**LOYOLA ACADEMY DEGREE & PG COLLEGE**

**OLD ALWAL, SECUNDERABAD - 500 010, TELANGANA, INDIA**

**An Autonomous Institution Affiliated to Osmania University**

**Re-accredited with ‘A’ Grade (III Cycle) by NAAC A “College with Potential for Excellence” by UGC**

# Practical Record CERTIFICATE

This is to certify that this is a Bonafide record work done in **CLOUD BASED APPLICATION DEVELOPMENT AND DEPLOYMENT** practical during 3RD year 5TH semester of the academic year 2025 - 2026

**Name:**

**UID No:**

**Class:**

**Signature of Internal** **Signature of HoD**

**Signature of External** **Signature of Principal**

# INDEX

## PROGRAM NO: 01

## Design and development of web applications using MVC Framework

**1.Requirement Analysis**

- Understand the purpose of the web app and its functionalities.
- Identify input, processing, and expected output.

2.**Setup Development Environment**

- Install **JDK**, IDE (Eclipse/STS/IntelliJ), and **Apache Tomcat** or use Spring Boot.
- Add MVC libraries or create a Spring Boot MVC project.

3.**Create Project Structure**

- Create a new **MVC Project**.
- Configure **folders**:
    - **Model** → Java classes for data (POJO).
    - **View** → JSP/HTML/Thymeleaf templates.
    - **Controller** → Java classes to handle requests.

4.**Design Model**

- Define data objects (e.g., `Student.java`, `Product.java`).
- Connect to a database if needed (JDBC/ORM).

5.**Design View**

- Create **UI pages** (HTML/JSP) for user interaction.
- Add forms, tables, buttons, etc.

6.**Design Controller**

- Write **Controller classes** to handle requests and map them to Models & Views.
- Example: `@Controller` in Spring MVC.

7.**Configure Application**

- Configure **application.properties**, `web.xml`, or Spring Boot auto-configurations.
- Add routes, database connection, and dependencies.

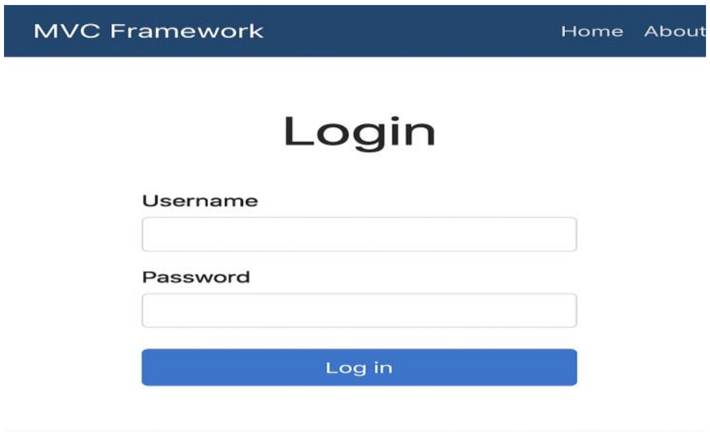
8.**Testing & Debugging**

- Run on **localhost**.
- Test all features and fix any errors.

9.**Deployment**

- Package the application as `.war` (for Tomcat) or run Spring Boot `.jar`.
- Deploy on a server or cloud platform (AWS, GAE, Azure).

**OUTPUT:**



## PROGRAM NO: 02

## Design and deploy a simple message on local host using MVC framework

### 1. Install the tooling (one-time)

- Install **.NET SDK 8+** from Microsoft's site.
- After installing, verify:

```
dotnet --version
```

You should see something like 8.x.x.

### 2. Create a new MVC project

```
dotnet new mvc -n HelloMvc
cd HelloMvc
```

This scaffolds a ready-to-run MVC app.

### 3. Run it once (baseline)

```
dotnet run --urls http://localhost:5000
```

Open http://localhost:5000 to see the default template running.

