
Reimbursement Submission Application Documentation

Overview

The **Reimbursement Submission Application** allows university employees to submit receipts for reimbursement. It provides a frontend for submitting receipt data and a backend API to store and manage the receipts. The backend and frontend are designed to work together to streamline the reimbursement process.

---

1. Application Architecture

- **Frontend**: Angular
- **Backend**: .NET Core (API)
- **Database**: MySQL (using Pomelo Entity Framework Core for MySQL)

The frontend interacts with the backend via REST API endpoints, and data is stored in a MySQL database.

---

2. Backend Setup (API)

Requirements

- .NET SDK (version 8.0 or above)
- MySQL database (locally or using any MySQL service)
- Visual Studio or any code editor that supports .NET

Steps to Run the Backend

1. **Clone the Project** (if you haven't already)
    - Clone the backend repository to your local machine.
2. **Install Dependencies**

Open a terminal in the backend directory and run the following command to install the necessary dependencies:

```
dotnet restore
```

    -
3. **Configure Database Connection**
    - Ensure you have MySQL running and a database named `reimbursementapp` created.

In `appsettings.json`, update the **DefaultConnection** string with your MySQL server details:

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"server=localhost;port=3306;database=reimbursementapp;user=root;password=root"
  }
}
```

    -
4. **Run Database Migrations**

Apply the database migrations to set up the required tables by running the following command in your terminal:

```
dotnet ef database update
```

    -
5. **Run the Backend**

Start the backend API using the following command:

```
dotnet run
```

-
- By default, it will run on `http://localhost:5141`.

---

3. Frontend Setup (Angular)

Requirements

- Node.js (version 16 or above)
- Angular CLI

Steps to Run the Frontend

1. **Clone the Frontend Project** (if you haven't already)
    - Clone the Angular frontend repository to your local machine.
2. **Install Dependencies**

Navigate to the frontend directory in your terminal and install the required dependencies using:

```
npm install
```

-
3. **Run the Frontend**

Start the Angular development server by running:

