### College Management Application

**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

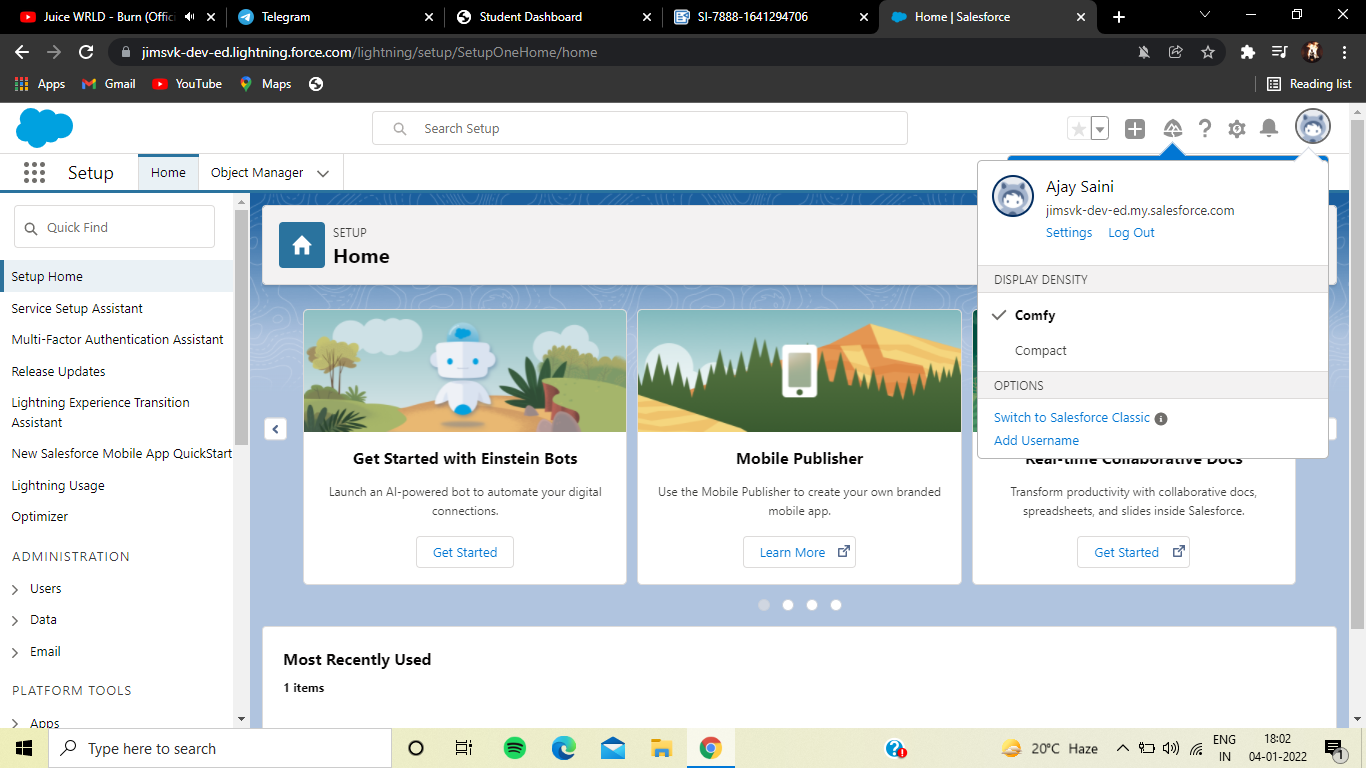
f.Postal Code : pin code

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 :

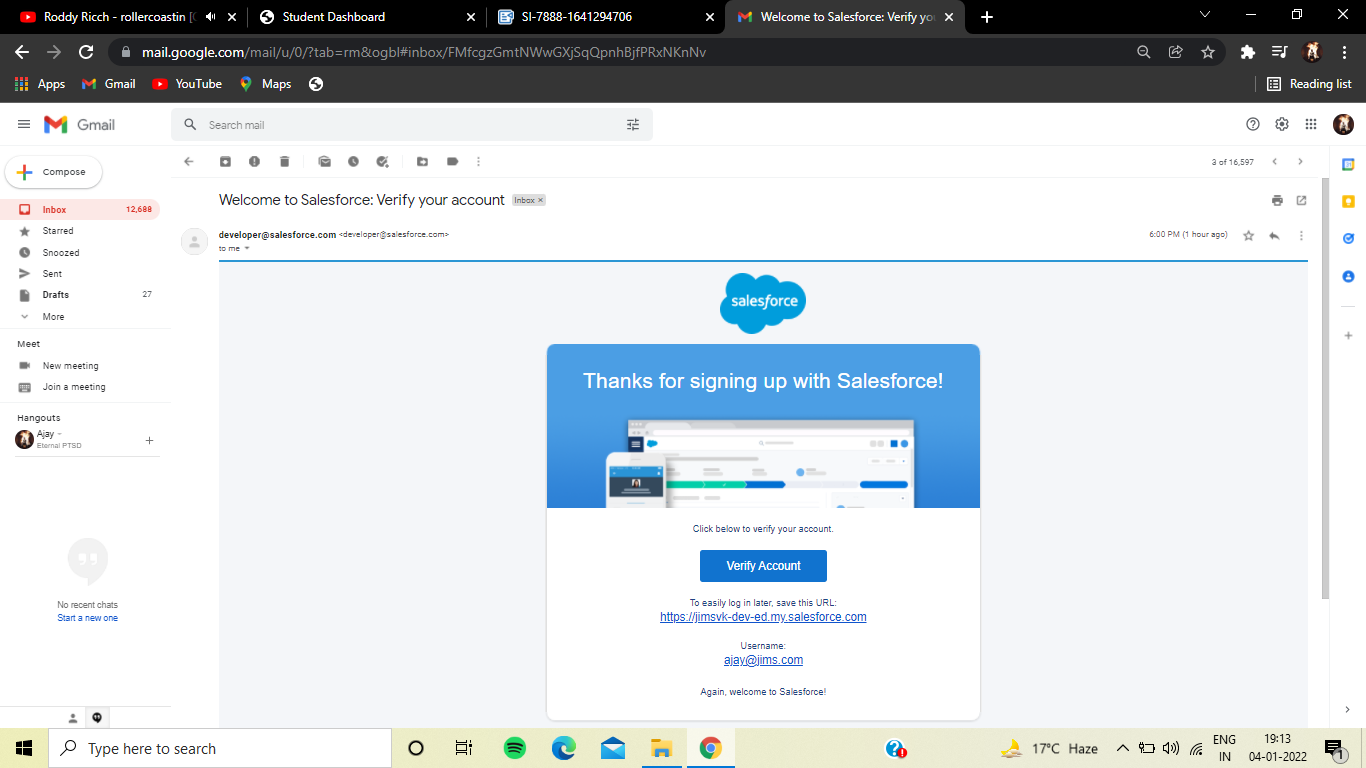
username@organization.com

Click on sign up



### **Account Activation**

### Activation Email

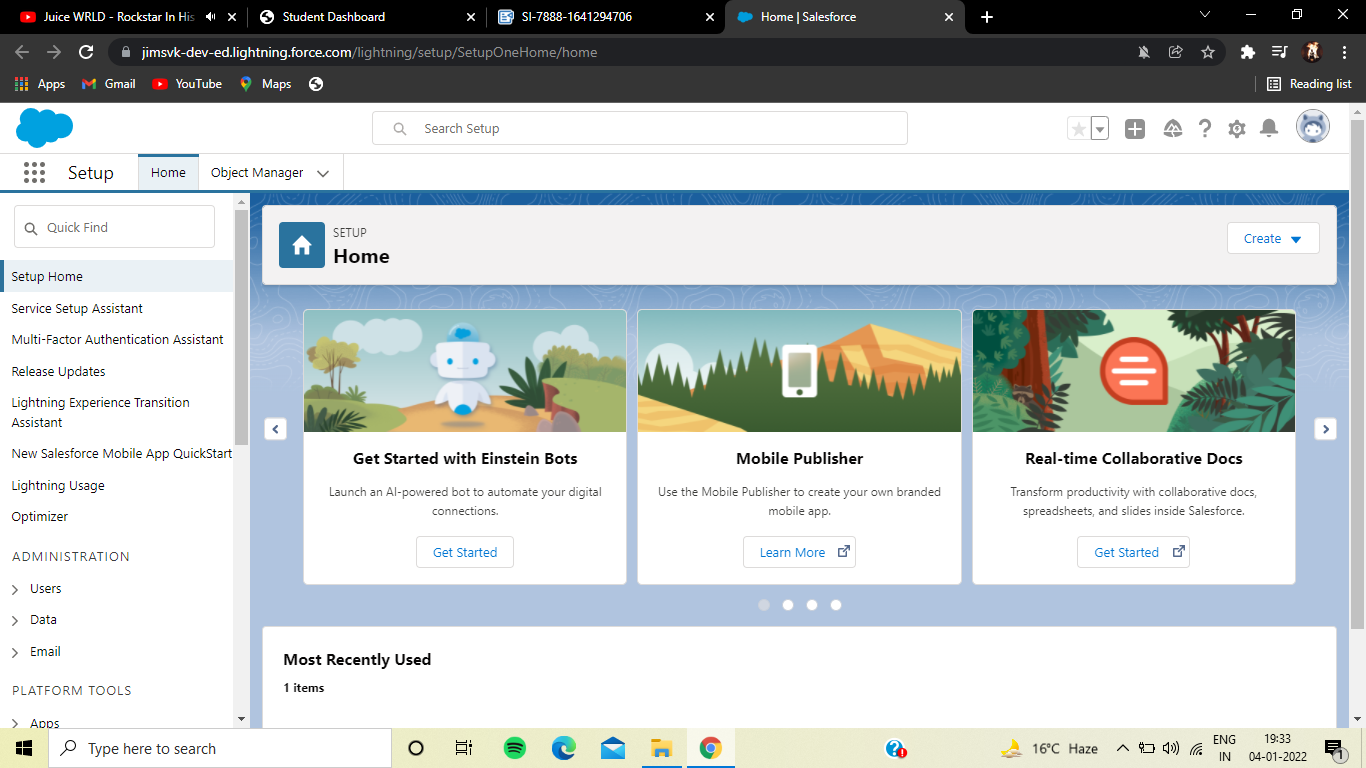


### **Login Into Salesforce Account**

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

2.Enter the username and password.

