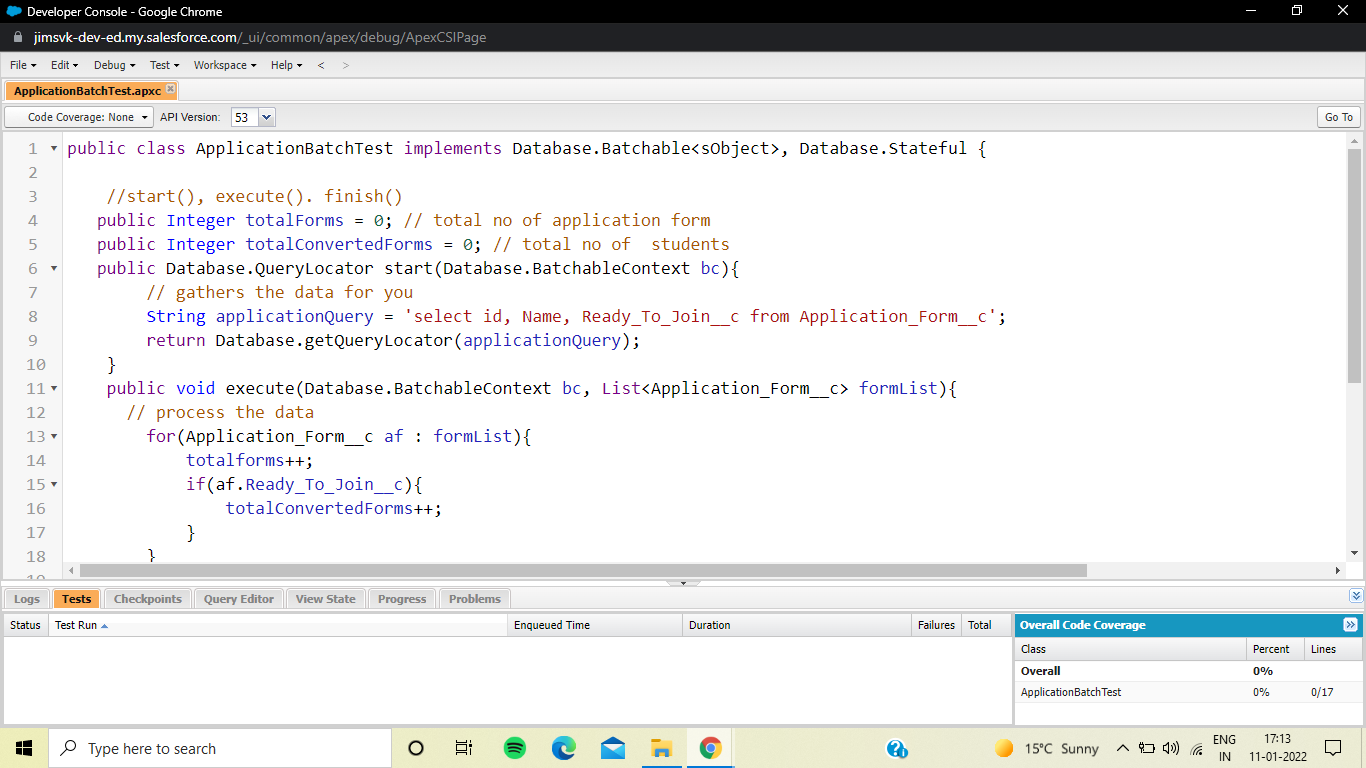
public void finish(Database.BatchableContext bc){

// execute any post-processing operations

}

}

### 1. From the developer console created a new apex class ApplicationBatchTest



public class ApplicationBatchTest implements Database.Batchable<sObject>, Database.Stateful {

//start(), execute(). finish()

public Integer totalForms = 0; // total no of application form

public Integer totalConvertedForms = 0; // total no of students

public Database.QueryLocator start(Database.BatchableContext bc){

// gathers the data for you

String applicationQuery = 'select id, Name, Ready\_To\_Join\_\_c from Application\_Form\_\_c';

return Database.getQueryLocator(applicationQuery);

}

public void execute(Database.BatchableContext bc, List<Application\_Form\_\_c> formList){

// process the data

for(Application\_Form\_\_c af : formList){

totalforms++;

if(af.Ready\_To\_Join\_\_c){

totalConvertedForms++;

}

}

}

public void finish(Database.BatchableContext bc){

// emails ,

Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

// address, subject, content( data to sent to admins)

mail.setSubject(' Application form and student record data as of today ');

mail.setPlainTextBody(' Total no of application form records are : '+totalForms+ ' out of which no of students as per today : '+totalConvertedForms);

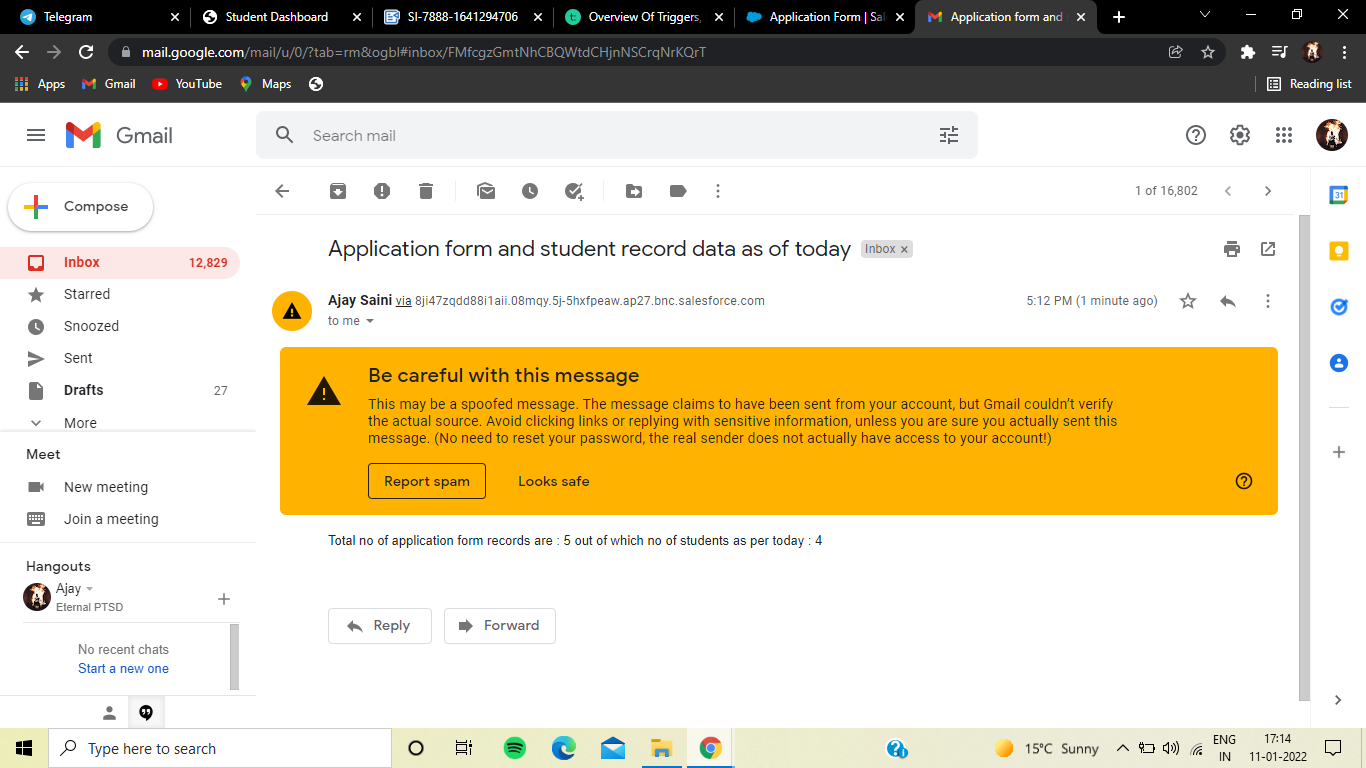
String[] emailAddress = new String[]{'sainiajay882@gmail.com'};

mail.setToAddresses(emailAddress);