*(Keep this terminal open while testing.)*

### 4. Add a Controller that returns your message

Create Controllers/HelloController.cs with this content:

```
using Microsoft.AspNetCore.Mvc;

 namespace HelloMvc.Controllers
{
 public class HelloController : Controller
{
// Returns a View (HTML) showing the message
   public IActionResult Index()
{
 ViewData["Message"] = "Hello from MVC on localhost!";
 return View();
}
```

```
  // Optional: returns plain text (no View)
   public IActionResult Plain()
 {
   return Content("Hello from MVC (plain text)!");
 }
 }
}
```

**5. Add the View (HTML page)**

Create the folder Views/Hello/ and inside it the file Views/Hello/Index.cshtml:

```
@{
ViewData["Title"] = "Hello";
}
<h1>@ViewData["Message"]</h1>
<p>This page was rendered by the View, populated by the Controller.</p>
```

**6. Ensure routing is standard (usually already set)**

Open Program.cs and confirm you have this (it’s generated by the template):

```
 var builder = WebApplication.CreateBuilder(args);
 builder.Services.AddControllersWithViews();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
app.UseExceptionHandler("/Home/Error");
app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
name: "default",
pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

This means visiting /Hello will call HelloController.Index.

**7. Run and see your message**

If you stopped the server, start it again:

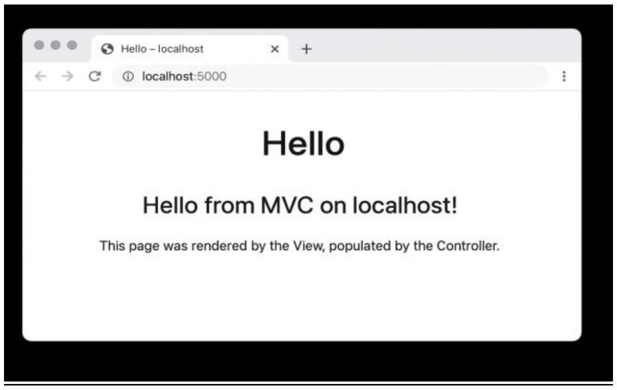
```
dotnet run --urls http://localhost:5000
```

Now browse to:

- **HTML view**: http://localhost:5000/Hello
- **Plain text**: http://localhost:5000/Hello/Plain

You should see your custom message.

**OUTPUT:**



Hello

Hello from MVC on localhost!

This page was rendered by the View, populated by the Controller.

## PROGRAM NO: 03

## Design and develop a Student form on local host using MVC framework

**1. StudentmvcApplication.java**

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class StudentmvcApplication {
  public static void main(String[] args) {
     SpringApplication.run(StudentmvcApplication.class, args);
  }
}
```

**2.StudentController.java**

```
package com.example.demo.controller;
import com.example.demo.model.student;
import com.example.demo.service.StudentService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
public class StudentController {

  private final StudentService service;

  public StudentController(StudentService service) {
     this.service = service;
  }

  @GetMapping("/")
  public String viewHomePage(Model model) {
     model.addAttribute("students", service.getAllStudents());
     return "index";
  }
```

```
    @GetMapping("/add")
    public String showAddForm(Model model) {
        model.addAttribute("student", new student());
        return "add-student";
    }

    @PostMapping("/save")
    public String saveStudent(@ModelAttribute student student) {
        service.save(student);
        return "redirect:/";
    }
}
```

**3.Student.java**

```
package com.example.demo.model;



import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String email;

    // Getters and setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

**4.StudentRepository.java**

```
package com.example.demo.repository;



import com.example.demo.model.student;
import org.springframework.data.jpa.repository.JpaRepository;

public interface StudentRepository extends JpaRepository<student, Long> {
}
```

**5.StudentService.java**

```
package com.example.demo.service;



import com.example.demo.model.student;
import com.example.demo.repository.StudentRepository;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class StudentService {

    private final StudentRepository repository;
    public StudentService(StudentRepository repository) {
        this.repository = repository;
    }

    public void save(student student) {
        repository.save(student);
    }

    public List<student> getAllStudents() {
        return repository.findAll();
    }
}
```

**6.template**

Add-student.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Add Student</title>
</head>
<body>
    <h2>Add Student</h2>
    <form action="#" th:action="@{/save}" th:object="${student}" method="post">
        Name: <input type="text" th:field="*{name}" /><br/>
        Email: <input type="text" th:field="*{email}" /><br/>
        <button type="submit">Save</button>
    </form>
</body>
</html>
```