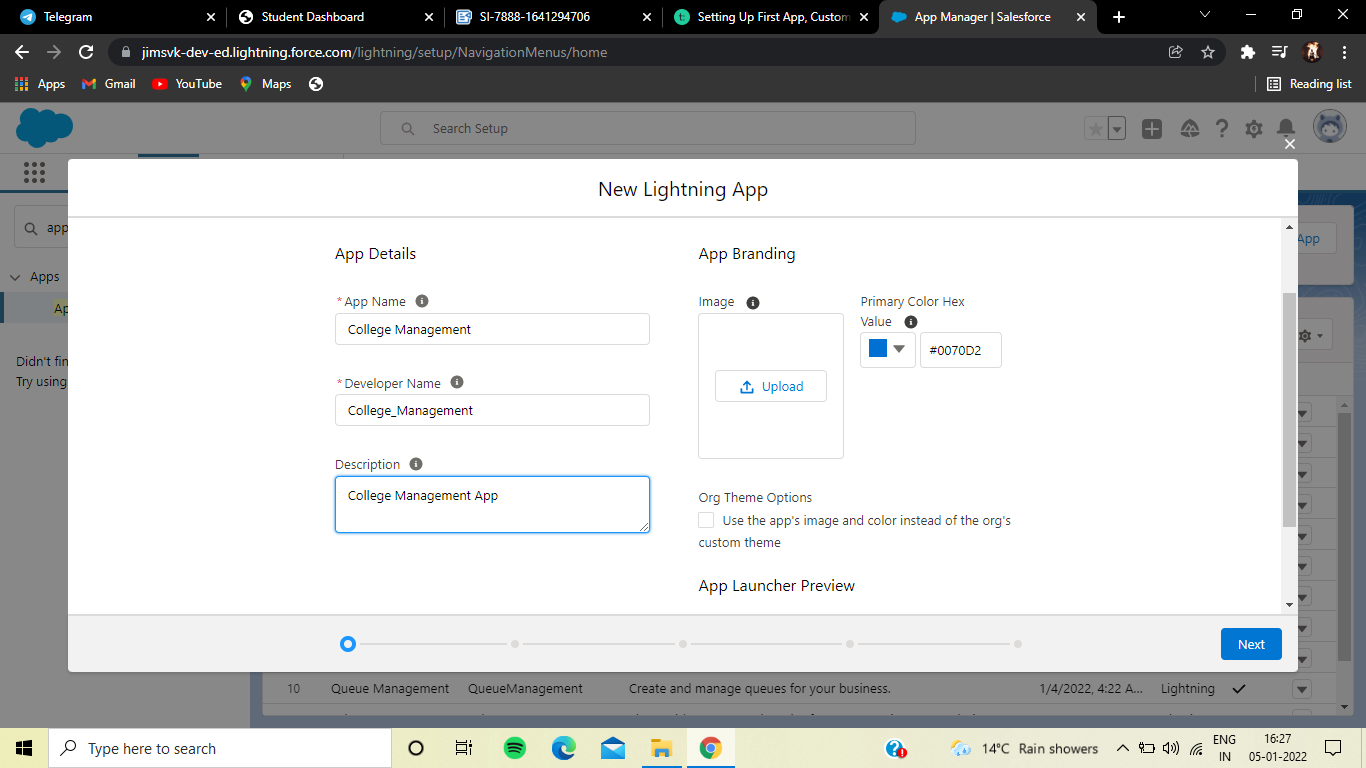
3.After login you'll see the home page.