Messaging.sendEmail(new Messaging.SingleEmailMessage[]{ mail } );

}

}



### Create A Schedular Class

1. From the developer console create a new apex class applicationSchedule

Code :

public class applicationSchedule implements Schedulable{

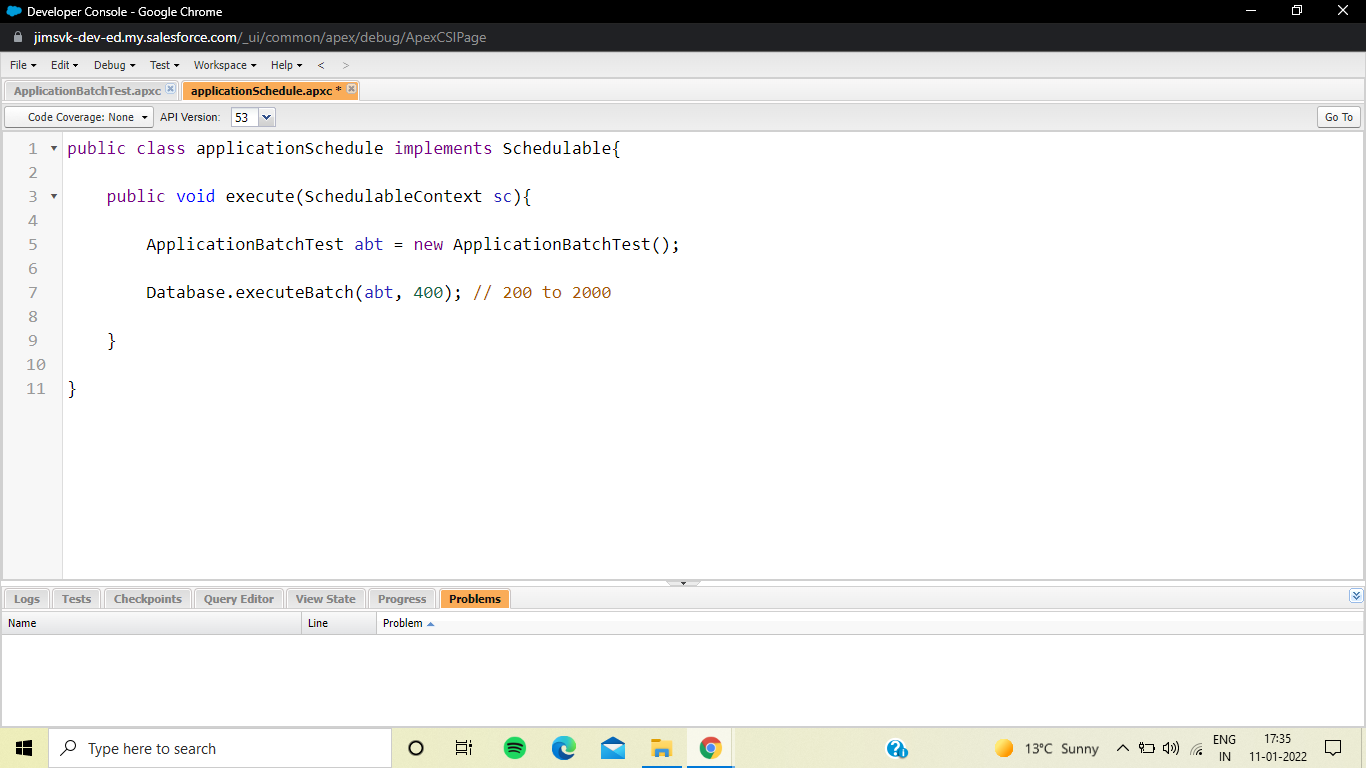
public void execute(SchedulableContext sc){

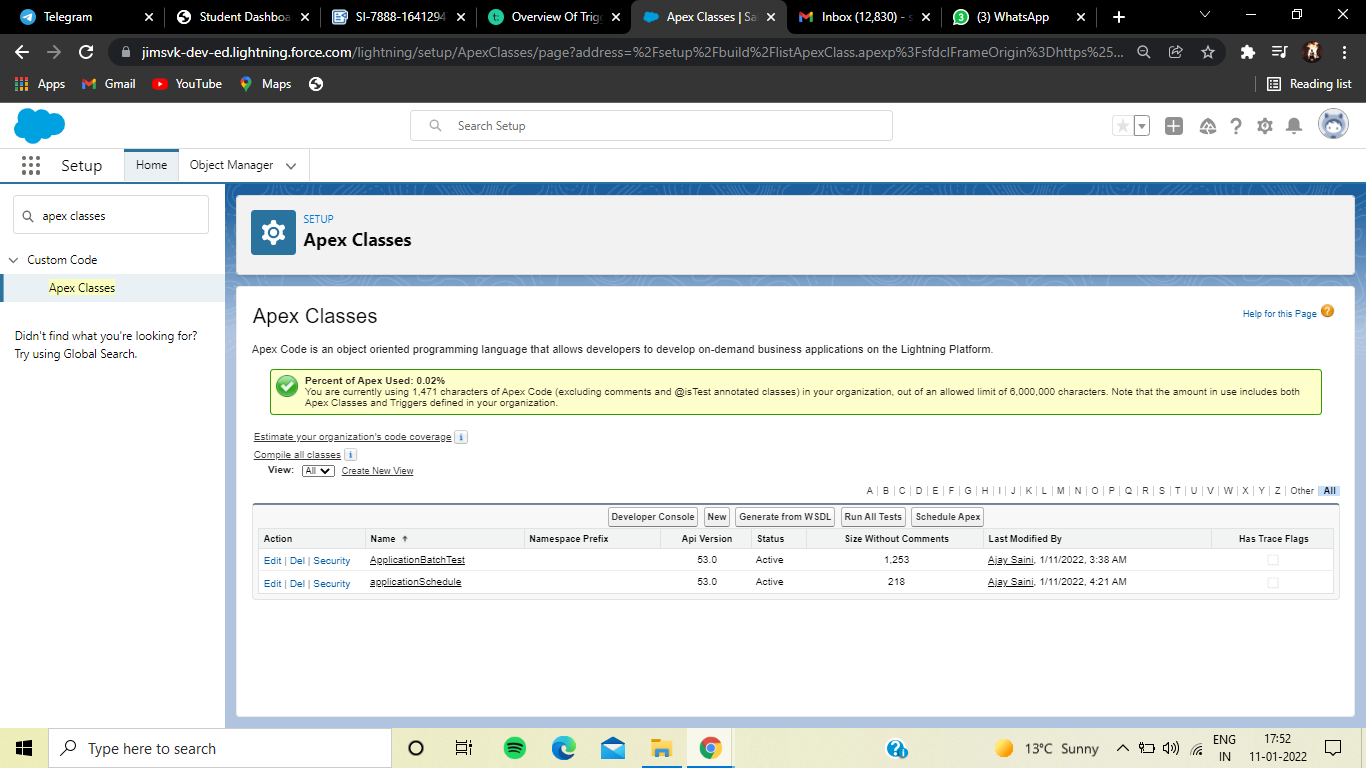
ApplicationBatchTest abt = new ApplicationBatchTest();