```
ng serve
```

-

- By default, the frontend will be available at `http://localhost:4200`.

4. **Configure the API Endpoint**

   - Ensure the frontend is pointing to the correct backend API endpoint. If necessary, update the API URL in your Angular service to `http://localhost:5141/api/receipts`.

---

4. Testing the Application

After running both the **frontend** and **backend**:

1. Open your browser and go to `http://localhost:4200` to access the Angular frontend.

2. Fill out the reimbursement form and submit it. The form will send a POST request to `http://localhost:5141/api/receipts`.

3. The backend will handle the data submission, save it to the database, and return a success response.

You can also check the backend logs to see the console outputs for any errors or status updates.

---

5. Business Rules and Validations

The application implements the following business rules and validations:

- **Required Fields**: The **date**, **amount**, and **description** are required fields for the receipt submission.

- **Amount Validation**: The amount entered must be a positive number.

- **File Validation**: Only files under a certain size and in a valid format (e.g., images like PNG, JPG) are accepted.

- **Receipt Submission Logic**: The backend stores the submitted receipt data and file in the database. It ensures the receipt image is saved in the file system under the `uploads` directory.

---

6. Troubleshooting

- **404 Not Found Error**: If you are getting a `404` error, make sure that the backend API is running correctly on `http://localhost:5141`. Verify that the API controller's route is set up properly.