**7. index.html**

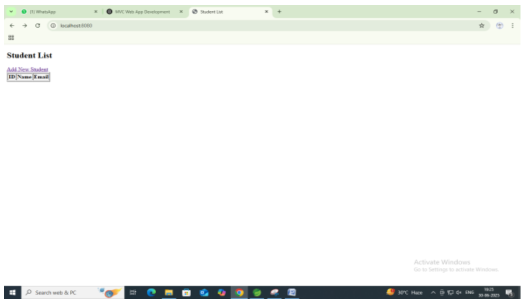
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Student List</title>
</head>
<body>
    <h2>Student List</h2>
    <a th:href="@{/add}">Add New Student</a>
    <table border="1">
        <tr>
            <th>ID</th><th>Name</th><th>Email</th>
        </tr>
        <tr th:each="student : ${students}">
```

```
            <td th:text="${student.id}"></td>
            <td th:text="${student.name}"></td>
            <td th:text="${student.email}"></td>
        </tr>
    </table>
</body>
</html>
```

**8.application properties**

```
spring.application.name=studentmvc
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
```

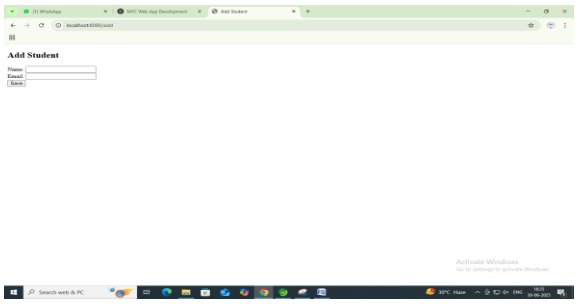
**<u>OUTPUT:</u>**



Student List

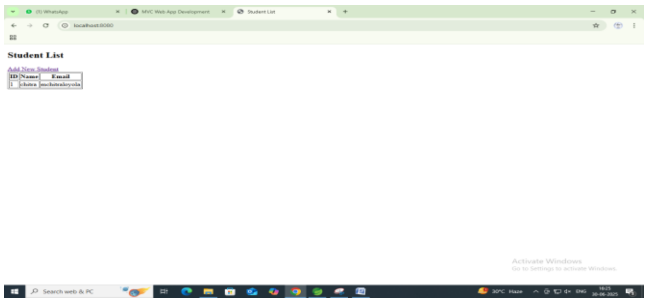
Add New Student

| ID | Name | Email |
|---|---|---|



Add Student

Name:
Email:
Save



Student List

Add New Student

| ID | Name | Email |
|---|---|---|
| 1 | chitra | mchitraloyola |

## PROGRAM NO: 04

## Design and develop web application using spring boot web application and running on local host

**1. ThymeleafspringboottutoaApplication.java**

```
  package com.example.demo;

  import org.springframework.boot.SpringApplication;
  import org.springframework.boot.autoconfigure.SpringBootApplication;
  @SpringBootApplication
   public class ThymeleafespringboottutoApplication {
   public static void main(String[] args) {
   SpringApplication.run(ThymeleafespringboottutoApplication.class, args);
 }
}
```

**2. Helloworldcontroller.java**

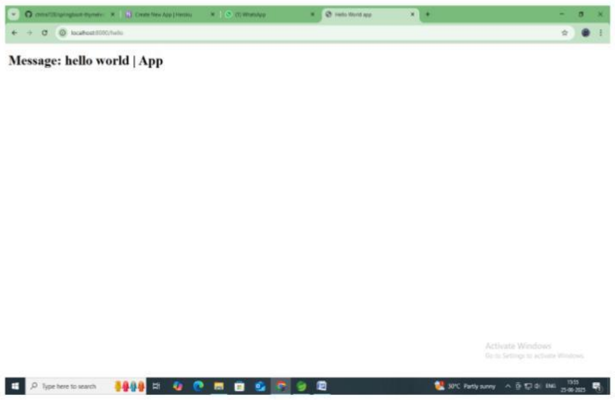
```
 package com.example.demo;

 import org.springframework.stereotype.Controller;
 import org.springframework.ui.Model;
 import org.springframework.web.bind.annotation.GetMapping;



