Timetable Generator Project

Delia Grigoriță *Ionela Cotiugă*
Paul Duțuc *Raul Boboc*

Coordinator: Conf. dr. Arusoaie Andrei

UAIC, FII, Advanced Software Engineering Techniques - 2024

Program Flow & Eclipse

AOP & MOP

Testing

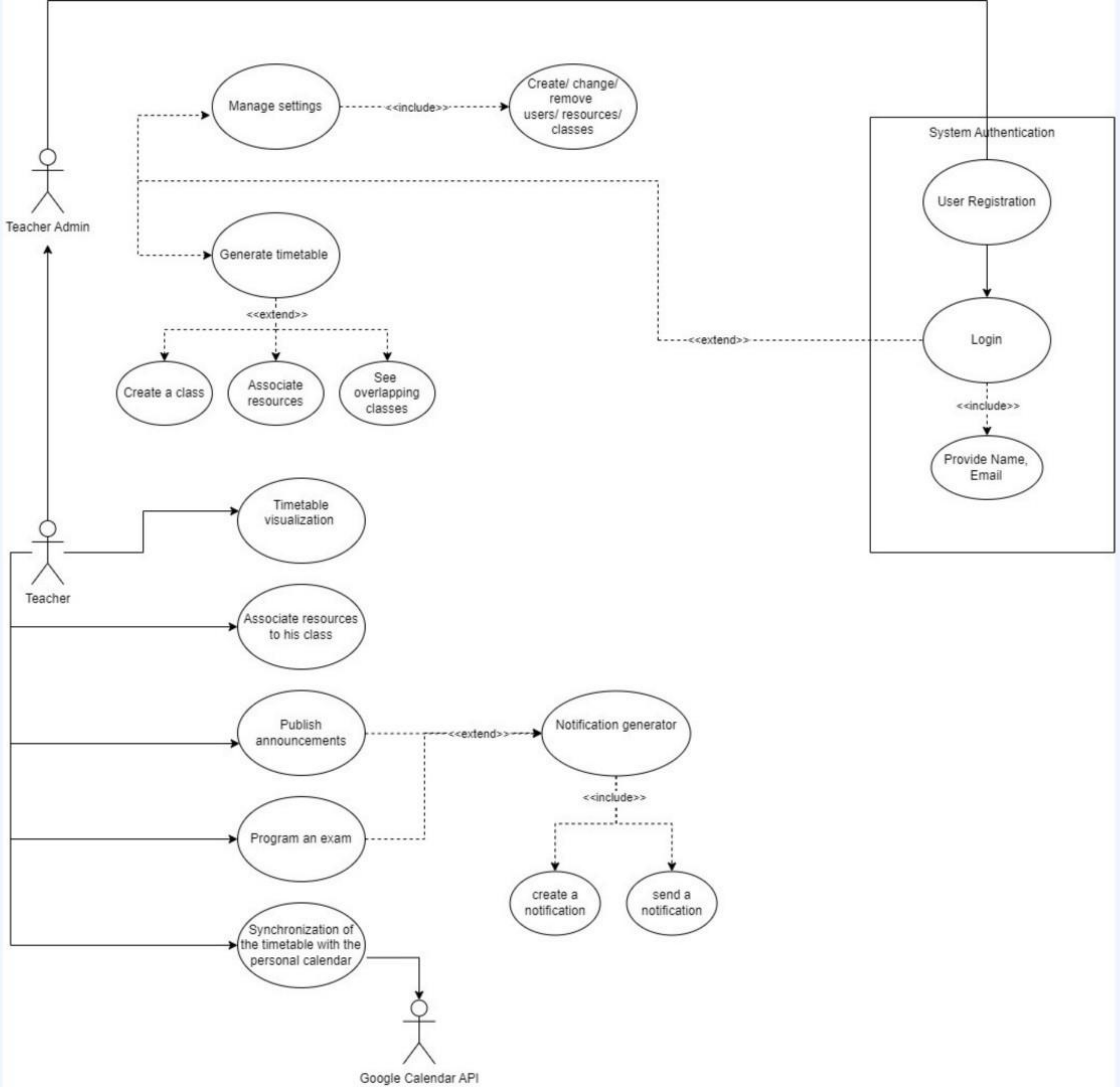
Github copilot

Program Flow

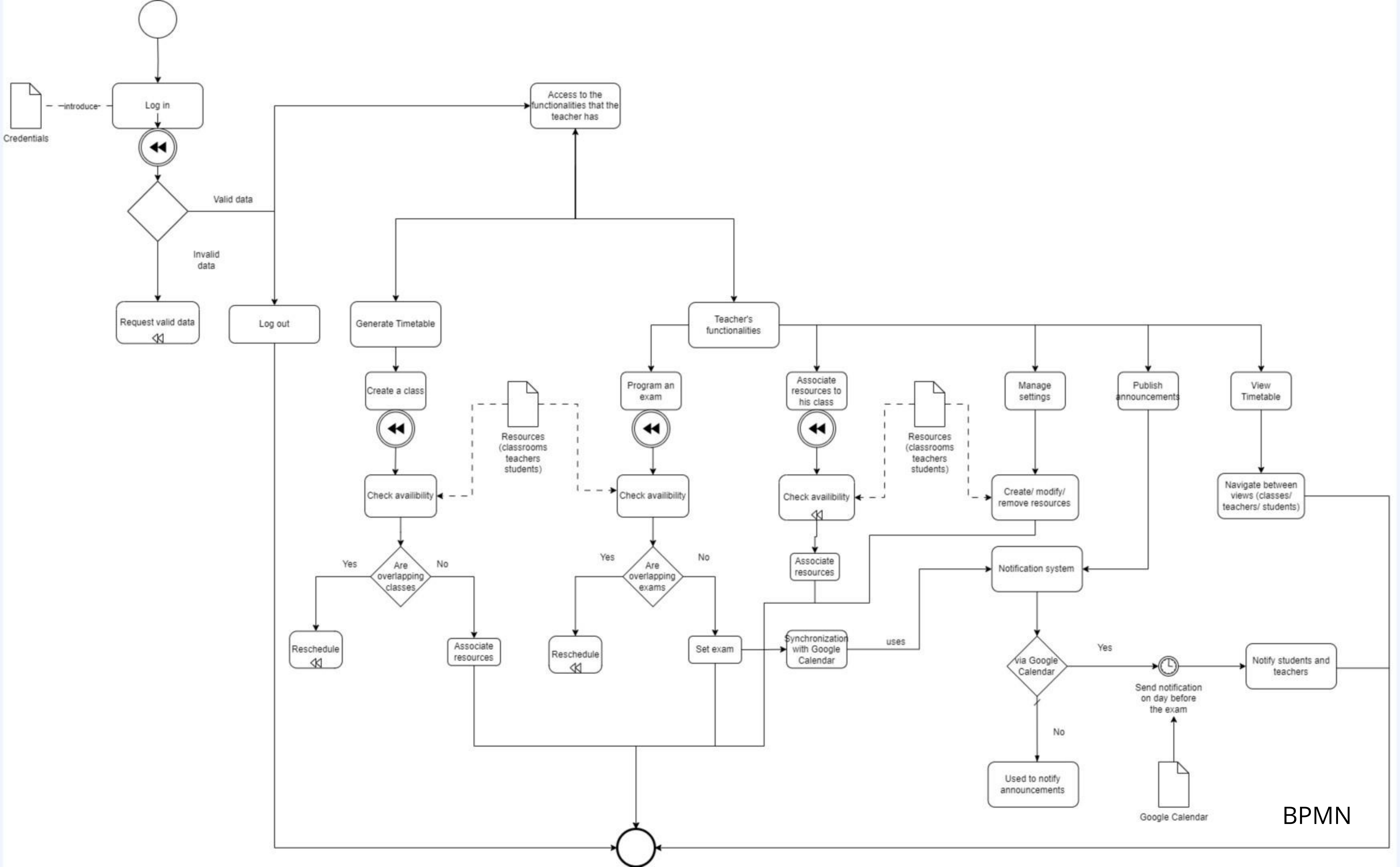
UML Diagrams

Business Process Modeling Notation

Eclipse


Teacher Admin
Manage settings
<<include>>
Create/ change/ remove users/ resources/ classes
Generate timetable
<<extend>>
Create a class
Associate resources
See overlapping classes
System Authentication
User Registration
Login
<<extend>>
<<include>>
Provide Name, Email
Teacher
Timetable visualization
Associate resources to his class
Publish announcements
<<extend>>
Notification generator
<<include>>
Program an exam
create a notification
send a notification
Synchronization of the timetable with the personal calendar
Google Calendar API

UML


Credentials
introduce
Log in
Valid data
Invalid data
Request valid data