- **CORS Issues**: Ensure that the CORS policy in the backend is configured to allow requests from `http://localhost:4200` (the Angular frontend).

- **Database Connection**: If you encounter database connection issues, ensure that the MySQL database is running and the connection string in `appsettings.json` is correct.

Below is the full documentation for the Reimbursement Submission Application, covering both the backend (API) and frontend (Angular) components, as well as the additional business rules that were added.

Reimbursement Submission Application Documentation

Overview

The Reimbursement Submission Application allows university employees to submit receipts for reimbursement. It provides a frontend for submitting receipt data and a backend API to store and manage the receipts. The backend and frontend are designed to work together to streamline the reimbursement process.

---

1. Application Architecture

- **Frontend: Angular**
- **Backend: .NET Core (API)**
- **Database: MySQL (using Pomelo Entity Framework Core for MySQL)**

The frontend interacts with the backend via REST API endpoints, and data is stored in a MySQL database.

Backend (API)

The backend is built using .NET Core, with Entity Framework Core for data management. The application supports storing and retrieving receipt information, as well as handling file uploads for receipt images.

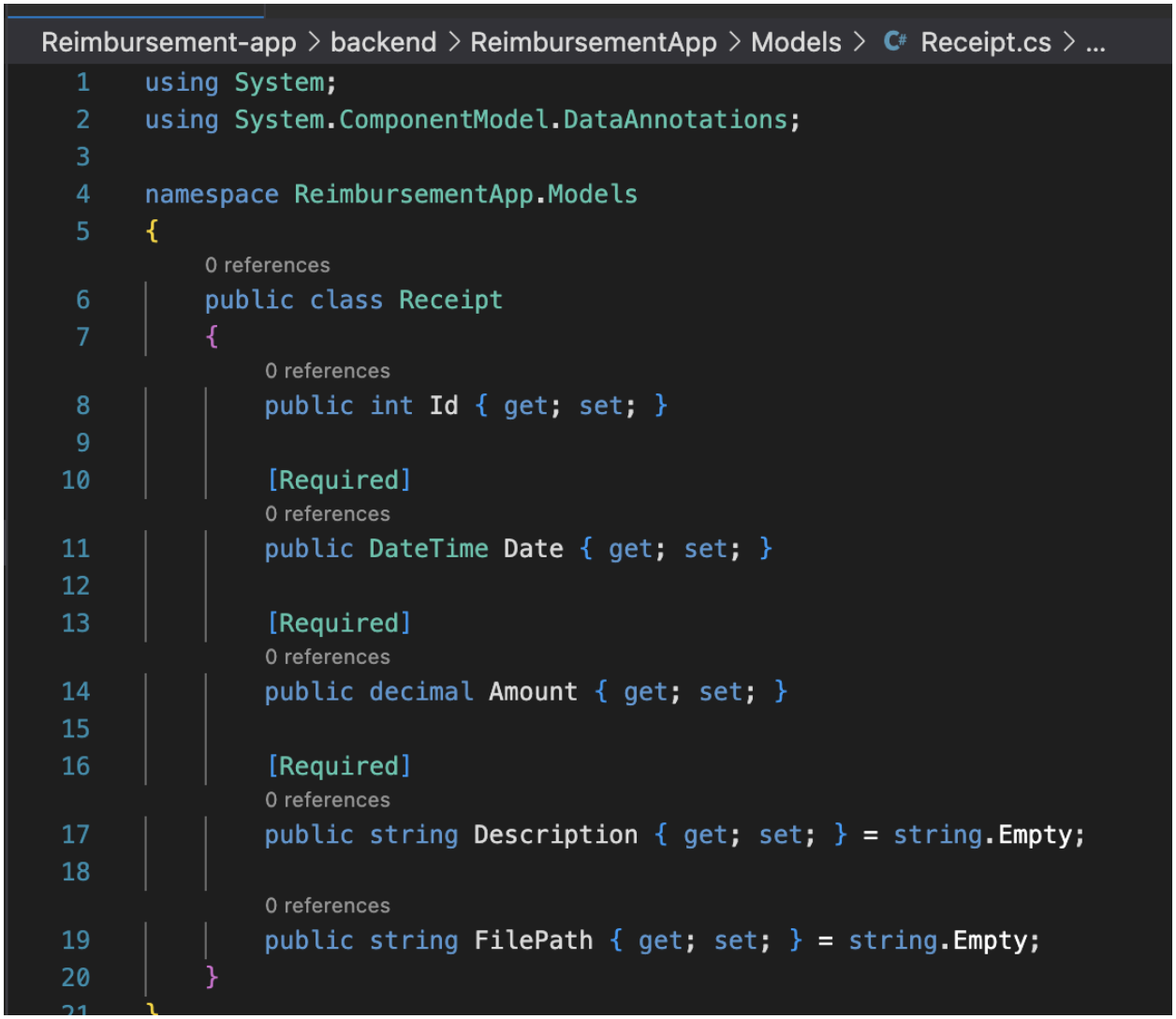
---

2. Backend Code Overview

Models

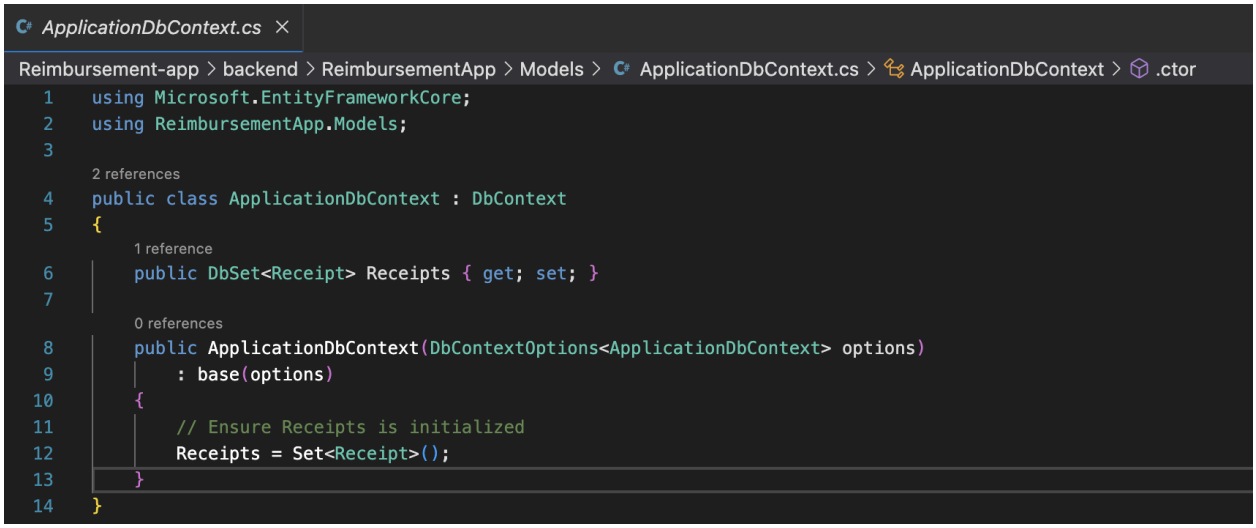
1. **Receipt Model**