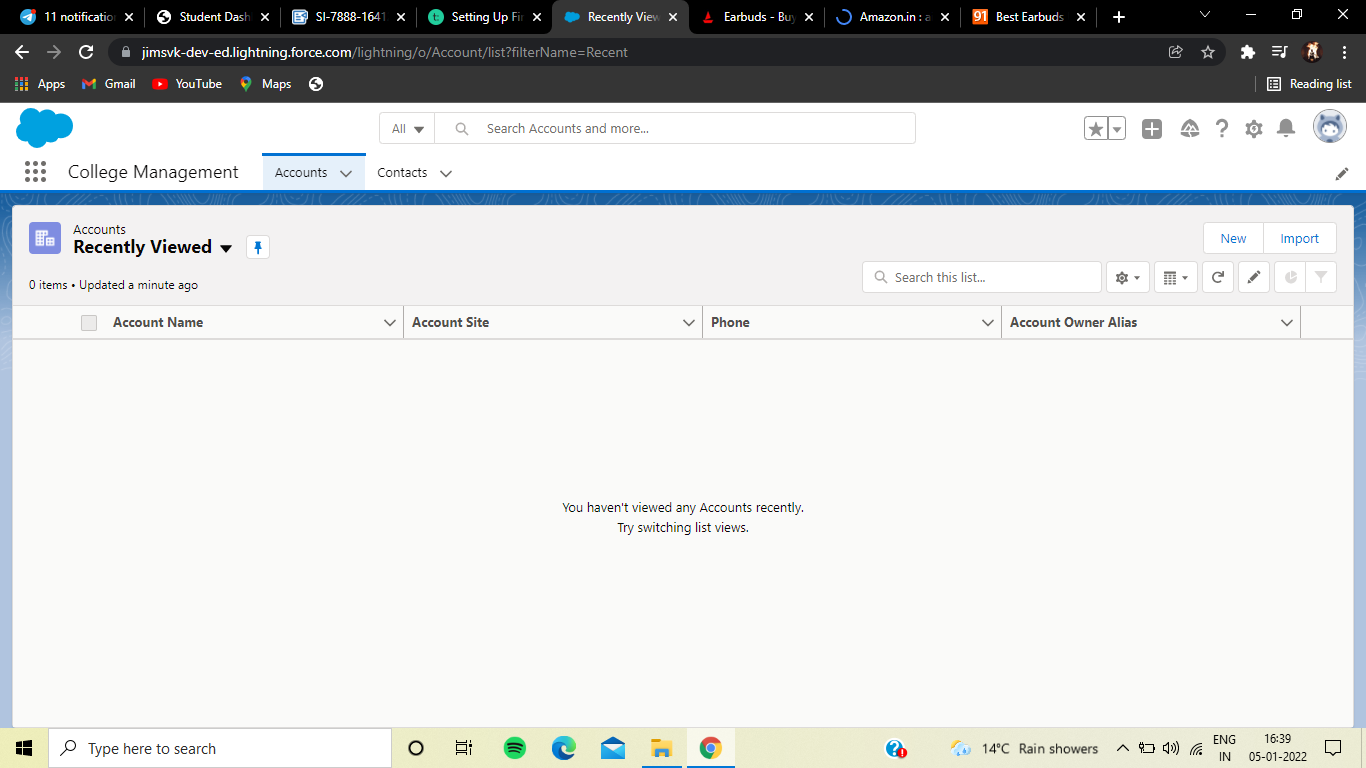
**Day 2 :**

## 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

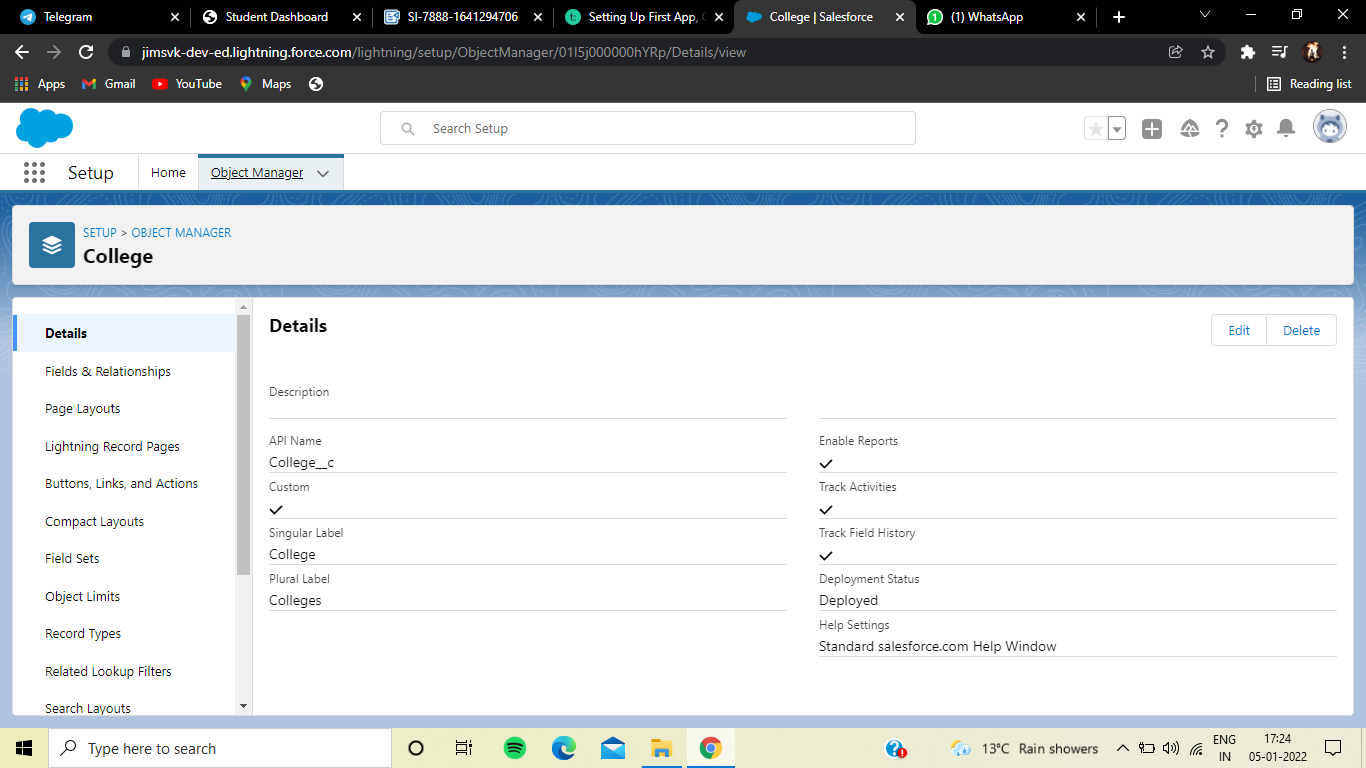


### **Milestone:** Creating The Custom Objects For The Application

**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

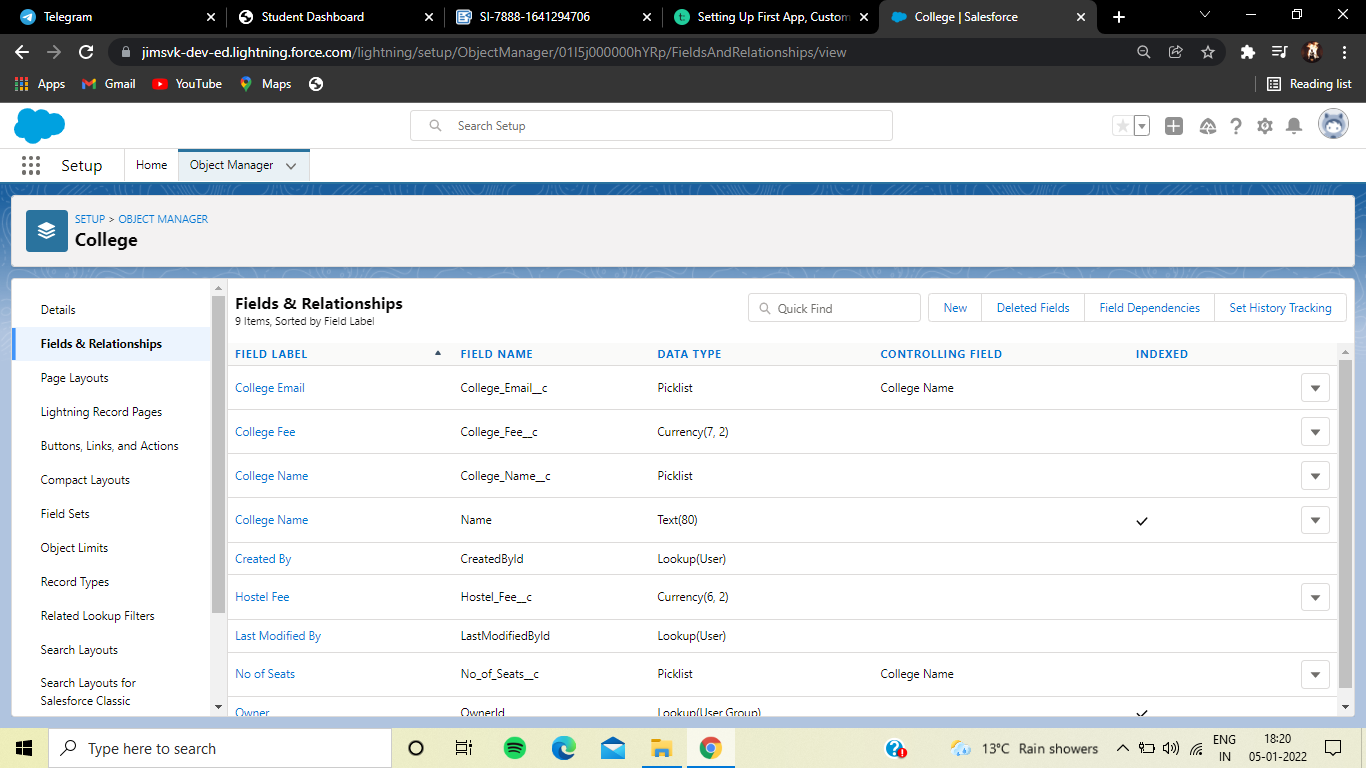
No of Seats Picklist

500- 1000, 1000-2500,

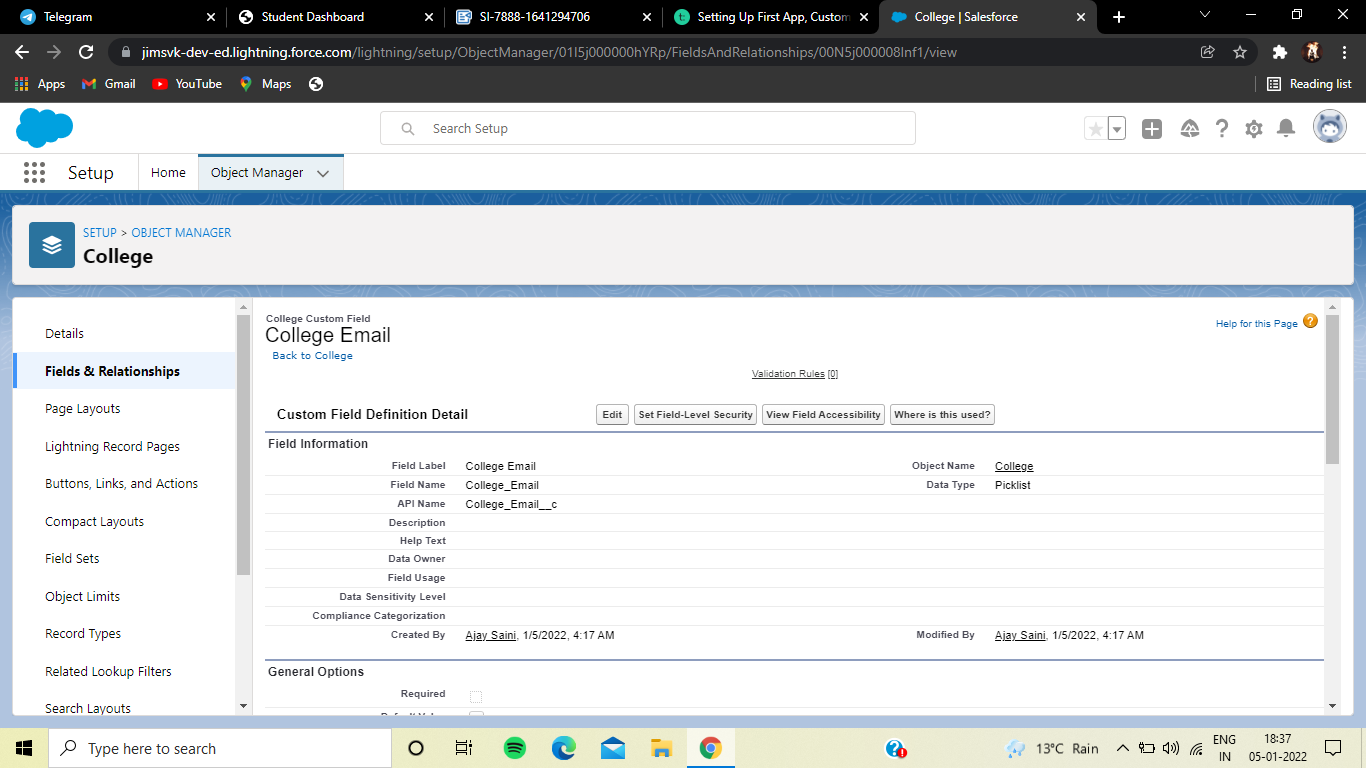
2000-6000, 6000 - 10000

College Fee Currency Yes

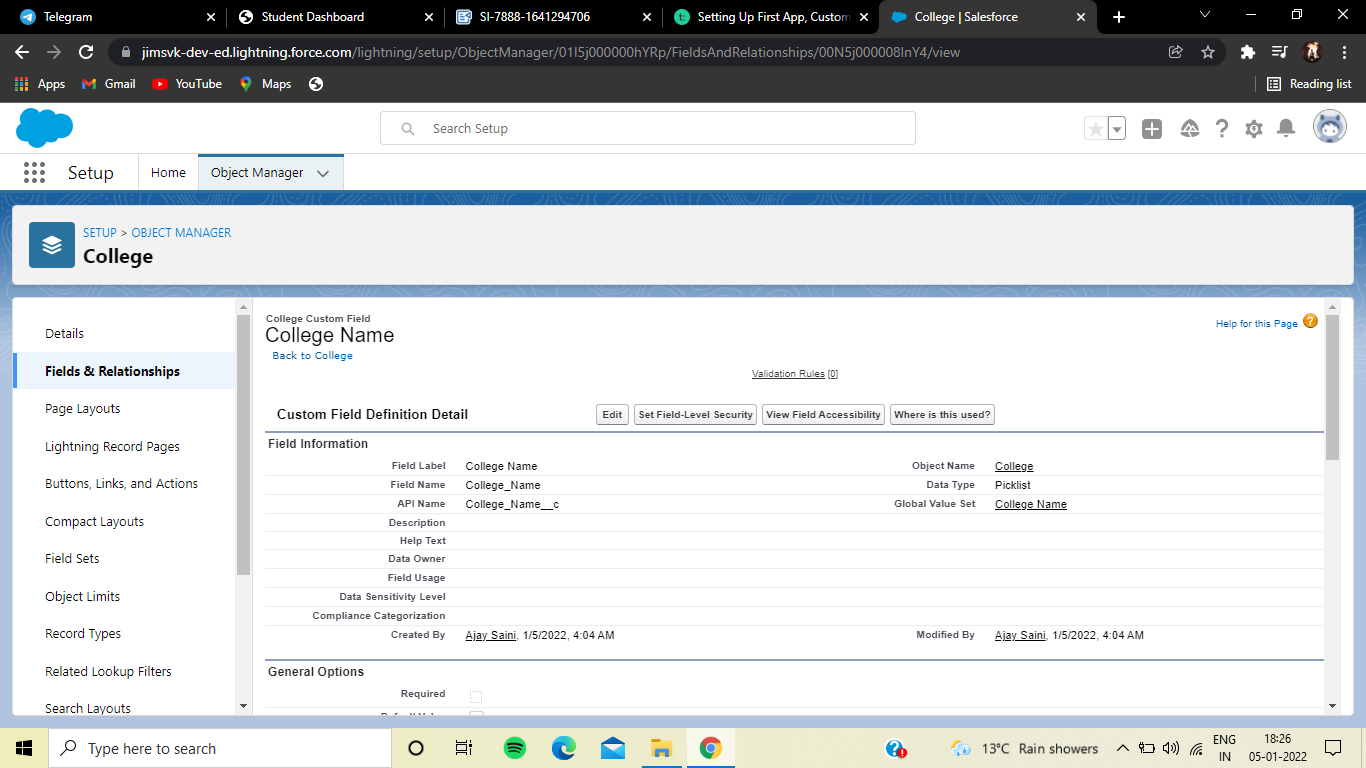
Hostel Fee Currency Yes



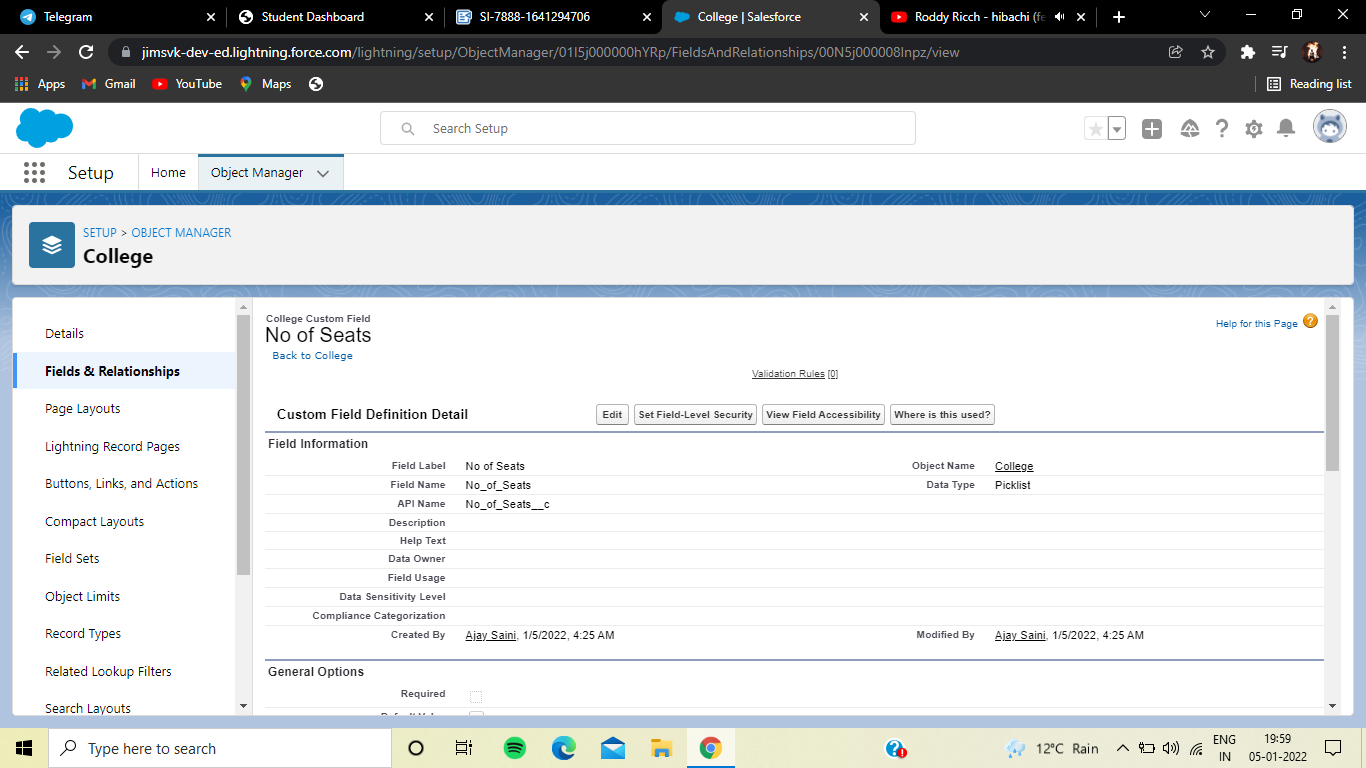
College Email :



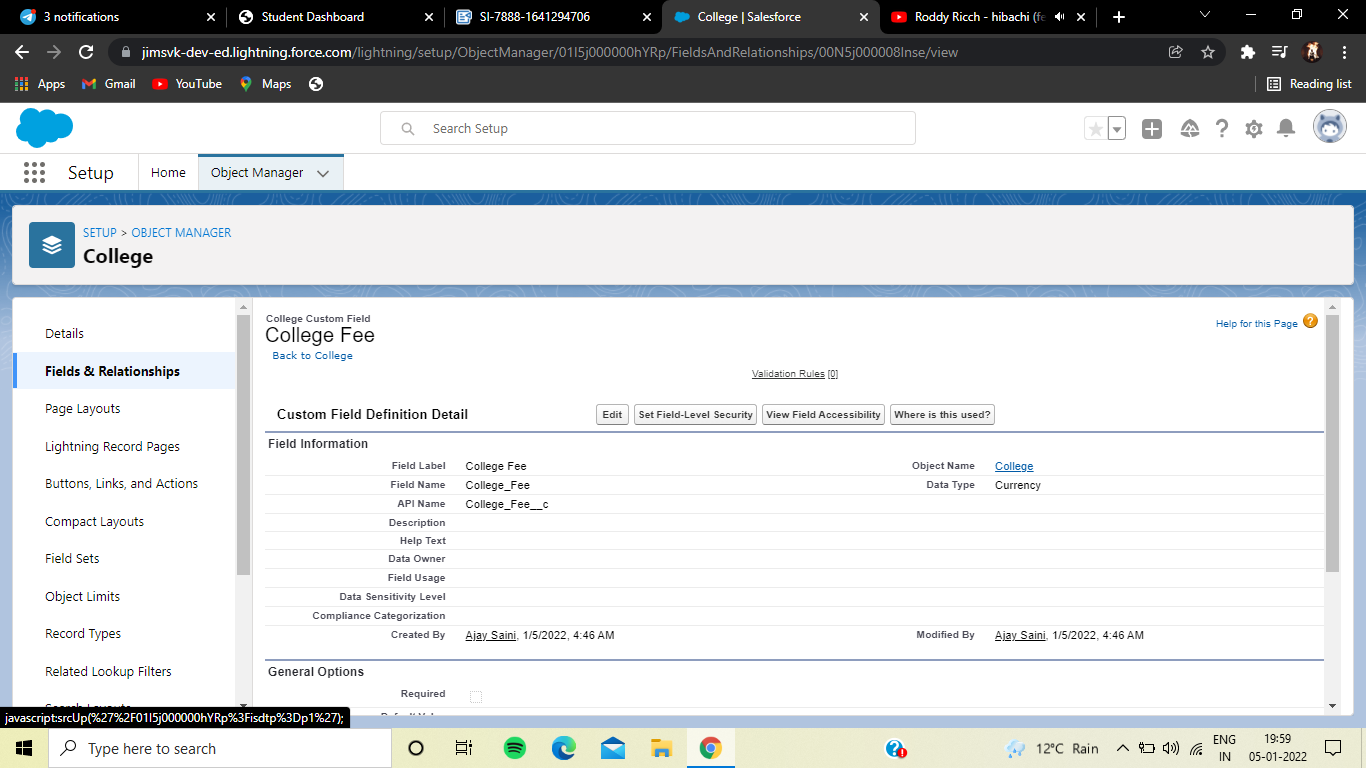
College Name:



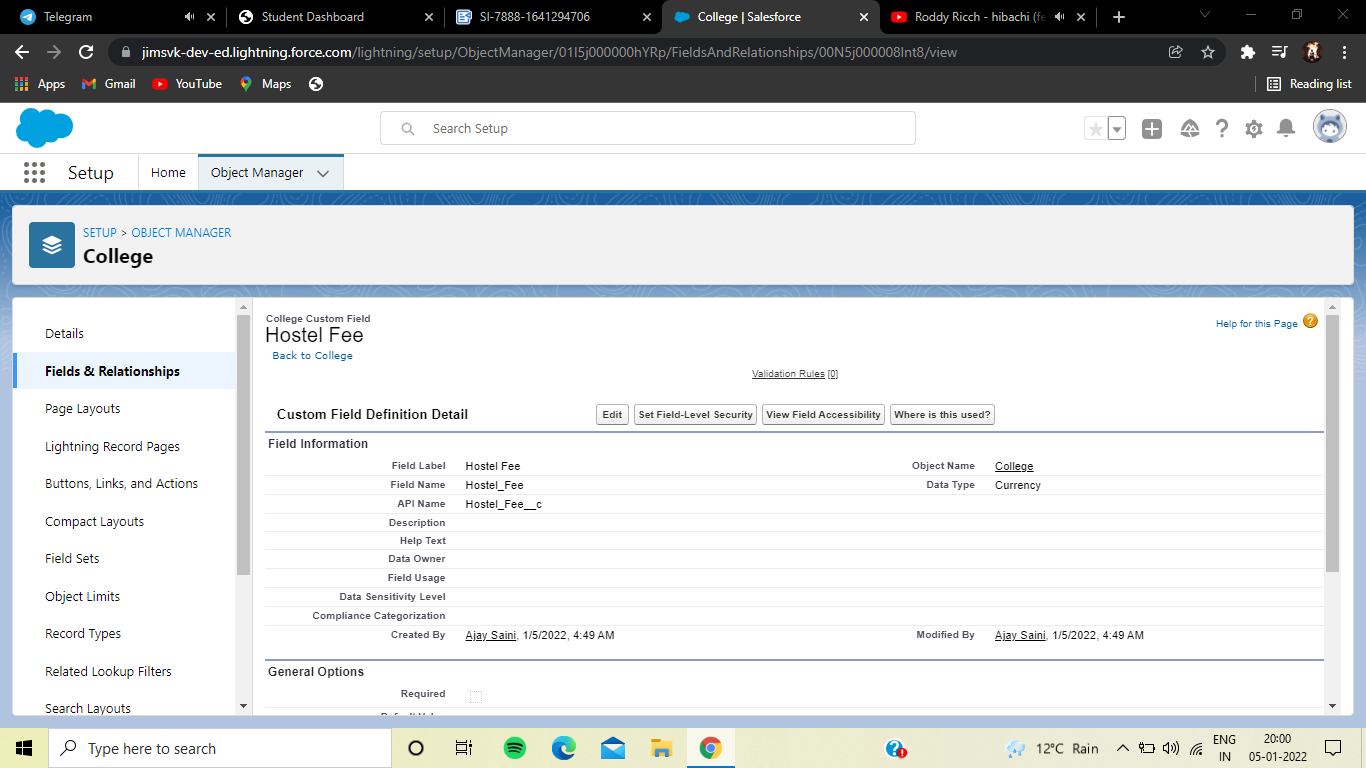
No of Seats:



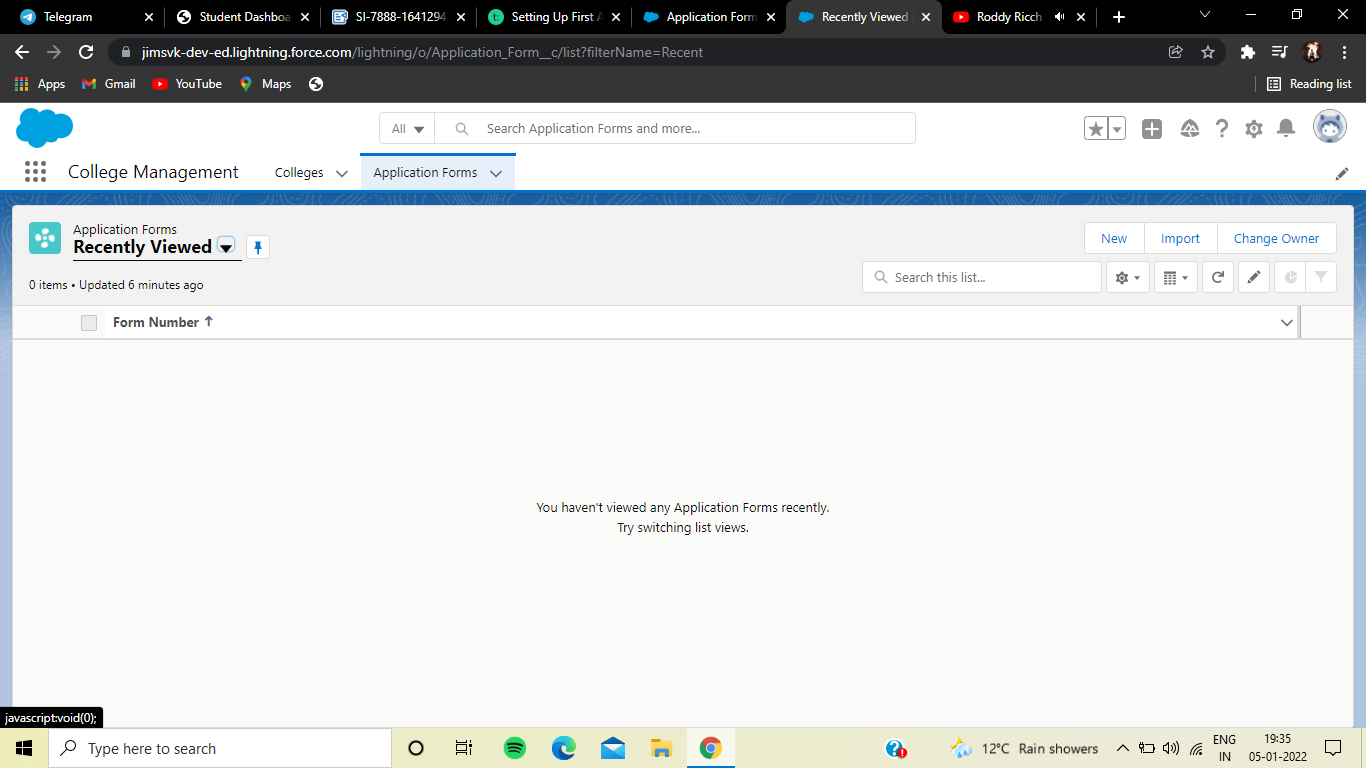
College Fee:

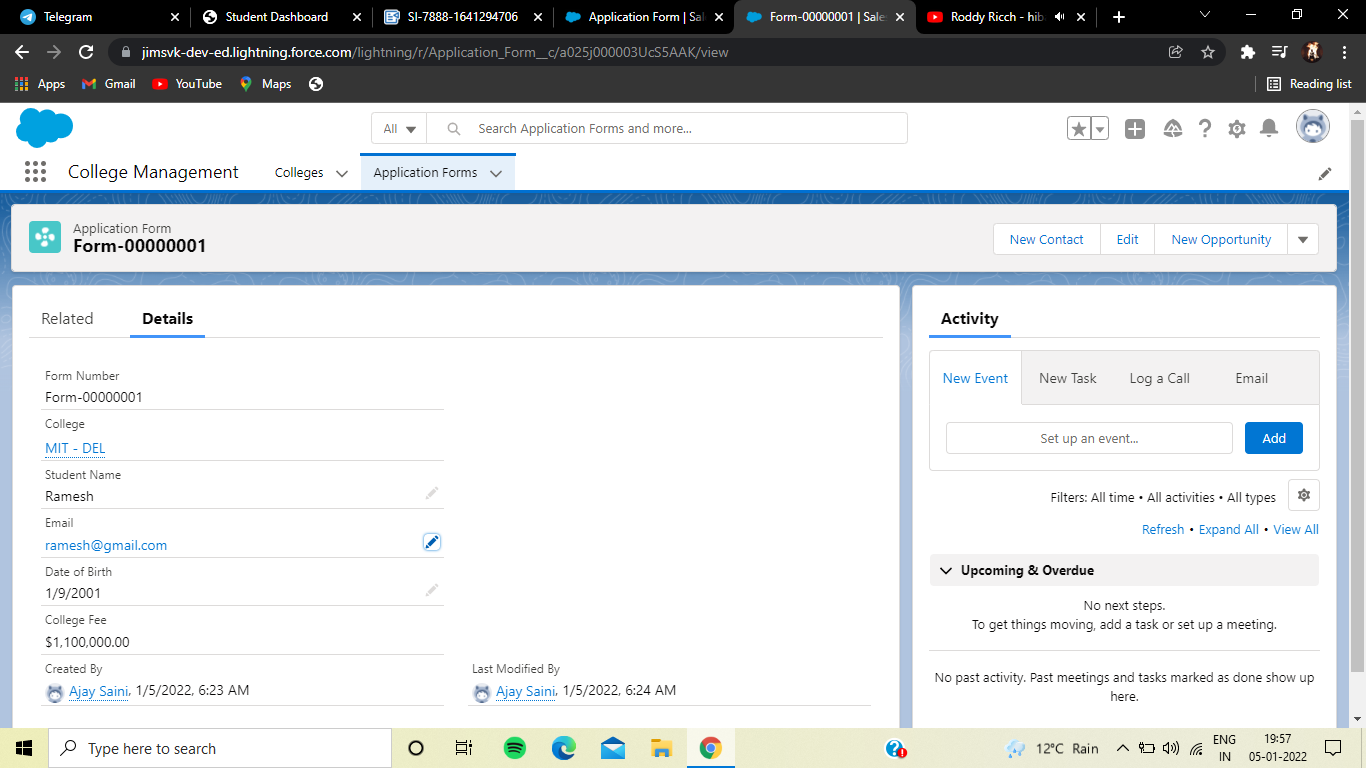


Hostel Fee:



**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

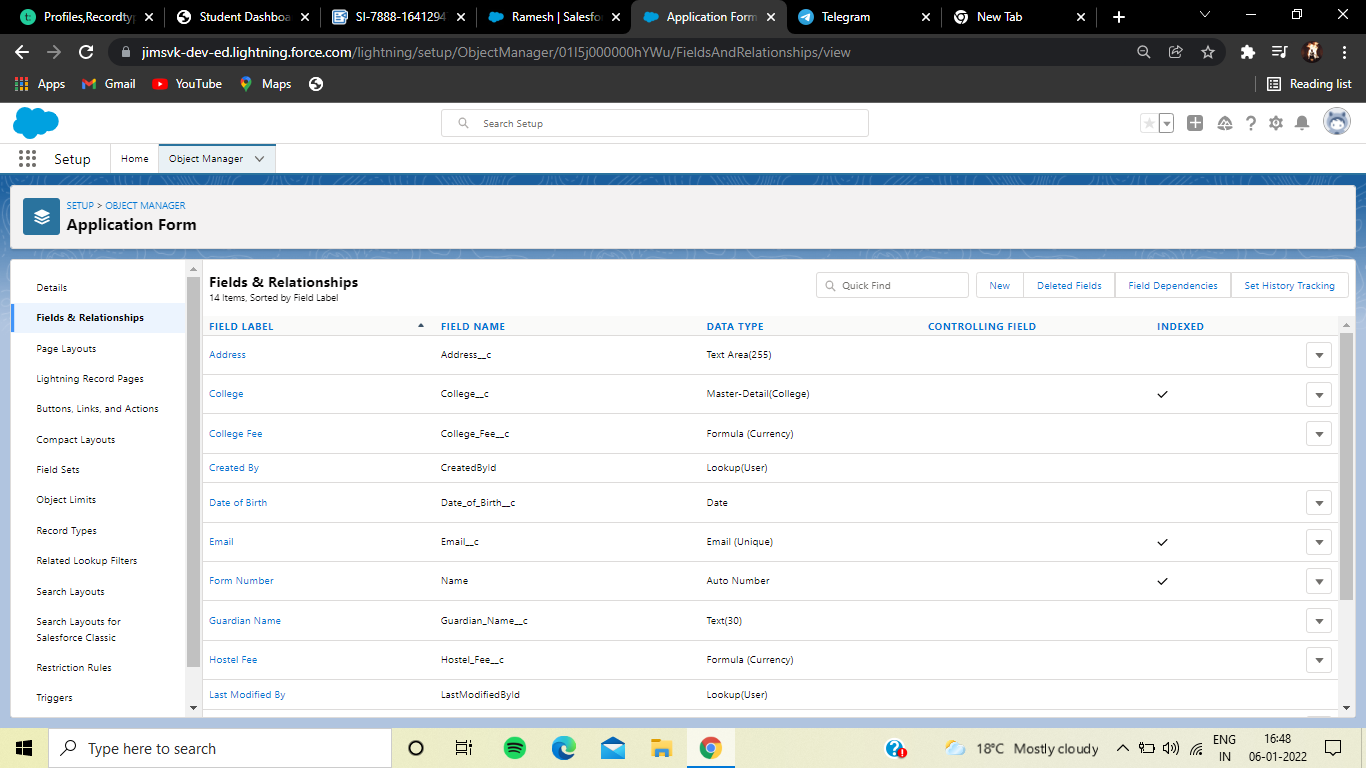
Stay Checkbox(default=Uncheck)

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

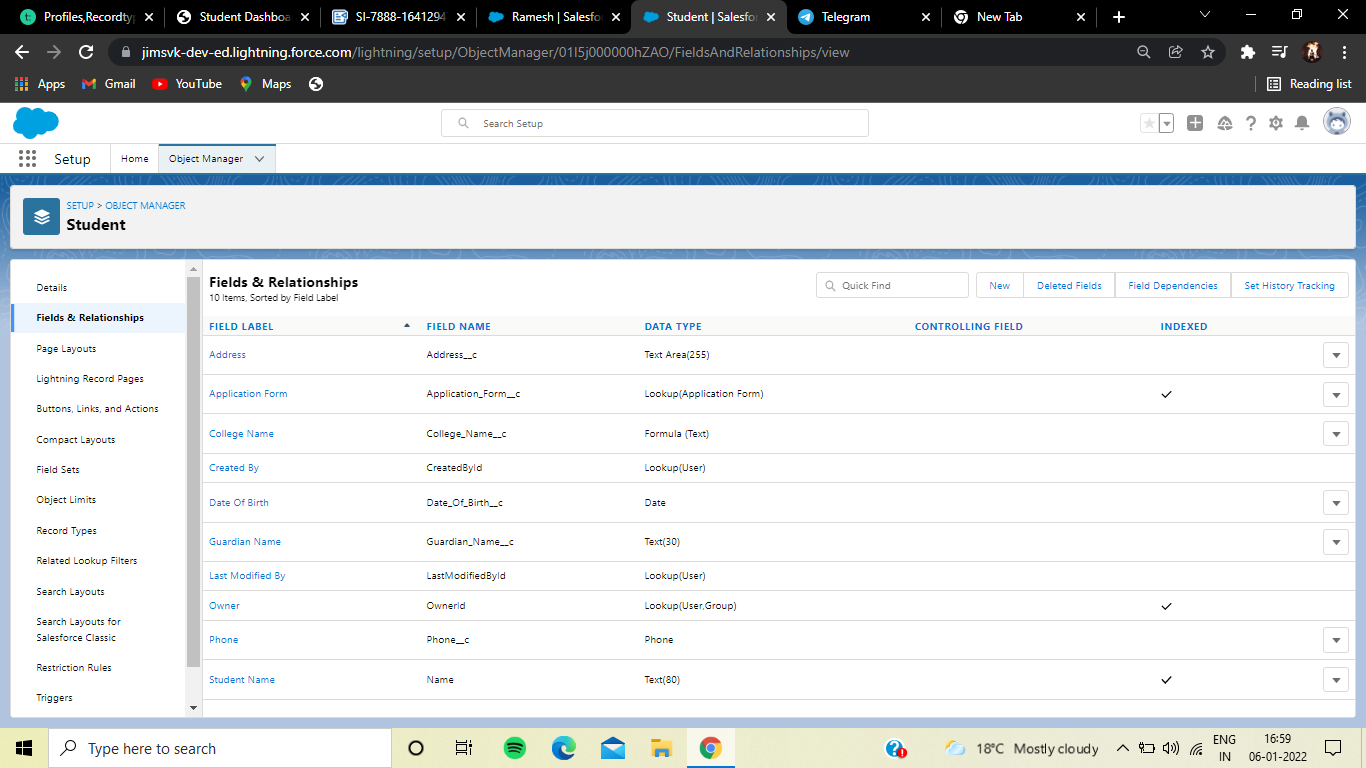
Date Of Birth Date Yes

College Name Formula(Text)

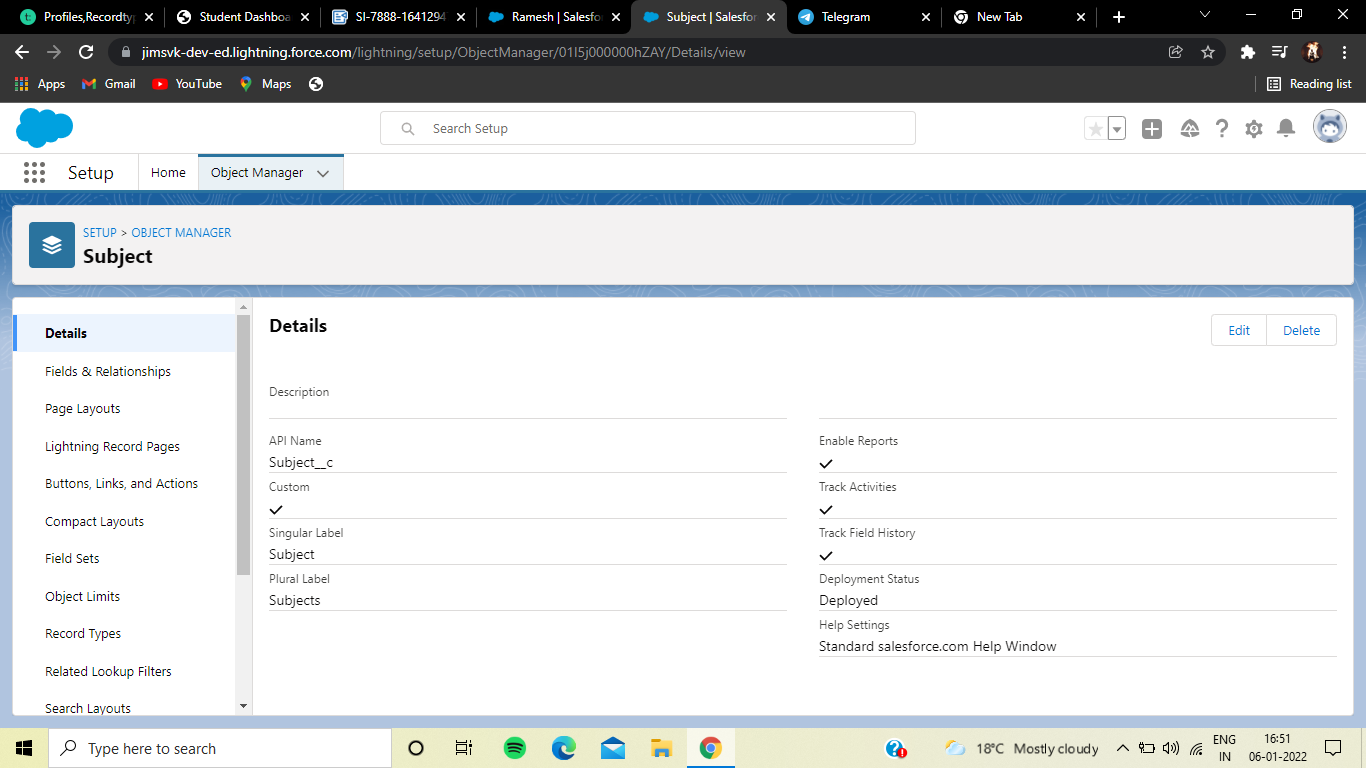
Application Form Lookup(Application Form)

Address Text(255) Yes

Student Name



**4. Subject**



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

Field Name Data Type Required Value

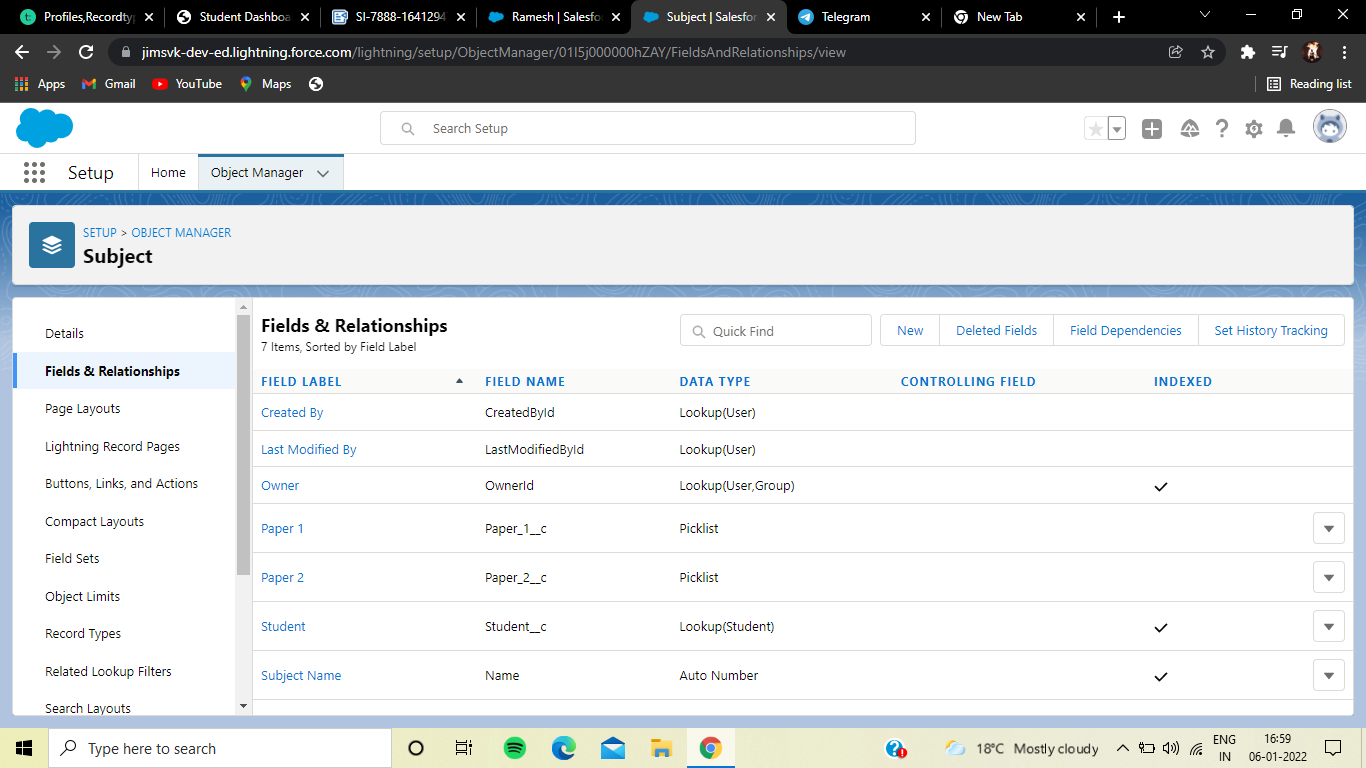
Student Lookup(Student)