Database.executeBatch(abt, 400); // 200 to 2000

}

}





### Day 7 :

### Topic: Triggers

### Milestones: Created Application Form Trigger

**Apex** can be invoked by using *triggers*. Apex triggers enable you to perform custom actions before or after changes to Salesforce records, such as insertions, updates, or deletions.

A trigger is Apex code that executes before or after the following types of operations:

* insert
* update
* delete
* merge
* upsert
* undelete

For example, you can have a trigger run before an object's records are inserted into the database, after records have been deleted, or even after a record is restored from the Recycle Bin.

You can define triggers for top-level standard objects that support triggers, such as a Contact or an Account, some standard child objects, such as a CaseComment, and custom objects. To define a trigger, from the object management settings for the object whose triggers you want to access, go to Triggers.

There are two types of triggers:

* *Before triggers* are used to update or validate record values before they’re saved to the database.
* *After triggers* are used to access field values that are set by the system (such as a record's Id or LastModifiedDate field), and to affect changes in other records, such as logging into an audit table or firing asynchronous events with a queue. The records that fire the *after trigger* are read-only.

Triggers can also modify other records of the same type as the records that initially fired the trigger. For example, if a trigger fires after an update of contact *A*, the trigger can also modify contacts *B*, *C*, and *D*. Because triggers can cause other records to change, and because these changes can, in turn, fire more triggers, the Apex runtime engine considers all such operations a single unit of work and sets limits on the number of operations that can be performed to prevent infinite recursion. See [Execution Governors and Limits](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_gov_limits.htm).

Additionally, if you update or delete a record in its before trigger, or delete a record in its after trigger, you will receive a runtime error. This includes both direct and indirect operations. For example, if you update account *A*, and the before update trigger of account *A* inserts contact *B*, and the after insert trigger of contact *B* queries for account *A* and updates it using the DML update statement or database method, then you are indirectly updating account *A* in its before trigger, and you will receive a runtime error.

### Applicationformtrigger

trigger applicationFormTrigger on Application\_Form\_\_c (before insert, after insert, before update, after update) {

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

formhelper1.getoldNewValues(trigger.oldMap, trigger.new);

}

}

### formhelper1

public class formhelper1 {

public static void getoldNewValues(Map<id, Application\_Form\_\_c> formoldMap, List<Application\_Form\_\_c> formlist){

for(Application\_Form\_\_c fm : formlist){

Application\_Form\_\_c oldFormRecord = formoldMap.get(fm,id);

System.debug('old values : '+oldFormRecord.Student\_Name\_\_c);

System.debug('new values : '+ fm.Student\_Name\_\_c+ 'oldvalues : '+formoldMap.get(fm.id).Student\_Name\_\_c);

if(fm.Student\_Name\_\_c != formoldMap.get(fm,id).Student\_Name\_\_c)[

]

}

}

}

formoldMap.get(fm,id).Student\_Name\_\_c

map.get(key) = value

### Day 8-9:

### Topic: LWC

**Milestones: Setting up LWC**

### Salesforce CLI

Like many other programming languages and models, Salesforce includes a command-line interface (CLI). If you’ve ever used npm, yarn, gradle, or maven, Salesforce CLI will seem familiar to you—just tailor-made for Salesforce development tasks (and if those acronyms look like alphabet soup to you, that’s fine, too).

Salesforce CLI allows you to interact with your Salesforce environments in many ways, like retrieving or pushing code or interacting with data. The CLI consists of several plugins. These plugins provide important specific functionality. For example, the salesforcedx plugin provides the ability to interact with Salesforce orgs and their data.

Complete installation instructions.

1.Install the CLI from <https://developer.salesforce.com/tools/sfdxcli>.

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

**sfdx update**

You should see output like **sfdx-cli: Updating CLI....**

### Visual Studio Code

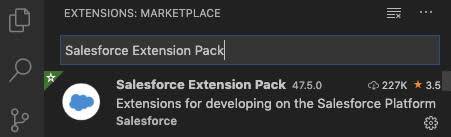
Visual Studio Code is the go-to code editor for Salesforce developers. It is free, open-source, and available for Windows, Linux, and macOS. Visual Studio Code is a well-established IDE among web developers. Now it’s also an effective IDE for building Lightning web components, and Salesforce provides free extensions for Visual Studio Code that simplify your development experience even more.

Follow these instructions to install Visual Studio Code.

1.Download and install the latest version of [Visual Studio Code](https://code.visualstudio.com/) for your operating system. If you already have Visual Studio Code installed, there’s no need to reinstall it.

2.Launch Visual Studio Code.

3.Click the  icon for Extensions in the sidebar.

4.Search for Salesforce Extension Pack and click Install. If you already have it installed, then you just need to click on the Reload button.  


5.Press Command + Shift + P on macOS or Ctrl + Shift + P on Windows or Linux to reveal the command palette. In the command palette, type sfdx to display an initial list of available commands.

Done! You’ve installed all the tools you need for developing your first Lightning web component.