This model represents a receipt submission, which includes essential details like the date, amount, description, and file path for the uploaded receipt image.



2. **ApplicationDbContext**

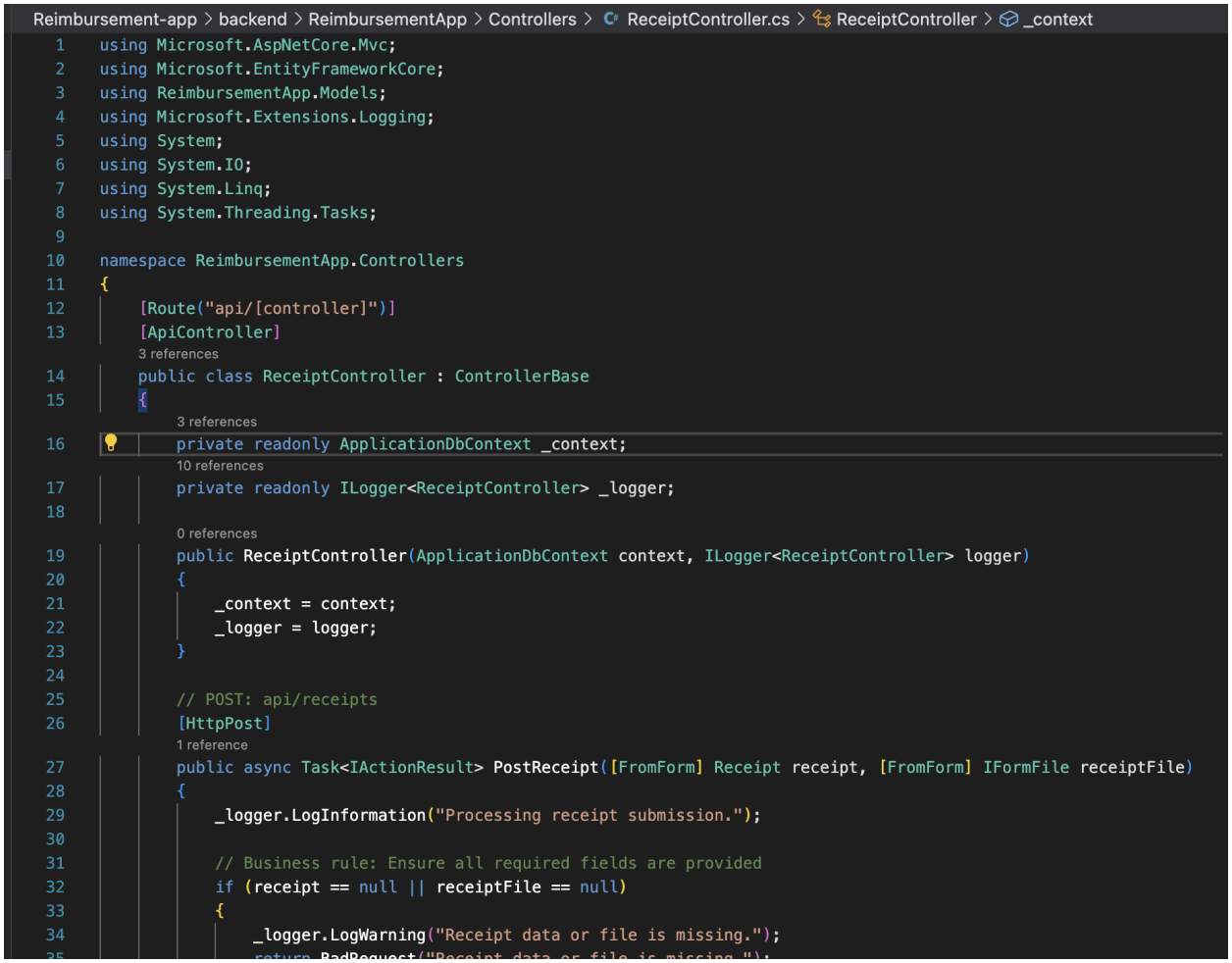
This is the Entity Framework DbContext class that represents the database context for the application. It includes the `DbSet<Receipt>` property for interacting with the receipts table in MySQL.

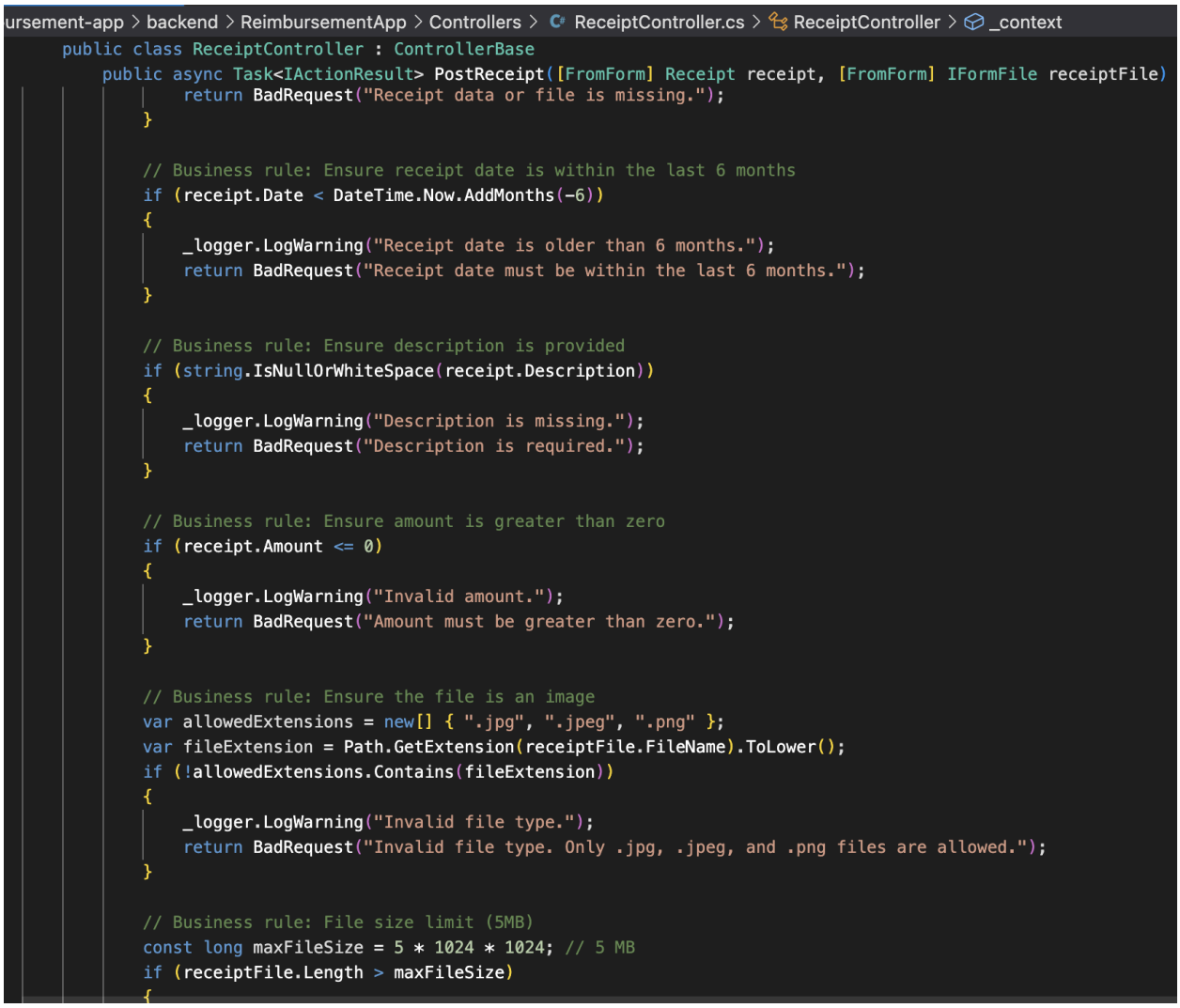


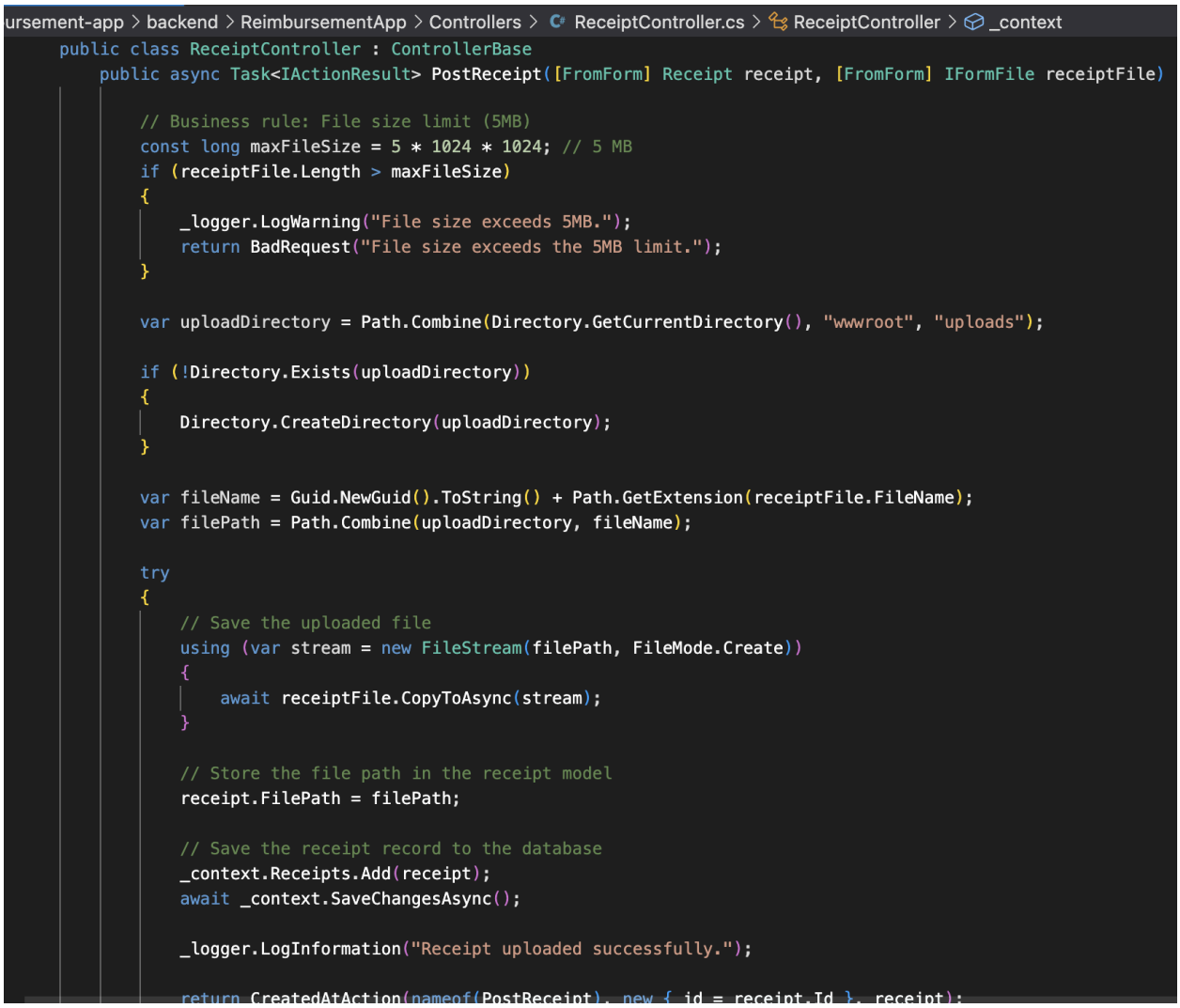
Controllers

1. **ReceiptController**

This API controller provides the functionality for handling the creation of receipts. It allows users to submit their receipts along with an image file via a POST request.


Reimbursement-app > backend > ReimbursementApp > Controllers > ReceiptController.cs > ReceiptController > _context
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ReimbursementApp.Models;
using Microsoft.Extensions.Logging;
using System;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
namespace ReimbursementApp.Controllers
{
[Route("api/[controller]")]
[ApiController]
3 references
public class ReceiptController : ControllerBase
{
3 references
private readonly ApplicationDbContext _context;
10 references
private readonly ILogger<ReceiptController> _logger;
0 references
public ReceiptController(ApplicationDbContext context, ILogger<ReceiptController> logger)
{
_context = context;
_logger = logger;
}
// POST: api/receipts
[HttpPost]
1 reference
public async Task<IActionResult> PostReceipt([FromForm] Receipt receipt, [FromForm] IFormFile receiptFile)
{
_logger.LogInformation("Processing receipt submission.");