Log out
Access to the functionalities that the teacher has
Generate Timetable
Teacher's functionalities
Create a class
Resources (classrooms teachers students)
Check availibility
Are overlapping classes
Yes
No
Reschedule
Associate resources
Program an exam
Check availibility
Are overlapping exams
Yes
No
Reschedule
Set exam
Associate resources to his class
Check availibility
Associate resources
Synchronization with Google Calendar
uses
Resources (classrooms teachers students)
Manage settings
Create/ modify/ remove resources
Publish announcements
Notification system
via Google Calendar
Yes
No
Used to notify announcements
Send notification on day before the exam
Google Calendar
Notify students and teachers
View Timetable
Navigate between views (classes/ teachers/ students)

BPMN

Eclipse Modeling Framework

We use the Eclipse Modeling Framework (EMF) to create Ecore diagrams, generating an EMF model with the help of the Ecore model. By running plugins, this triggers the appearance of a pop-up window where we create a new model. This process ensures an efficient and systematic approach to creating and working with complex domain models in our project.

AOP & MOP

AOP & MOP

In our project, we have adopted both **Aspect-Oriented Programming (AOP)** and **Monitoring-Oriented Programming (MOP).**

Aspect-Oriented Programming

Monitoring-Oriented Programming

AOP & security

Leveraging the aspectlib library in our project, we've integrated aspects into all 10 views: ***admin***, ***class***, ***classroom***, ***student***, ***resource***, ***schedule***, ***subject***, ***teacher***, ***auth***, and ***timetable***. This means that when we perform actions like adding a student or class, notifications promptly inform us of the changes, keeping us closely connected to the real-time dynamics of our project.

Let's look in the code

Security

Relevant example

The function *add_student* is defined as an aspect (**@add_student**), which means it can be used to add additional behavior to other functions.