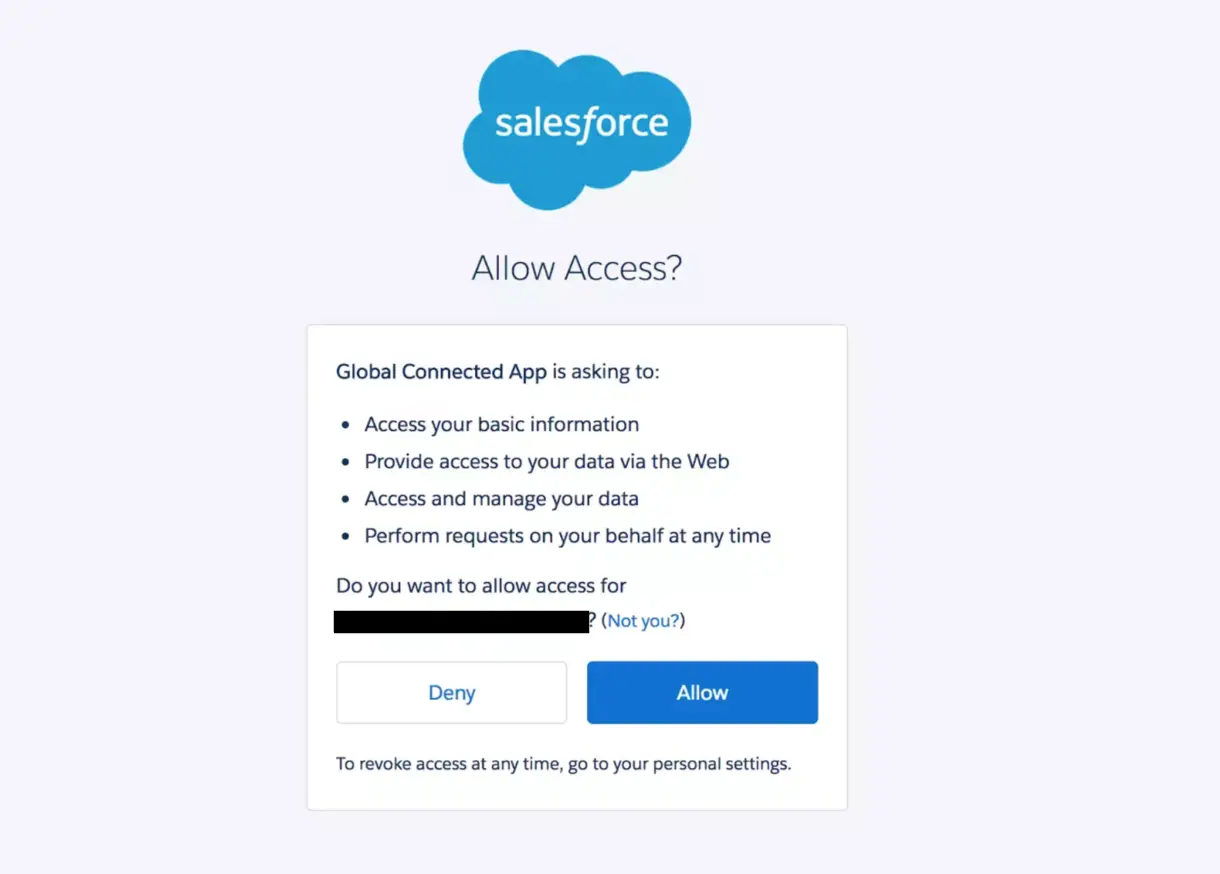
### Create a Salesforce DX Project

### The basic foundation for interacting with an org using Salesforce CLI is a Salesforce DX project. A project consists of several local configuration files, as well as the code you want to deploy. In Salesforce terms we call this code metadata, which is the foundation of the Salesforce Platform. If you’re new to Salesforce, check out Platform Development Basics to learn more about our metadata-driven approach.

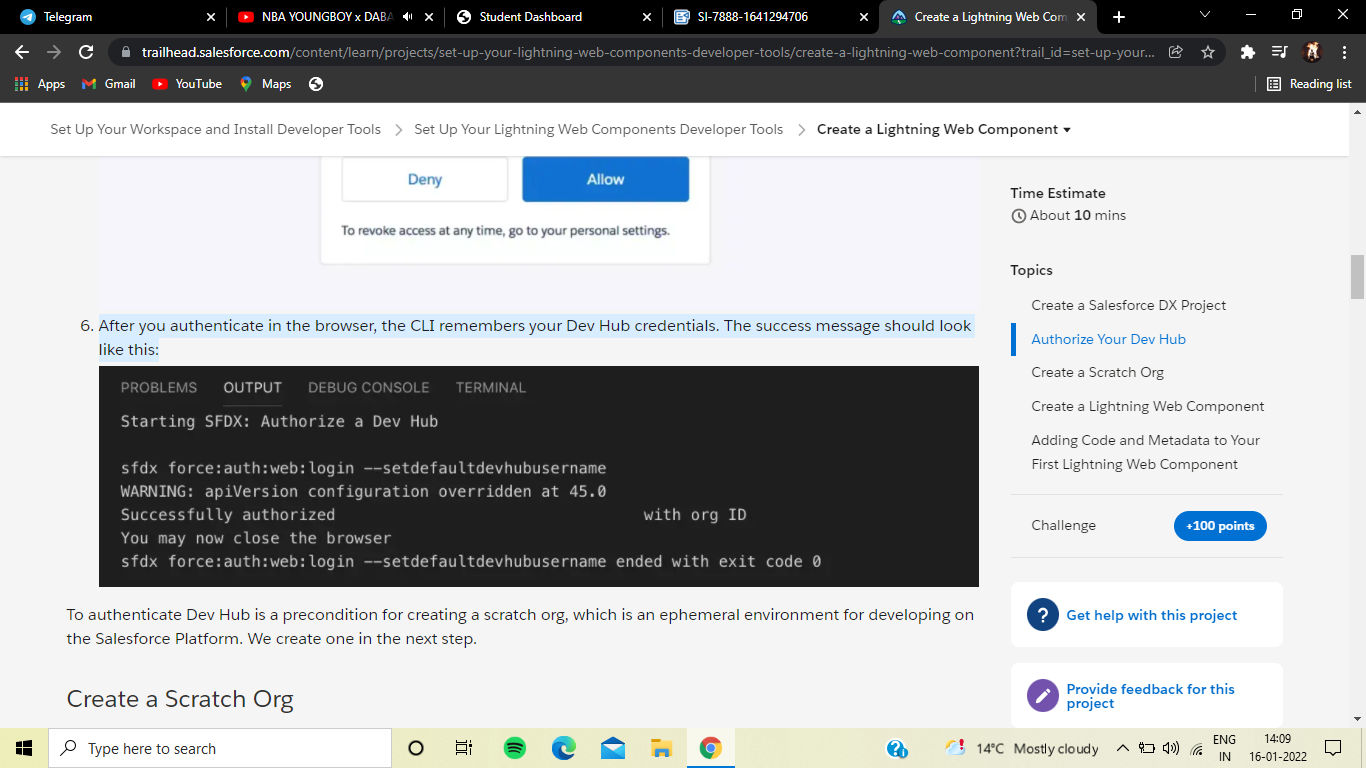
1. Open **Visual Studio Code**.
2. Press **Command + Shift + P** on macOS or **Ctrl + Shift + P** on Windows or Linux, then type **create project**. Select **SFDX: Create Project**, and press **Enter**.
3. Leave the default project type selection **Standard** as is, and press **Enter**.
4. Enter **trailhead** as project name, and press **Enter**.
5. Choose a directory on your local machine where the project will be stored. Click **Create Project**.

### Authorize Your Dev Hub

1. In **Visual Studio Code**, press **Command + Shift + P** on macOS or **Ctrl + Shift + P** on Windows or Linux.
2. Type **sfdx**.
3. Select **SFDX: Authorize a Dev Hub**.
4. Log in using your Dev Hub org credentials.
5. Click **Allow**.