// Business rule: Ensure all required fields are provided
if (receipt == null || receiptFile == null)
{
_logger.LogWarning("Receipt data or file is missing.");


ursement-app > backend > ReimbursementApp > Controllers > ReceiptController.cs > ReceiptController > _context
public class ReceiptController : ControllerBase
public async Task<IActionResult> PostReceipt([FromForm] Receipt receipt, [FromForm] IFormFile receiptFile)
return BadRequest("Receipt data or file is missing.");
}
// Business rule: Ensure receipt date is within the last 6 months
if (receipt.Date < DateTime.Now.AddMonths(-6))
{
_logger.LogWarning("Receipt date is older than 6 months.");
return BadRequest("Receipt date must be within the last 6 months.");
}
// Business rule: Ensure description is provided
if (string.IsNullOrWhiteSpace(receipt.Description))
{
_logger.LogWarning("Description is missing.");
return BadRequest("Description is required.");
}
// Business rule: Ensure amount is greater than zero
if (receipt.Amount <= 0)
{
_logger.LogWarning("Invalid amount.");
return BadRequest("Amount must be greater than zero.");
}
// Business rule: Ensure the file is an image
var allowedExtensions = new[] { ".jpg", ".jpeg", ".png" };
var fileExtension = Path.GetExtension(receiptFile.FileName).ToLower();
if (!allowedExtensions.Contains(fileExtension))
{
_logger.LogWarning("Invalid file type.");
return BadRequest("Invalid file type. Only .jpg, .jpeg, and .png files are allowed.");
}
// Business rule: File size limit (5MB)
const long maxFileSize = 5 * 1024 * 1024; // 5 MB
if (receiptFile.Length > maxFileSize)
{