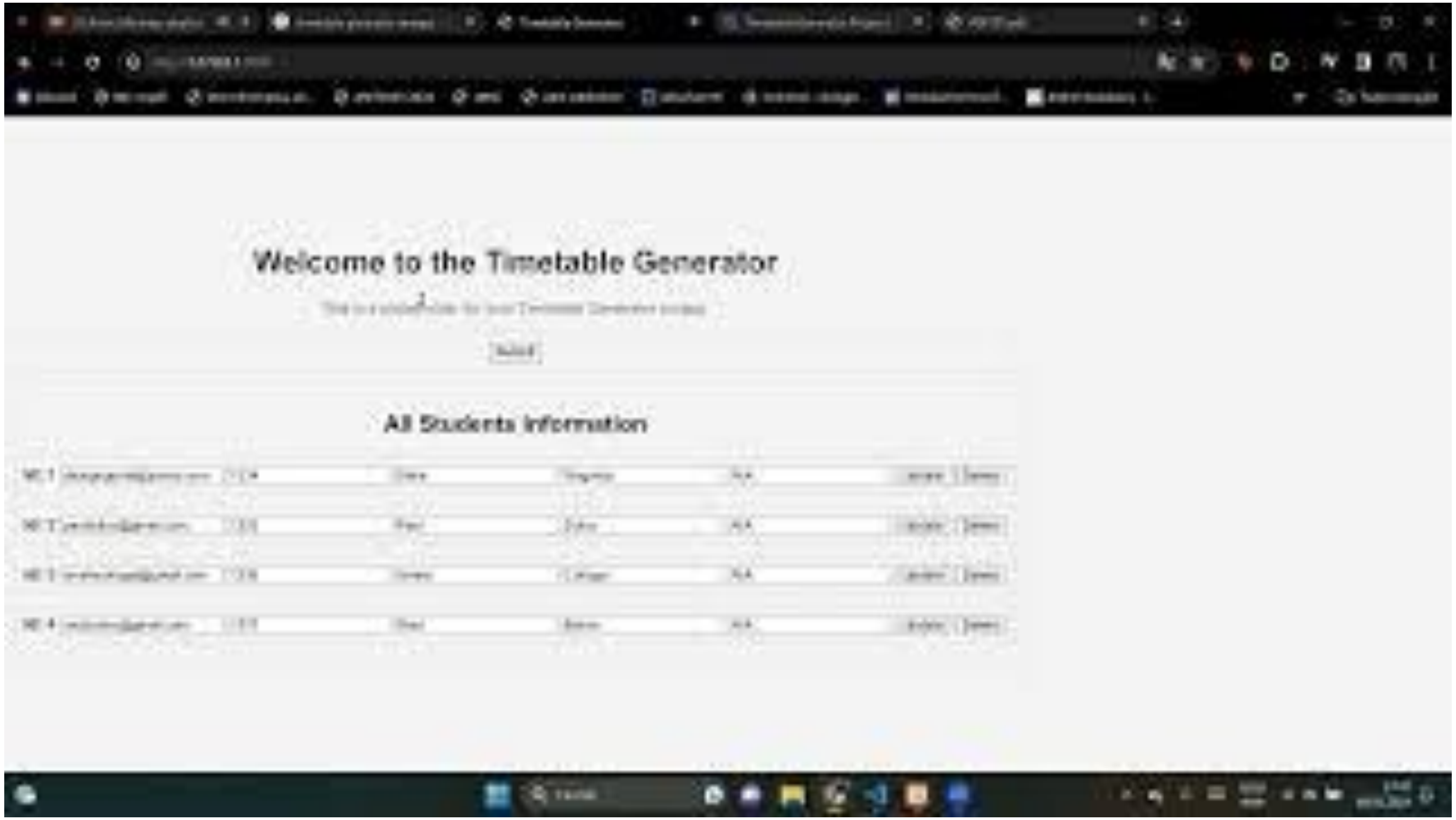
This aspect prints "Adding a student..." before the execution of the function it's applied to, and "Student added successfully." after the function has executed.

```
@aspectlib.Aspect
def add_student(*args, **kwargs):
   print("Adding a student...")
   result = yield aspectlib.Proceed
   print("Student added successfully.")
   return result
```

Security

Additionally, we've implemented a security aspect . With this aspect in place, when logged in, users(teachers) can add resources to their timetable. However, for unauthorized attempts, the system responds with an error status "401," ensuring robust security measures and controlled access to sensitive functionalities.

Let's see how it works!



Welcome to the Timetable Generator

[illegible]

All Students Information

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	NA	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	NA	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	NA	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	NA	[illegible]	[illegible]

Monitoring-Oriented Programming

In addition to aspect-oriented programming, we've introduced monitors in our project. Our specific monitor focuses on preventing overlapping courses when creating a schedule. This monitor not only alerts us to any overlapping classes but also dynamically modifies the schedule, ensuring that the conflicting courses are rescheduled to a free interval hour that aligns with our criteria and preferences.

this feature contributes to a more efficient, error-resistant, and user-friendly scheduling process in our project.