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



### Create a Scratch Org

1. In Visual Studio Code, press **Command + Shift + P** on macOS or **Ctrl + Shift + P** on Windows or Linux.
2. Type **sfdx**.
3. Select **SFDX: Create a Default Scratch Org...**.
4. Press **Enter** to accept the default project-scratch-def.json.
5. Press **Enter** to accept the default trailhead scratch org alias.
6. Press **Enter** to accept the default 7 days scratch org duration.
7. Be patient, creating a scratch org can take a minute. The success message should look like this in the output panel of VS Code:

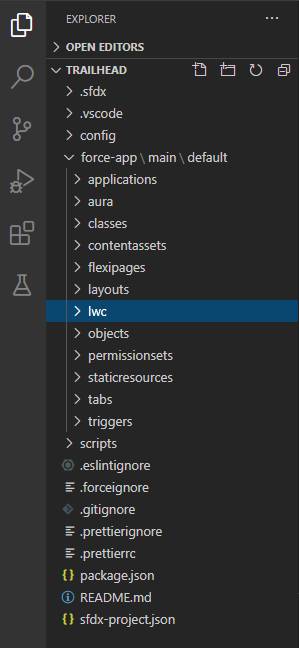
17:18:11.779 sfdx force:org:create -f config\project-scratch-def.json --setalias trailhead --durationdays 7 --setdefaultusername --json --loglevel fatal ended with exit code 0

Now you’re all set to develop your first Lightning web component

### Create a Lightning Web Component

Creating a Lightning web component is a straightforward process. And Salesforce CLI already created a project structure that helps make getting started even easier.

The folder structure looks like this:



The project we created has a special folder, force-app/main/default. This folder, called a package directory, contains all the metadata of your current Salesforce DX project. Because Lightning web components are also metadata, they are stored in a subfolder named lwc. In the next step, we add a Lightning web component to this folder.

We can use Visual Studio Code for creating a Lightning web component, just as we did to create the Salesforce DX project. Or we can use Salesforce CLI directly.

1.Open Visual Studio Code.

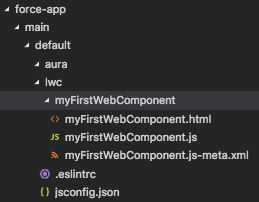
2.Press Command + Shift + P on macOS or Ctrl + Shift + P on Windows or Linux, then type focus terminal. Press Enter.

3.Enter

sfdx force:lightning:component:create -n myFirstWebComponent -d force-app/main/default/lwc --type lwc

, and confirm with Enter.

This creates the needed files for your first Lightning web component.



### Adding Code and Metadata to Your First Lightning Web Component

1. Open the new subfolder for **myFirstWebComponent**, which we just created in the lwc subfolder
2. Click **myFirstWebComponent.js-meta.xml**.
3. Replace the contents of the XML file with this XML markup

<?xml version="1.0" encoding="UTF-8"?>

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

<apiVersion>51.0</apiVersion>

<isExposed>true</isExposed>

<targets>

<target>lightning\_\_AppPage</target>

<target>lightning\_\_RecordPage</target>

<target>lightning\_\_HomePage</target>

</targets>

</LightningComponentBundle>

4. Press **CMD + S** on macOS, or **CTRL + S** on Windows or Linux, to save the file.

Next we’re updating the JavaScript file of your Lightning web component.

1. In Visual Studio Code click **myFirstWebComponent.js**.
2. Paste this

import { LightningElement } from 'lwc';

export default class MyFirstWebComponent extends LightningElement {

@track

contacts = [

{

Id: 1,

Name: 'Amy Taylor',

Title: 'VP of Engineering',

},

{

Id: 2,

Name: 'Michael Jones',

Title: 'VP of Sales',

},

{

Id: 3,

Name: 'Jennifer Wu',

Title: 'CEO',

},

];

}

3.Press CMD + S on macOS, or CTRL + S on Windows or Linux, to save the file.

After you save the file you’ll immediately notice a few things.

* The annotated word @track is underlined with a red squiggly line.
* The JavaScript file color in the explorer changed to red, and has a 2 next to it.

HTML markup

1. In Visual Studio Code click **myFirstWebComponent.html**.
2. Insert this markup within the existing <template></template> tags

<lightning-card title="ContactInformation" icon-name="custom:custom14">

<div class="slds-m-around\_medium">

<template for:each={} for:item="contact">

<div>

{contact.Name}, {contact.Title}

</div>

</template>

</div>

</lightning-card>

3.Press **CMD + S** on macOS, or **CTRL + S** on Windows or Linux, to save the file.

### Deploy and Configure Your New Lightning Web Component

1. Open **Visual Studio Code**.
2. Press **Command + Shift + P** on macOS or **Ctrl + Shift + P** on Windows or Linux, then type **focus terminal**. Press **Enter**.
3. Enter this command to deploy the metadata to your org:  
   sfdx force:source:push
4. Press **Enter**

### Day 10 :

### Topic: Lightning Web Components

### Why Lightning Web Components(LWC)?

AURA framework which was used for current Lightning Components was based on standards of 2014 but are outdated now and it was time for change because for the following reasons:

* Rendering could be optimized.
* Standard UI elements were scarce.
* Lacked modern constructs.
* Was not fit for modular apps.
* Web standards were updated.
* AURA Framework became skill and had its own learning curve.

Additionally, Lightning Web Components(LWC) can coexist and interoperate with Aura components.

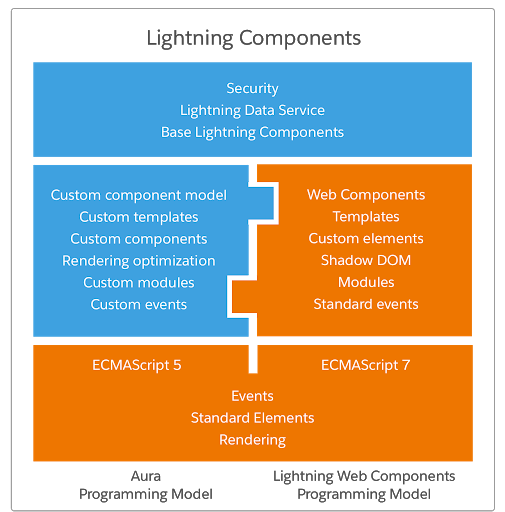
### What is Lightning Web Components(LWC)?

LWC is a new programming model levering the recent web standards. Rather than being a totally custom and development wise rigid framework, It’s quite flexible. It’s mostly the common Web Standards and a Thin Layer of Specialized services to make it a perfect fit for Modern Rich UI Implementations in Salesforce. This thin layer of specialized services contain Base Lightning Components, Lightning Data Service and User Interface API which work behind the curtain for LWC.

A thin layer of specialized services on top of a standard web stack results in:

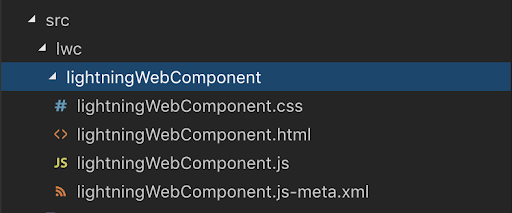
* Ease of development for large scale modular apps.
* Ease of Leveraging the latest web functionalities and constructs.
* A common model and transferable skills.  
  (Any web developer working on modern JS frameworks could easily ramp-up LWC).
* Interoperable components.
* Better performance.