```
            _logger.LogInformation("Receipt uploaded successfully.");

            return CreatedAtAction(nameof(PostReceipt), new { id = receipt.Id }, receipt);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error occurred while processing receipt submission.");
            return StatusCode(500, "Internal server error. Please try again later.");
        }
    }
}
```

Startup Configuration

1. **Program.cs**

This file configures the services, middleware, and routes for the application.

Reimbursement-app > backend > ReimbursementApp > Program.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ReimbursementApp.Models;
using Pomelo.EntityFrameworkCore.MySql.Infrastructure;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseMySql(builder.Configuration.GetConnectionString("DefaultConnection"),
    new MySqlServerVersion(new Version(8, 0, 23))));

builder.Services.AddControllers();

// CORS configuration for allowing requests from your Angular app
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAngularApp", builder =>
        builder.WithOrigins("http://localhost:4200")  // Allow Angular app's URL
               .AllowAnyMethod()
               .AllowAnyHeader());
});

// Add logging services
builder.Services.AddLogging();

// Build the app
var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}
else
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

// Enable static file serving (for uploads)
app.UseStaticFiles();

// Enable CORS
app.UseCors("AllowAngularApp"); // Apply CORS policy

app.UseRouting();
app.UseAuthorization();
app.MapControllers();  // Ensure this is added so that controllers are mapped

app.Run();
```