  @Controller
   public class helloworldcontroller {
   @GetMapping("/hello")
    public String hello(Model model) {
    model.addAttribute("message", "hello world");
     return "helloworld";
 }
}
```

**3.helloworld.html**

```
 <!DOCTYPE html>
 <html xmlns:th="http://www.thymeleaf.org">
 <head>
 <meta charset="ISO-8859-1">
```

```
<title>Hello World app</title>
</head>
<body>
<h1 th:text="'Message: ' + ${message} + ' | App'"></h1>
</body>
</html>
```

**OUTPUT:**



Message: hello world | App

## PROGRAM NO: 05

## Design and deploy a python program using spring boot MVC web application and deploy on Google App Engine by creating a repository

### 1. Setup Environment

- Install **Java JDK** (17 or 21 LTS), **Maven**, and **Python 3**.
- Install **Spring Boot CLI** or use **start.spring.io**.
- Install **Google Cloud SDK** (gcloud CLI) and configure your Google Cloud project.

### 2. Create Spring Boot MVC Project

- Go to start.spring.io
- Select:
  - **Project:** Maven
  - **Language:** Java
  - **Spring Boot:** Latest stable
  - **Dependencies:** Spring Web, Thymeleaf (or REST if API-based)
- Generate & open project in **IDE** (IntelliJ / VS Code / Eclipse).

### 3. Integrate Python Program

- Place your Python script in src/main/resources/python/.
- Call Python from Spring Boot controller using **ProcessBuilder** or **Jython**.
- Example:

```
        ProcessBuilder pb = new ProcessBuilder("python3",
"src/main/resources/python/your_script.py");
        Process p = pb.start();
```

```
# add_two_numbers.py

# Take two numbers as input
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))

# Calculate sum
sum = a + b

# Print result
print("The sum is:", sum)
```

### 4. Build and Test Locally

- Run the project:

  “ mvn spring-boot:run”

- Open http://localhost:8080 in browser and verify output.

## 5. Create Repository

- Initialize Git repo:

```
git init
 git add .
 git commit -m "Initial Commit"
```

- Push to GitHub:

```
git remote add origin <your-repo-url>
git push -u origin main
```

## 6. Prepare for Google App Engine

- Create app.yaml in project root:

```
“runtime: java17
    entrypoint: java -jar target/<your-app>.jar
    service: default”
```

- Package app:

  “mvn clean package”

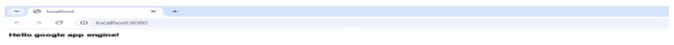
## 7. Deploy to Google App Engine

- Deploy with gcloud:

  “gcloud app deploy target/<your-app>.jar”

- Open deployed app: “gcloud app browse”

**OUTPUT:**