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

Colllege 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

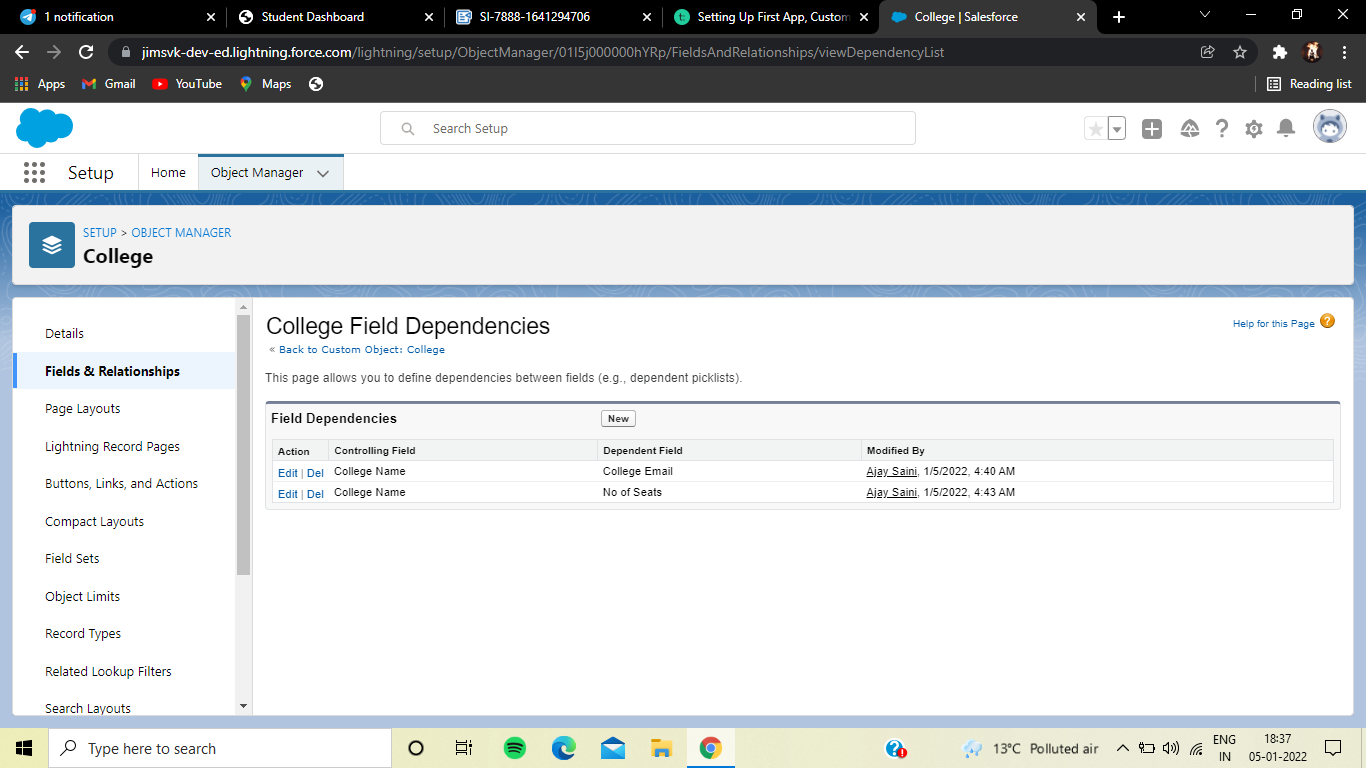
ENGLISH

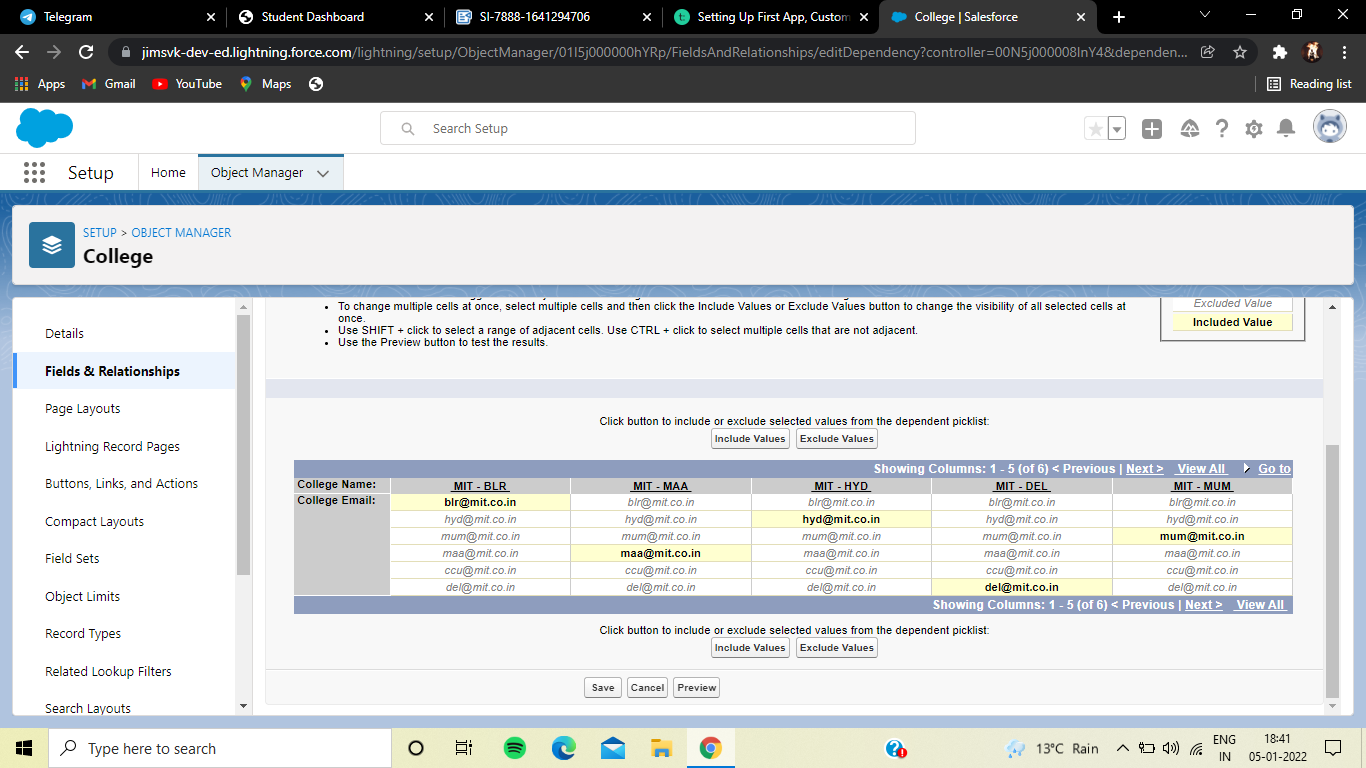
STATISTICS



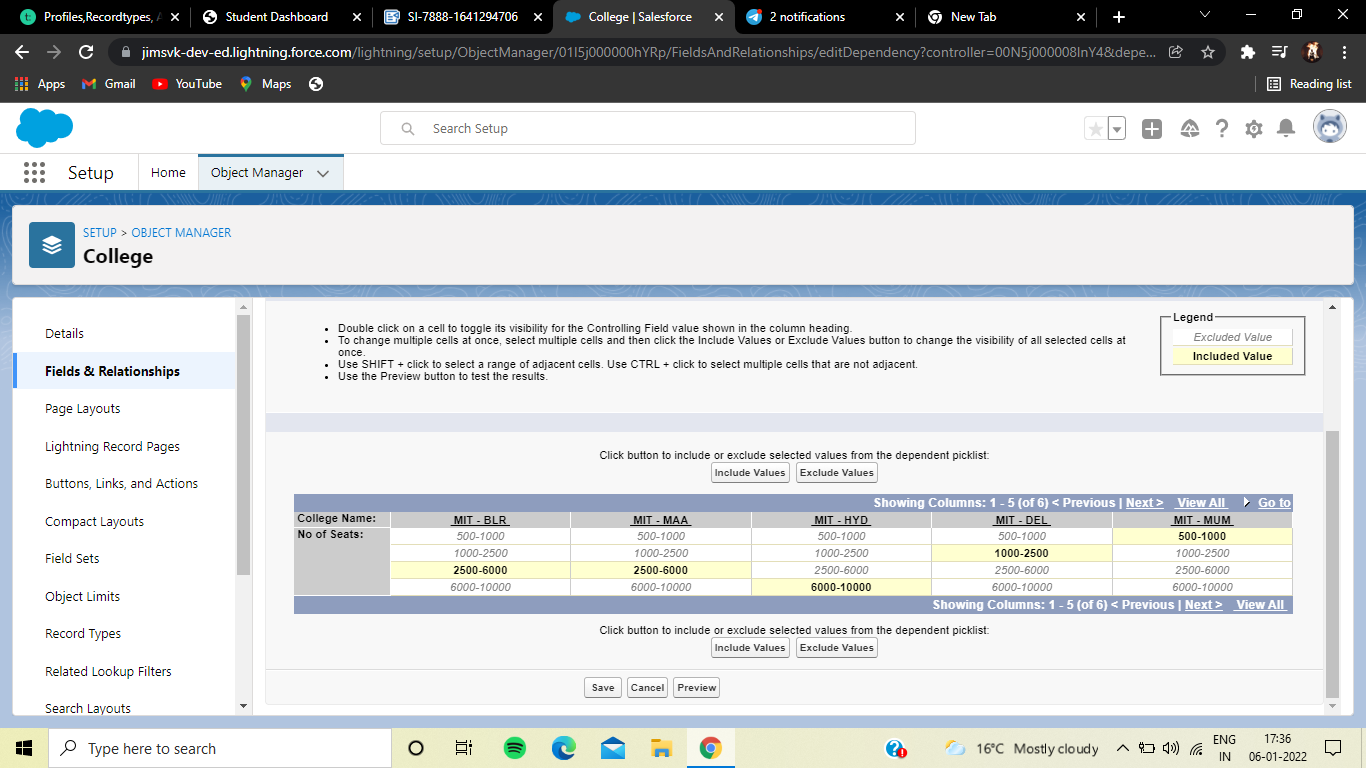
### Creating Field Dependencies

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.