---

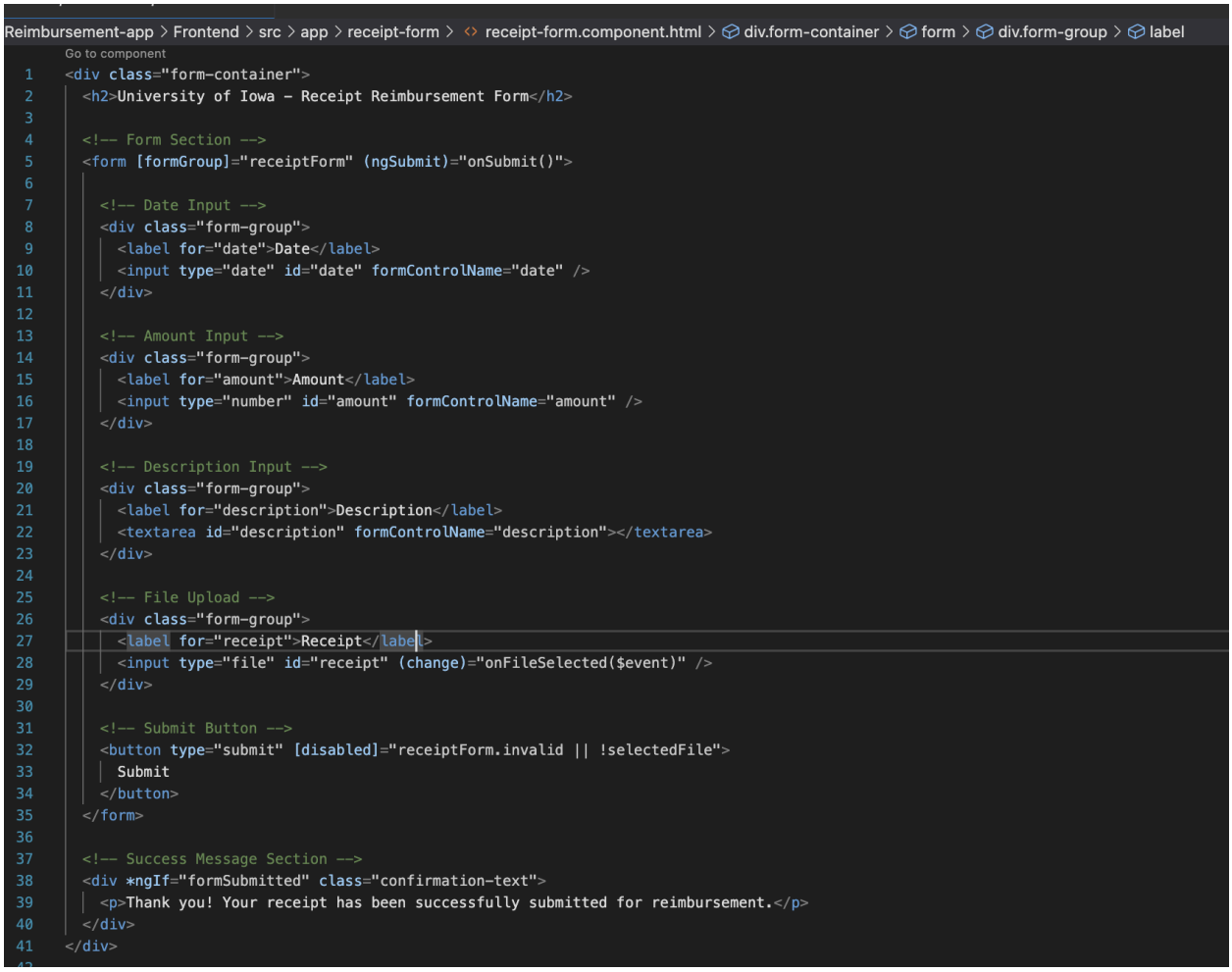
3. Frontend (Angular)

The frontend is developed using Angular to provide a simple and user-friendly interface for submitting reimbursement requests.

Frontend Code Overview

1. **Reimbursement Form Component**

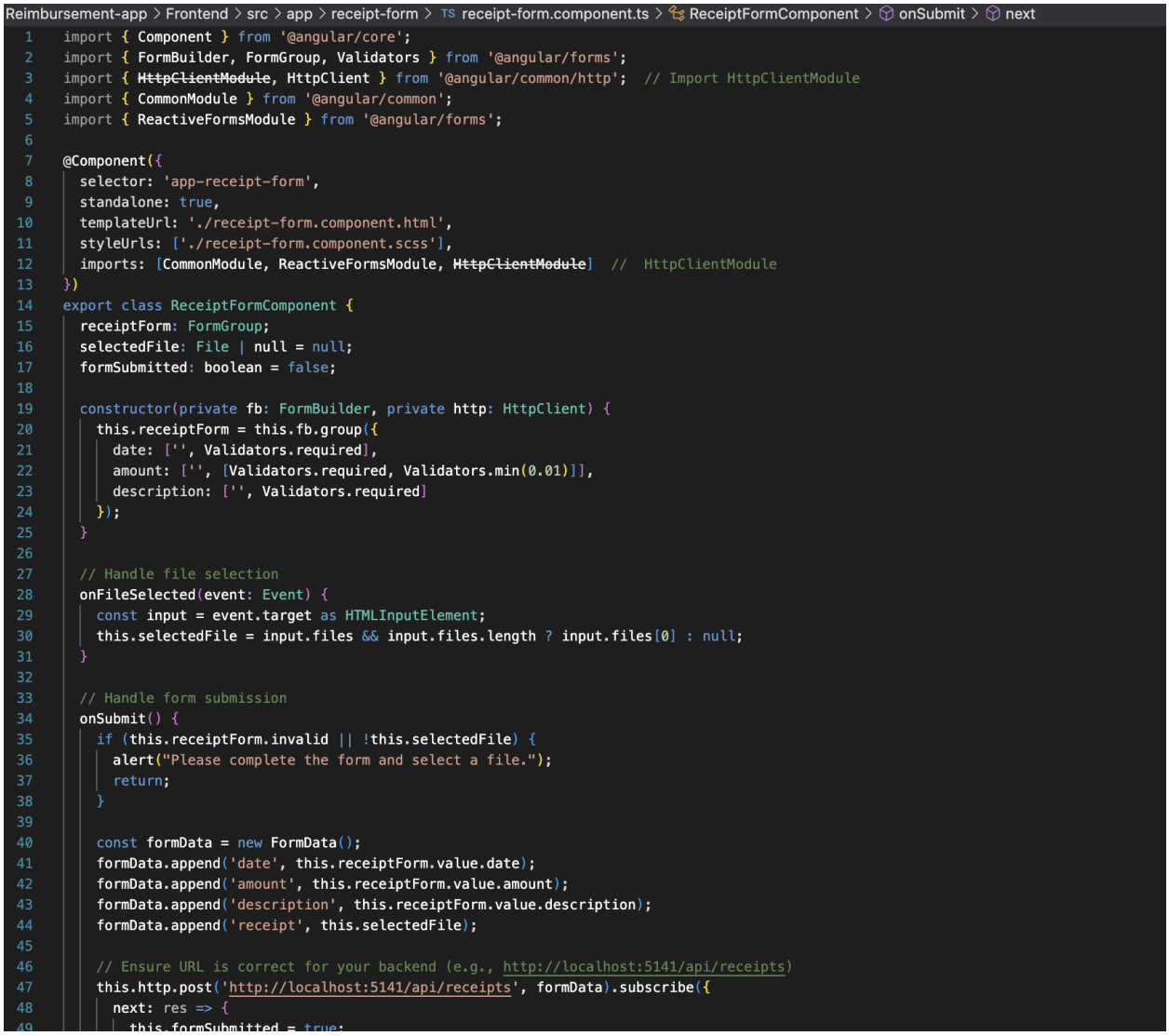
The form component allows users to fill in the details of their receipt and upload the receipt image file.