So, the new development stack looks like:



### How is a Lightning Web Component(LWC) formed?

Similar to an AURA component, the main contents of a LWC are also html, javascript. There are optional content like css. But then in addition to these for LWC, an xml configuration file is also included which defines the metadata values for the component.



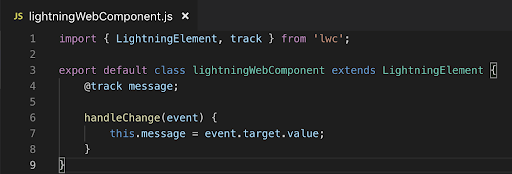
### HTML

* Has a root tag <template> which contains your component’s HTML.
* When renders, the <template> tag is replaced with <namespace-component-name>.



### Javascript

* Import functionality declared in a module eg-lwc(the core module), use the import statement.
* To allow other code to use functionality in a module, use the export statement.
* LightningElement is custom wrapper of the standard HTML element and we extend it in the component and export.



### Configuration

XML file that defines the metadata configuration values for the component eg-

* Components Label
* Availability
* Configuration Attributes
* Builder Attributes



### CSS

* To style a component.
* The style sheet is applied automatically



### Component UI

### 

### Day 11:

### Topic: Component Creation For Application

### Milestones: Created Components For Application

### 1.Create College DataTable Component( APEX CLASS)

we are going to create the college data table, for this we need to create an apex class, from which we are going to retrieve the data and Html, javascript files for UI .

public class GetapplicationDetails

{

@AuraEnabled(cacheable=true)

public static List<Application\_Form\_\_c> getapplicationvalues(id CollegeId)

{

List<Application\_Form\_\_c> formlist = [SELECT ID, College\_Fee\_\_c,Name,

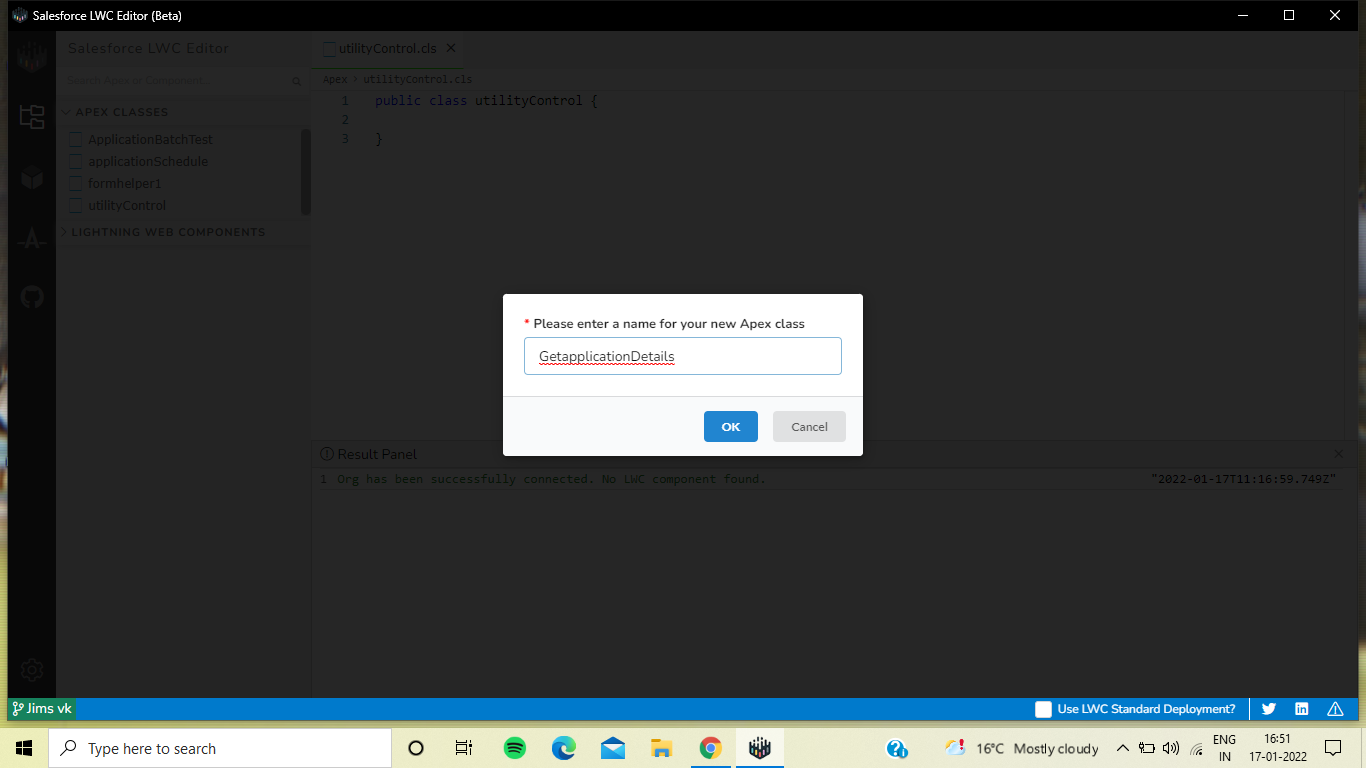
Date\_Of\_Birth\_\_c,Email\_\_c,Hostel\_Fee\_\_c, Student\_Name\_\_c FROM

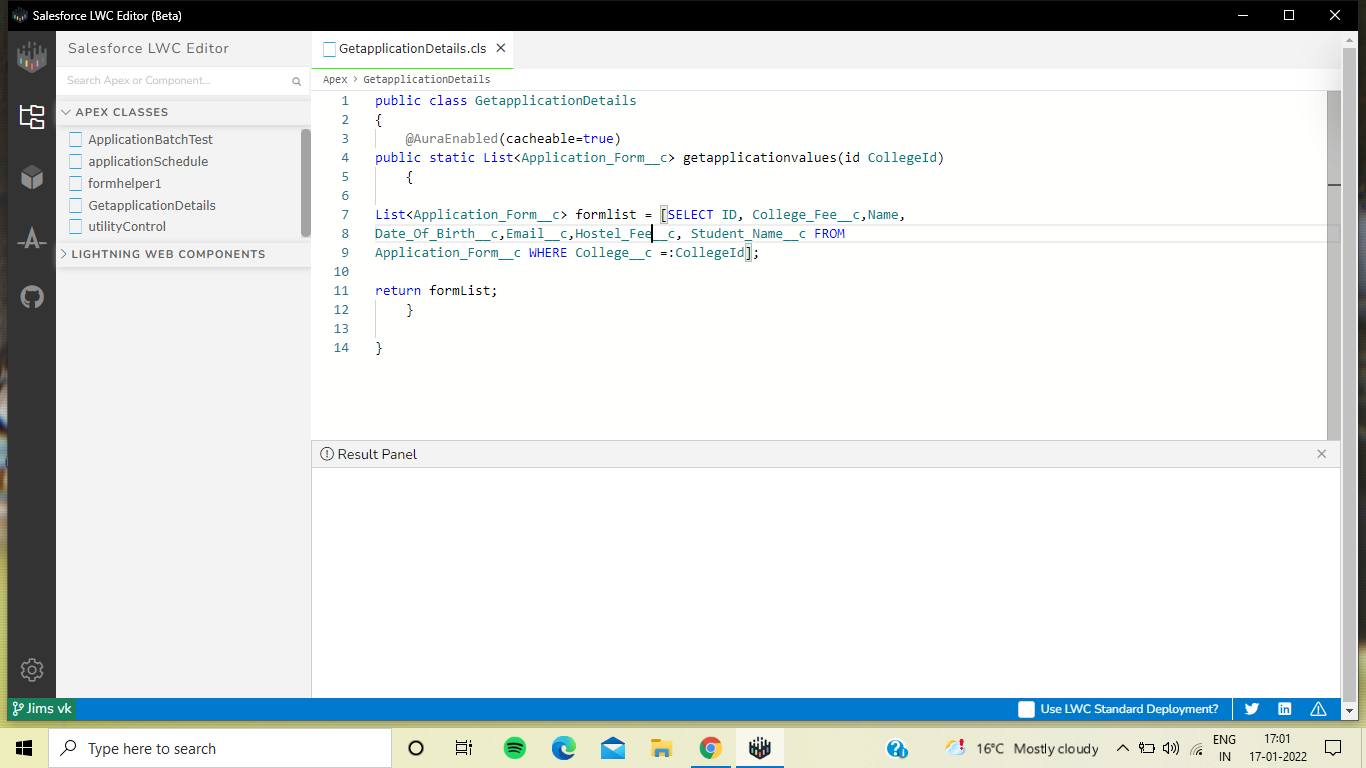
Application\_Form\_\_c WHERE College\_\_c =:CollegeId];