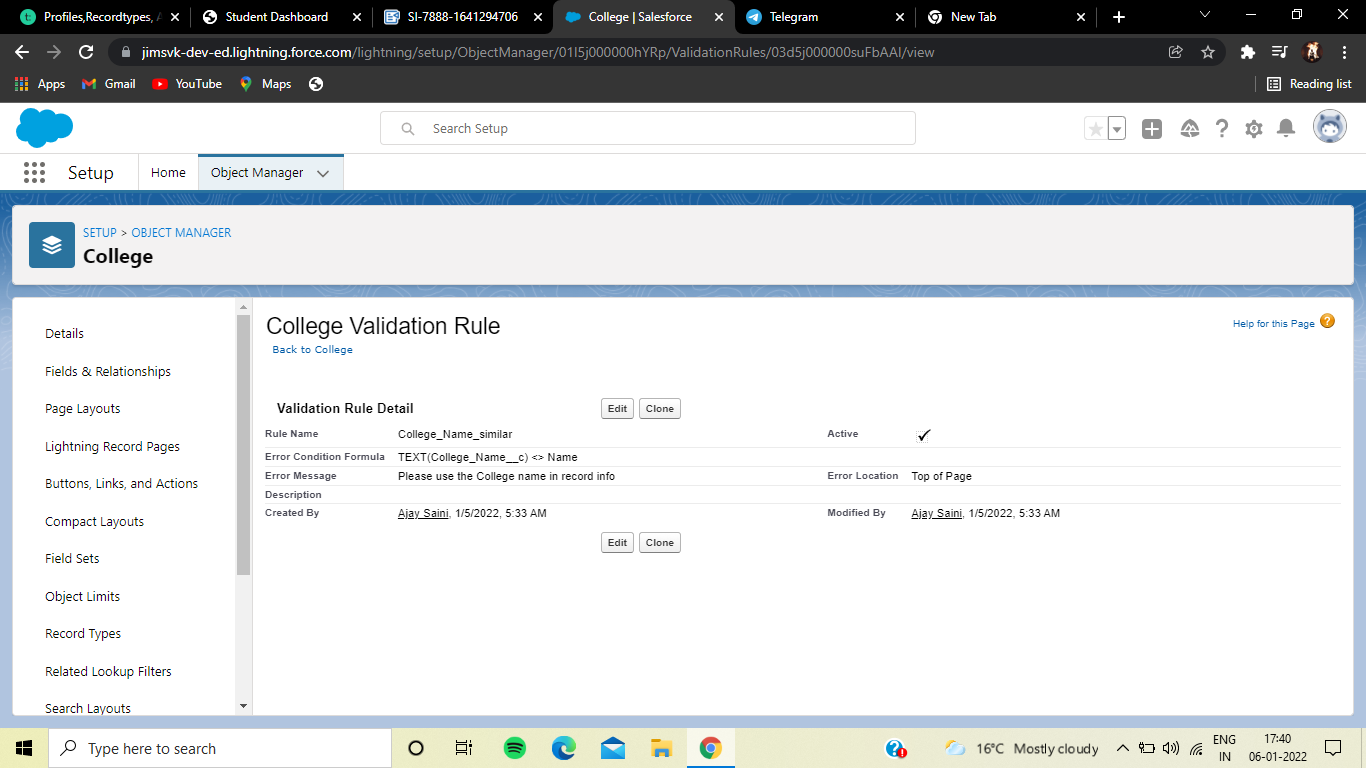
Created field dependency between college Name and capacity of students, where the controlling field is college Name and dependent field is Capacity of Students.



### 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 ) ,

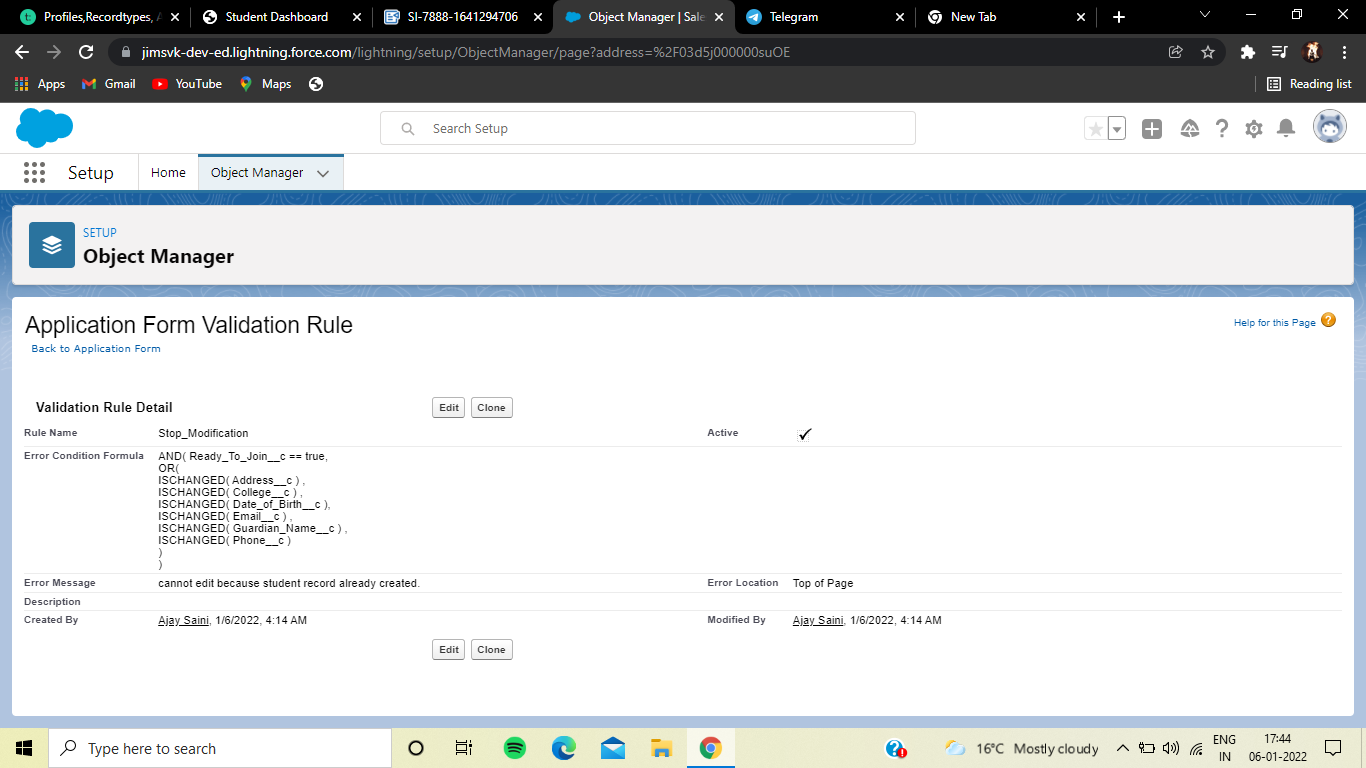
ISCHANGED( College\_\_c ) ,

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

ISCHANGED( Email\_\_c ) ,

ISCHANGED( Guardian\_Name\_\_c ) ,

ISCHANGED( Phone\_\_c ) ) )



**Day 3 :**

### Topic: Profiles, Recordtypes, Automation

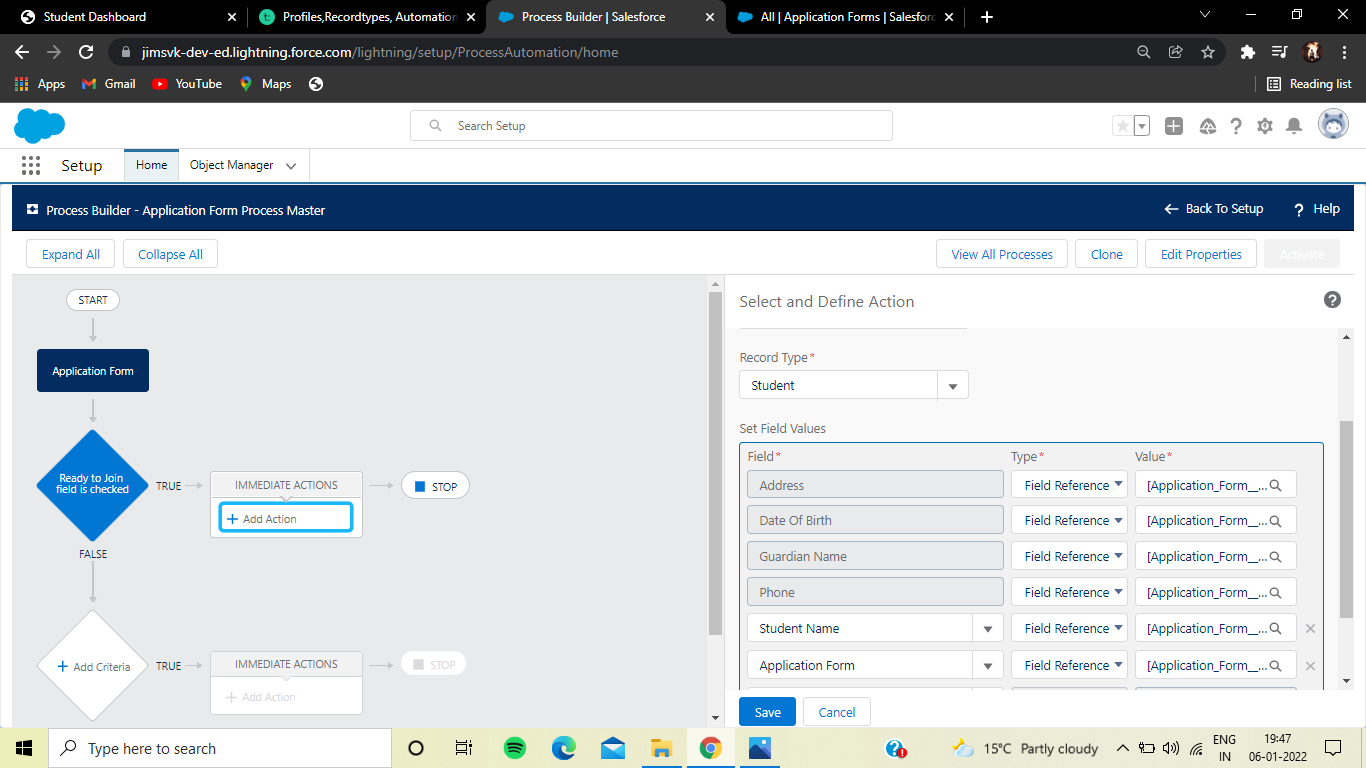
### Milestones: Creating an automation process

### 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:

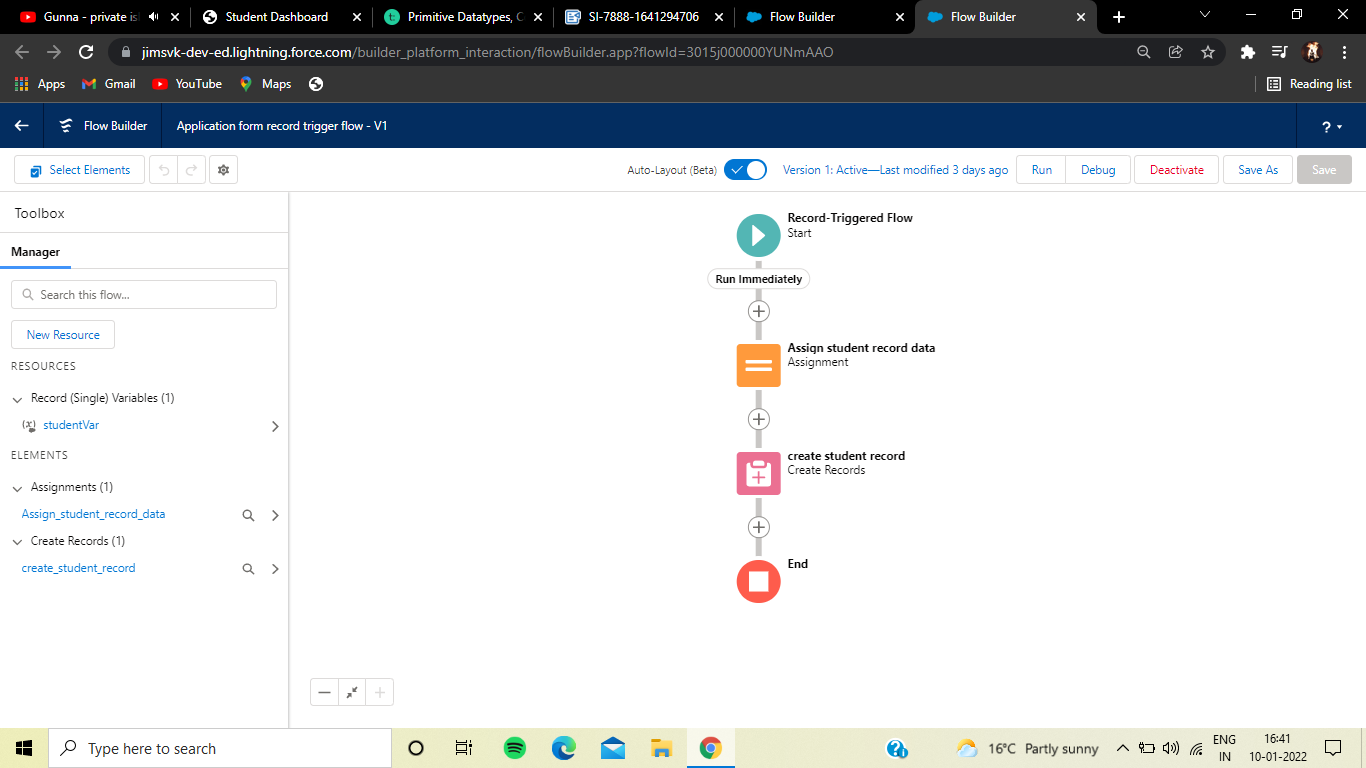
1. First deactivate the process builder which we created earlier.