2. Component TypeScript File

The component logic handles form submission and file upload to the backend.

Reimbursement-app > Frontend > src > app > receipt-form > TS receipt-form.component.ts > ReceiptFormComponent > onSubmit > next



```ts
import { Component } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { HttpClientModule, HttpClient } from '@angular/common/http';  // Import HttpClientModule
import { CommonModule } from '@angular/common';
import { ReactiveFormsModule } from '@angular/forms';

@Component({
  selector: 'app-receipt-form',
  standalone: true,
  templateUrl: './receipt-form.component.html',
  styleUrls: ['./receipt-form.component.scss'],
  imports: [CommonModule, ReactiveFormsModule, HttpClientModule]  //  HttpClientModule
})
export class ReceiptFormComponent {
  receiptForm: FormGroup;
  selectedFile: File | null = null;
  formSubmitted: boolean = false;

  constructor(private fb: FormBuilder, private http: HttpClient) {
    this.receiptForm = this.fb.group({
      date: ['', Validators.required],
      amount: ['', [Validators.required, Validators.min(0.01)]],
      description: ['', Validators.required]
    });
  }

  // Handle file selection
  onFileSelected(event: Event) {
    const input = event.target as HTMLInputElement;
    this.selectedFile = input.files && input.files.length ? input.files[0] : null;
  }

  // Handle form submission
  onSubmit() {
    if (this.receiptForm.invalid || !this.selectedFile) {
      alert("Please complete the form and select a file.");
      return;
    }

    const formData = new FormData();
    formData.append('date', this.receiptForm.value.date);
    formData.append('amount', this.receiptForm.value.amount);
    formData.append('description', this.receiptForm.value.description);
    formData.append('receipt', this.selectedFile);

    // Ensure URL is correct for your backend (e.g., http://localhost:5141/api/receipts)
    this.http.post('http://localhost:5141/api/receipts', formData).subscribe({
      next: res => {
        this.formSubmitted = true;
```

---

4. Business Rules and Validations

The application implements the following business rules and validations:

- **Required Fields: The date, amount, and description are required fields for the receipt submission.**
- **Amount Validation: The amount entered must be a positive number.**
- **File Validation: Only files under a certain size and in a valid format (e.g., images like PNG, JPG) are accepted.**

- **Receipt Submission Logic: The backend stores the submitted receipt data and file in the database. It ensures the receipt image is saved in the file system under the `uploads` directory.**

---

5. Conclusion

This Reimbursement Submission Application is a simple yet effective solution to streamline the reimbursement process for university employees. By using Angular for the frontend and .NET Core for the backend, along with a MySQL database, the application offers a flexible and scalable solution that can be easily extended in the future to include features like user authentication, detailed reporting, and more advanced reimbursement management.