return formList;

}

}





### Create College DataTable Component (HTML FILE)

<template>

<h1> College and Application form list table </h1>

<template if:true={recordList}>

<lightning-datatable

key-field="id"

data={recordList}

show-row-number-column

hide-checkbox-column

columns={columnsList}

>

</lightning-datatable>

</template>

<template if:true={error}>

{error}

</template>

</template>



### Create College DataTable Component(JAVA SCRIPT FILE)

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

import getapplicationvalues

from'@salesforce/apex/GetapplicationDetails.getapplicationvalues';

export default class CollegeDataTable extends LightningElement {

columnsList = [

{label : 'Application Form' , fieldName : 'Name', type:'text' },

{label : 'College Fee' , fieldName : 'College\_Fee\_\_c', type:'currency' },

{label : 'Date Of Birth' , fieldName : 'Date\_Of\_Birth\_\_c', type:'date' },

{label : 'Email' , fieldName : 'Email\_\_c', type:'email' },

{label : 'Hostel Fee' , fieldName : 'Hostel\_Fee\_\_c', type:'Currency' },

{label : 'StudentName' , fieldName : 'Student\_Name\_\_c', type:'text' }

];

@api recordId;

recordList;

error;

@wire(getapplicationvalues, {CollegeId : '$recordId'})

wiredCollegeData({data, error}){

if(data){

this.recordList = data;

}

else if(error){

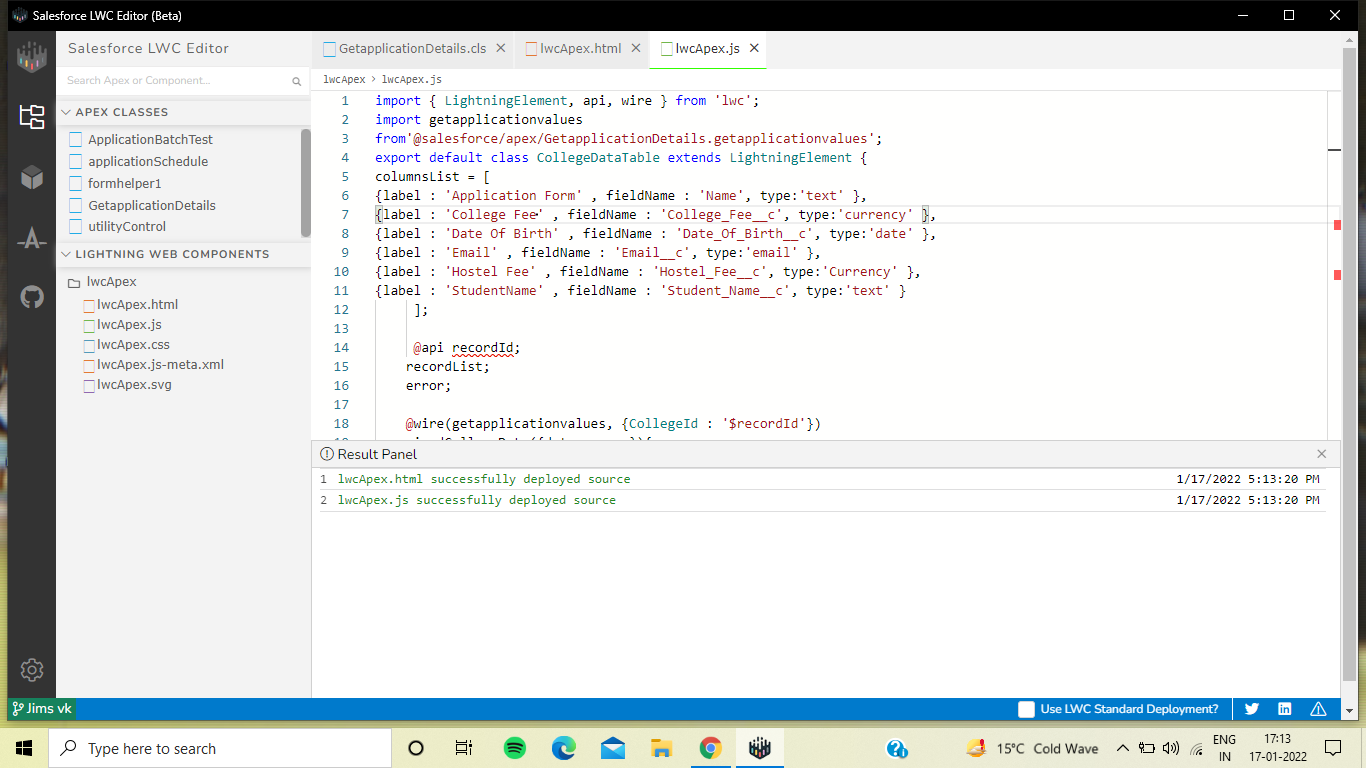
this.error = error;

this.recordList = undefined;