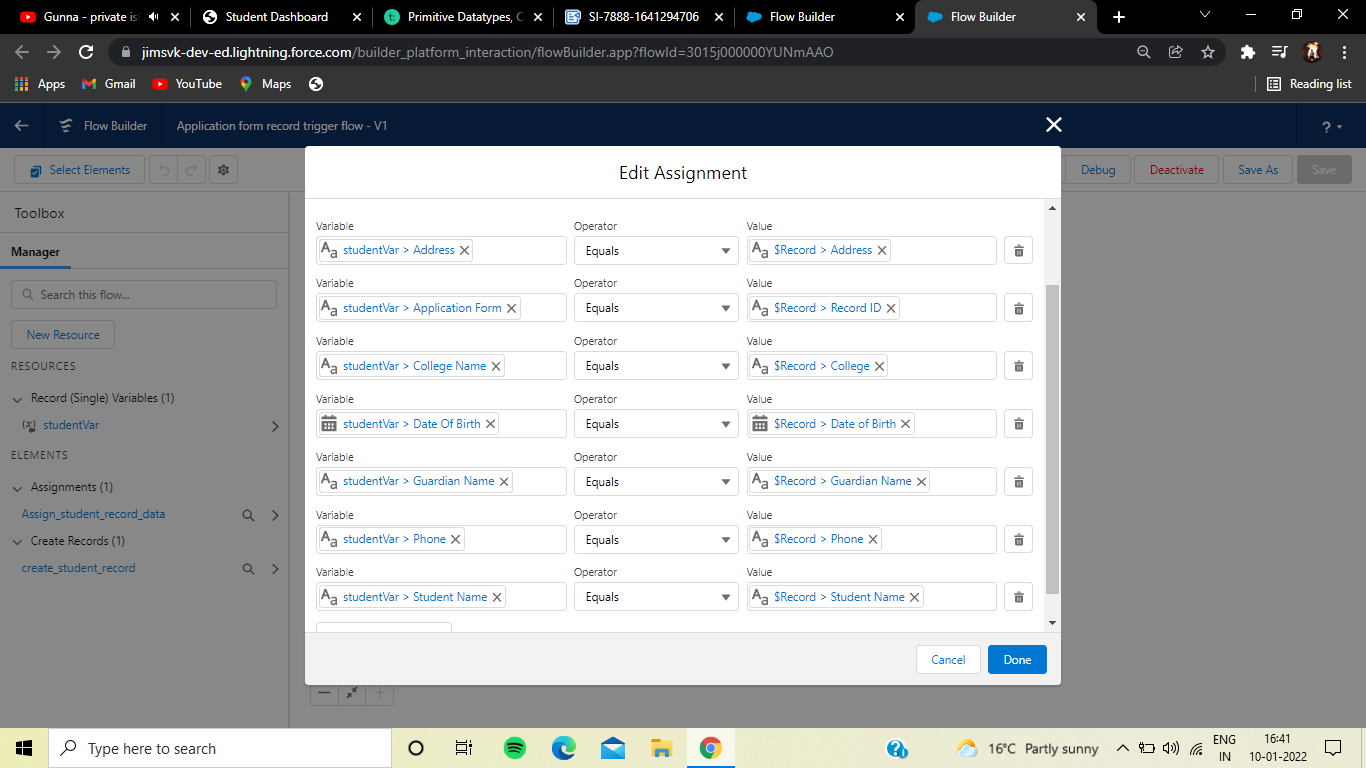
2. Now search for flows and select new flow ->record triggered flow

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

}

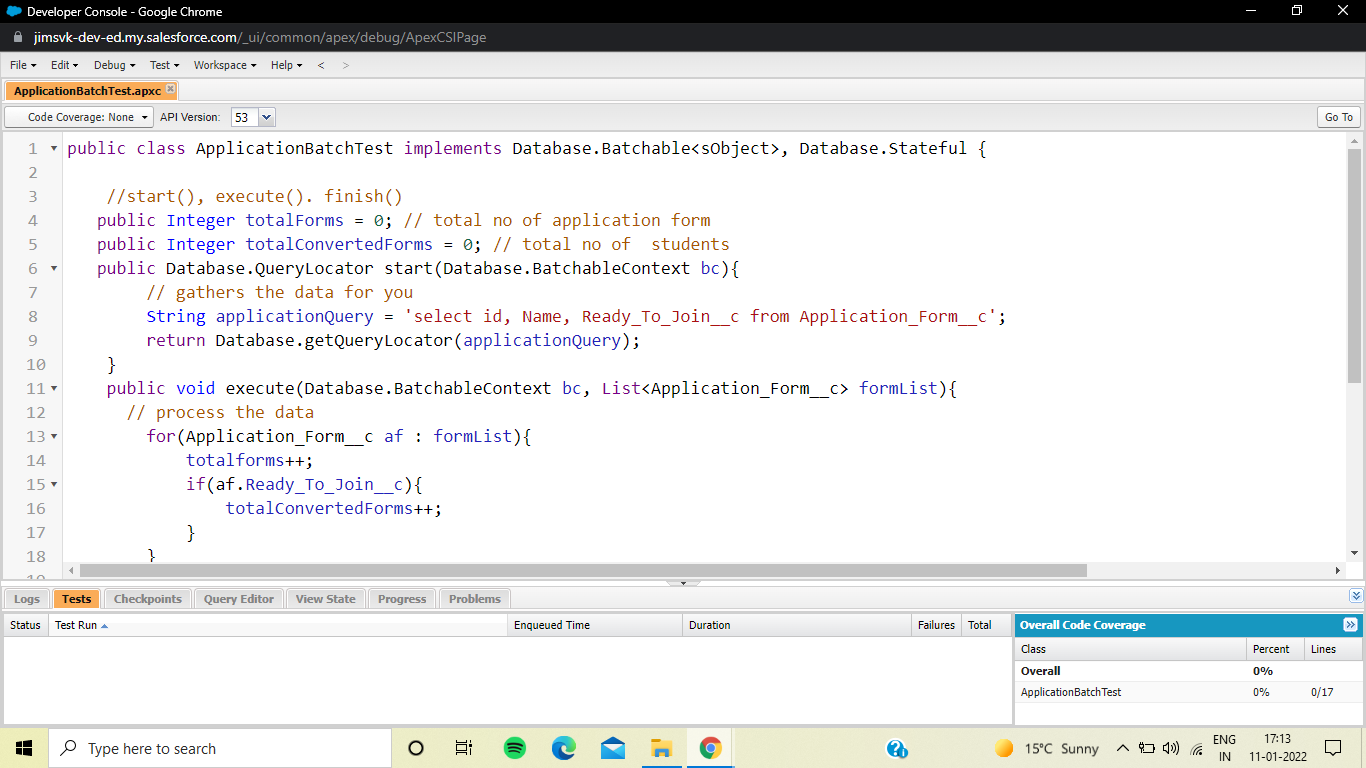
public void finish(Database.BatchableContext bc){

// 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);

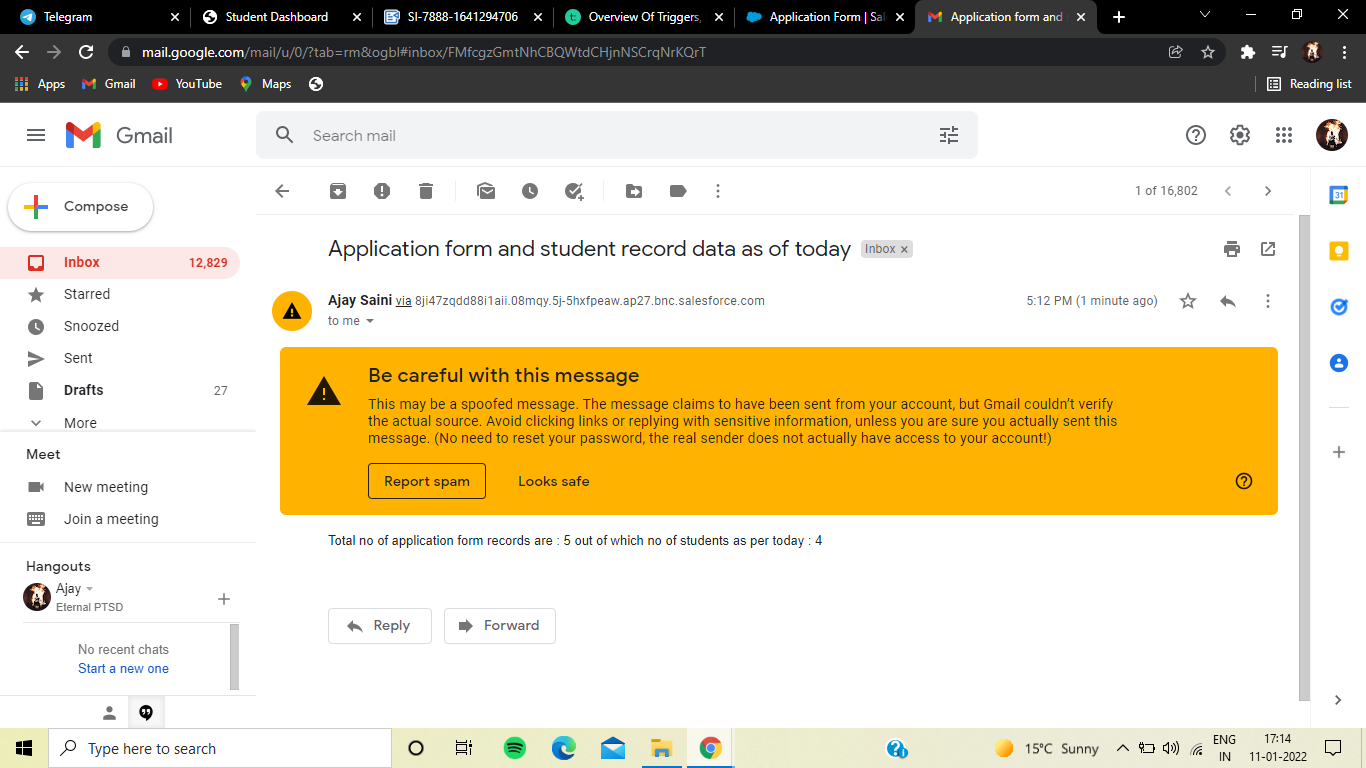
String[] emailAddress = new String[]{'sainiajay882@gmail.com'};

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

}

}