```
Hello google app engine!
```

# PROGRAM NO: 06

## Case study on the features of GAE PaaS Model

### 1. Introduction

- Briefly introduce **Google App Engine (GAE)**.
- Explain that it is a **PaaS (Platform as a Service)** offering from Google Cloud.
- Mention its purpose: **build, deploy, and scale applications without managing infrastructure**.

### 2. Objective of the Case Study

- State what you aim to achieve:
    - Understand GAE features.
    - Analyze its benefits for developers.
    - See real-world use cases.

### 3. Explain the GAE Architecture

- Describe **how GAE works**:
    - Supports multiple languages (Java, Python, Go, Node.js).
    - Automatic load balancing and scaling.
    - Deployment via gcloud app deploy.

### 4. List and Explain Key Features

Focus on **important features** with short points:

- **Automatic Scaling** – adjusts resources based on traffic.
- **Managed Infrastructure** – no need to manage servers.
- **Built-in Security** – authentication, firewalls, HTTPS.
- **Multiple Runtimes** – standard & flexible environments.
- **Version Control** – easy rollback and traffic splitting.
- **Integration with GCP** – datastore, cloud SQL, pub/sub.
- **Pay-per-Use Pricing** – pay only for what you use.

### 5. Case Example

- Pick a simple scenario (like deploying a Spring Boot app, Flask API, or website).
- Mention steps followed:
    1. Write code.
    2. Create app.yaml.
    3. Deploy with gcloud app deploy.
    4. Access via <project-id>.appspot.com.

## 6. Benefits Observed

- Easy to deploy & scale.
- No downtime during deployment.
- Reduced operational complexity.
- Cost-effective for startups & small projects.

## 7. Challenges / Limitations

- Limited customization in Standard Environment.
- Vendor lock-in risk.
- Cold start issues (for some runtimes).

## 8. Conclusion

- Summarize findings:
    - GAE is ideal for quick deployments and scalable apps.
    - Best suited for developers who don’t want to manage infrastructure manually.

## PROGRAM NO: 07

## Creating and running web application on local host and deploying the same in GAE

### 1. Setup Environment

- Install **Java JDK** (17/21) & **Maven** (for Spring Boot app).
- Install **IDE** (IntelliJ / Eclipse / VS Code).
- Install **Google Cloud SDK** (gcloud CLI).
- Create a **Google Cloud Project** in Google Cloud Console.

### 2. Create Web Application

- Go to start.spring.io.
- Choose:
  - **Project:** Maven
  - **Language:** Java
  - **Spring Boot:** Latest version
  - **Dependencies:** Spring Web
- Download and open the project in your IDE.

### 3. Write a Simple Controller

```
@RestController
public class HomeController {
   @GetMapping("/")
   public String home() {
      return "Hello, Google App Engine!";
   }
}
```

### 4. Run Locally

- Build and run:

```
mvn spring-boot:run
```

- Open http://localhost:8080 in browser to check output.

### 5. Prepare for Deployment

- Package the app:

```
mvn clean package
```

- Create **app.yaml** in project root:

```
runtime: java17
entrypoint: java -jar target/<your-app>.jar
service: default
```

**6. Deploy to Google App Engine**

- Authenticate and set project:

```
gcloud auth login
gcloud config set project <your-project-id>
```

- Deploy:

```
gcloud app deploy target/<your-app>.jar
```

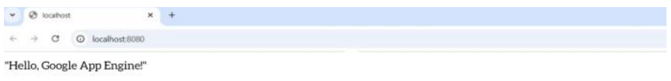
- Open deployed app:

```
gcloud app browse
```

**7. Verify Deployment**

- Visit the provided URL (https://<project-id>.appspot.com) to see the running app.

**OUTPUT:**



"Hello, Google App Engine!"

## PROGRAM NO: 08

## Write a case study on ASP.NET

### 1. Choose the Objective

Decide what your case study will focus on:

- ASP.NET as a framework for web development
- Its advantages over other frameworks
- A real-life project or application built using ASP.NET

### 2. Research ASP.NET

- Collect key information:
  - **Overview:** Introduce ASP.NET as a Microsoft framework for building web apps.
  - **Architecture:** Explain MVC pattern (Model-View-Controller) and .NET Core runtime.
  - **Languages Supported:** C#, VB.NET, F#.
  - **Development Tools:** Visual Studio / Visual Studio Code.

### 3. List Key Features

Include ASP.NET's main features in your case study:

- Cross-platform (.NET Core)
- MVC architecture
- Built-in security (authentication & authorization)
- Razor Pages for dynamic web content
- Scalability and performance
- Integration with Azure and databases (SQL Server, MySQL)
- Rich library and NuGet packages

### 4. Select a Case Example

- Pick a real or hypothetical application to explain:
  Example:
  - **Title:** "Online Student Portal using ASP.NET Core MVC"
  - Explain how it was developed (frontend, backend, database used).
  - Show how ASP.NET's features (security, MVC, deployment) made development easier.

### 5. Analyze Benefits

- Discuss how ASP.NET helped solve the problem:
  - Faster development using scaffolding.
  - Secure user login with Identity framework.
  - Easy deployment on IIS or Azure.
  - Good performance with caching and async programming.

## 6. Challenges and Limitations

- Include realistic issues:
  - Learning curve for beginners.
  - Licensing cost (for Windows Server/IIS).
  - Heavier runtime compared to lightweight frameworks.

## 7. Conclusion

- Summarize why ASP.NET is a powerful web development framework.
- Suggest where it is most suitable (enterprise apps, e-commerce, portals).

## PROGRAM NO: 09

## Studying the feature of Azure Platform

### 1. Understand the Basics

- Learn what **Microsoft Azure** is: a **cloud computing platform** providing **IaaS, PaaS, and SaaS** solutions.
- Know its purpose: **build, deploy, and manage apps/services on Microsoft-managed data centers**.

### 2. Explore Azure Services Categories

Break Azure into its main service areas:

- **Compute** – Virtual Machines, Azure App Service, Functions
- **Storage** – Blob, File, Queue, Disk storage
- **Networking** – Virtual Network, Load Balancer, CDN
- **Databases** – Azure SQL, Cosmos DB
- **AI & ML** – Cognitive Services, Azure ML
- **DevOps Tools** – Azure DevOps, GitHub Actions integration
- **Security** – Azure AD, Key Vault, Defender

### 3. Learn Key Features

Focus on the most important features:

- **Scalability & Elasticity** – Auto-scale resources based on demand
- **High Availability** – Global data centers, 99.9% uptime SLAs
- **Pay-as-you-Go Pricing** – Pay only for resources used
- **Security & Compliance** – Identity management, encryption, compliance certifications
- **Hybrid Capabilities** – Integration with on-premises systems
- **Global Reach** – Services in multiple regions worldwide

### 4. Use Azure Portal

- Sign in to Azure Portal.
- Explore **Dashboard, Resource Groups, Virtual Machines, App Services**.
- Try creating a **free-tier service** (like a simple web app).

### 5. Study Documentation & Tutorials

- Visit **Microsoft Learn** and go through:
  - *Azure Fundamentals (AZ-900)* learning path
  - Quickstarts for App Service, Virtual Machine, Functions

    - Tutorials for monitoring, scaling, and securing apps

## 6. Hands-on Practice

- Deploy a small **sample project**:
    - Example: Simple ASP.NET or Python web app
    - Deploy using Azure App Service
    - Enable monitoring & scaling to see features in action

## 7. Analyze Strengths & Challenges

- Document what makes Azure powerful:
    - Easy integration with Microsoft ecosystem
    - Enterprise-grade security
    - Global availability
- Note potential challenges:
    - Pricing complexity
    - Learning curve for non-Microsoft developers

## 8. Prepare Summary or Case Study

- Summarize your findings:
    - Key features & services
    - Benefits for businesses
    - Where Azure is most useful (enterprise, AI projects, DevOps pipelines)

## <u>PROGRAM NO: 10</u>

## <u>Write the steps to create an application in Dropbox to store data securely</u>

**1. Sign In to Dropbox**

- Go to the Dropbox App Console.
- Log in with your Dropbox account (create one if you don't have it).

**2. Create a New App**

- Click **Create App**.
- Choose:
  - **Scoped Access** (recommended for security).
  - **App Folder** (only lets your app access its own folder) or **Full Dropbox** (if you need full access).

**3. Name Your App**

- Enter a unique name for your app.
- Click **Create App** to generate it.

**4. Configure Permissions**

- Under **Permissions**, enable scopes your app needs:
  - files.content.write → to upload files.
  - files.content.read → to read/download files.
  - Add others only if required (least privilege principle).
- Save changes.

**5. Generate API Keys**

- Go to **Settings** tab.
- Copy the **App Key** and **App Secret** (used for authentication in your code).

**6. Create Access Token**

- In the **OAuth 2** section, click **Generate Access Token** for quick testing.
- Store this token securely (environment variables, not hardcoded).

**7. Integrate with Your Application**

- Use Dropbox **SDK** or **API** in your application:

    - Authenticate using your token.
    - Implement secure file upload/download.
    - Example: Use HTTPS for communication.

**8. Test & Deploy**

- Test uploading a sample file to Dropbox.
- Verify it appears in the App Folder in your Dropbox.
- Deploy your application after successful testing.