}

}

}



### Create College DataTable Component(META FILE)

<?xml version="1.0"?>

<LightningComponentBundle

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

<apiVersion>51.0</apiVersion>

<isExposed>true</isExposed>

<targets>

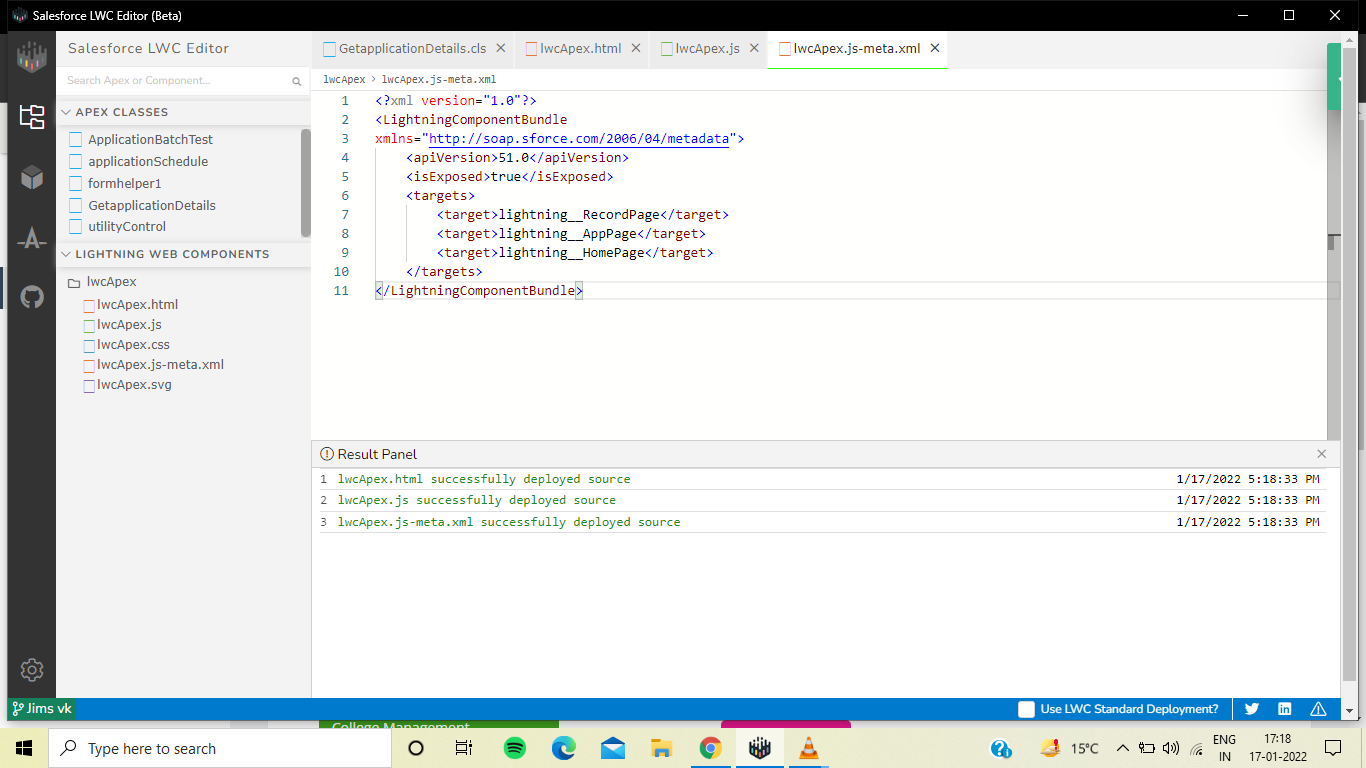
<target>lightning\_\_RecordPage</target>

<target>lightning\_\_AppPage</target>

<target>lightning\_\_HomePage</target>

</targets>

</LightningComponentBundle>



### 

### Day 12:

**Topic: Lifecycle Hooks**

A lifecycle hook is a callback method triggered at a specific phase of a component instance’s lifecycle.

**constructor()**

Called when the component is created. This hook flows from parent to child, which means that it fires in the parent first. You can’t access child elements because they don’t exist yet. Properties aren’t passed yet, either. Properties are assigned to the component after construction and before the connectedCallback( )

import { LightningElement } from 'lwc'

export default class HelloWorld extends LightningElement{

constructor(){

super();

}

}

**connectedCallback()**

Called when the element is inserted into a document. This hook flows from parent to child. You can’t access child elements because they don’t exist yet.

Use connectedCallback() to interact with a component's environment. For example, use it to:

Establish communication with the current document or container and coordinate behavior with the environment.

Perform initialization tasks, such as fetch data, set up caches, or listen for events

The connectedCallback() hook is invoked with the initial properties passed to the component. If a component derives its internal state from the properties, it's better to write the logic in a setter than in connectedCallback().

The connectedCallback() hook can fire more than one time. For example, if you remove an element and then insert it into another position, such as when you reorder a list, the hook fires several times. If you want code to run one time, write code to prevent it from running twice.

import { LightningElement } from 'lwc'

export default class HelloWorld extends LightningElement{

connectedCallback(){

console.log('CONNECTED CALLBACK');

}

}

**renderedCallback()**

Called after every render of the component. This lifecycle hook is specific to Lightning Web Components, it isn’t from the HTML custom elements specification. This hook flows from child to parent.

A component is rerendered when the value of a property changes and that property is used either directly in a component template or indirectly in the getter of a property that is used in a template.

If you use renderedCallback() to perform a one-time operation, you must track it manually (using an initialRender private property, for example). If you perform changes to reactive attributes, guard them or they can trigger wasteful rerenders or an infinite rendering loop.

Use renderedCallback() to interact with a component's UI. For example, use it to:

Compute node sizing

Perform tasks not covered by our template declarative syntax, such as add a listener for a non-standard event from a component’s child

Updating the state of your component in renderedCallback() can cause an infinite loop. For example:

Don’t update a wire adapter configuration object property.

Don’t update a public property or field in renderedCallback()

import { LightningElement } from 'lwc'

export default class HelloWorld extends LightningElement{

renderedCallback(){

console.log('RENDERED CALLBACK');

}

}.

**render**()

Call this method to update the UI. It may be called before or after connectedCallback().

It’s rare to call render() in a component. The main use case is to conditionally render a template. Define business logic to decide which template (HTML file) to use. The method must return a valid HTML template.

For example, imagine that you have a component that can be rendered in two different ways but you don’t want to mix the HTML in one file. Create multiple HTML files in the component bundle. Import them both and add a condition in the render() method to return the correct template depending on the component’s state.

**disconnectedCallback()**

Called when the element is removed from a document. This hook flows from parent to child.

Use disconnectedCallback() to clean up work done in the connectedCallback(), like purging caches or removing event listeners.

import { LightningElement } from 'lwc'

export default class HelloWorld extends LightningElement{

disconnectedCallback(){

console.log('DISCONNECTED CALLBACK');

}

}

**errorCallback(error, stack)**

Called when a descendant component throws an error. The error argument is a JavaScript native error object, and the stack argument is a string. This lifecycle hook is specific to Lightning Web Components, it isn’t from the HTML custom elements specification.

Implement this hook to create an error boundary component that captures errors in all the descendent components in its tree. Use the error boundary component’s errorCallback()lifecycle hook to log stack information and render an alternative view to tell users what happened and what to do next. The method works like a JavaScript catch{} block for components that throw errors in their lifecycle hooks or in their event handlers declared in an HTML template. It’s important to note that an error boundary component catches errors only from its children, and not from itself.

import { LightningElement } from 'lwc'

export default class HelloWorld extends LightningElement{

errorCallback(){

console.log('ERROR CALLBACK');

}

}