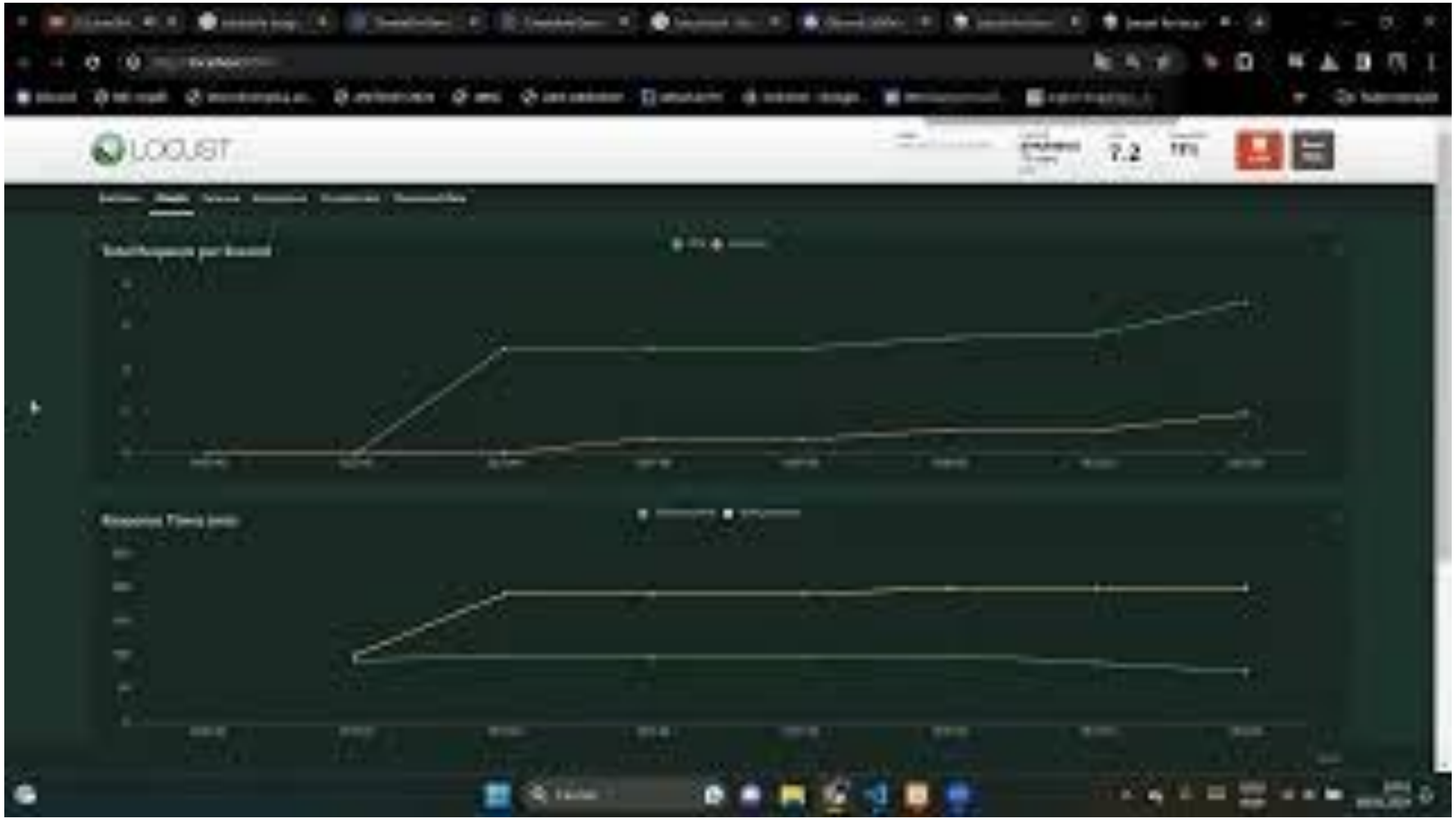
Functional & Non-functional testing

In our testing strategy, we've implemented both functional and non-functional testing.

Functional Testing:
We conducted thorough testing of essential functionalities, with a specific focus on our login function. This ensures that users can seamlessly and securely access our system, providing a foundation for a positive user experience.

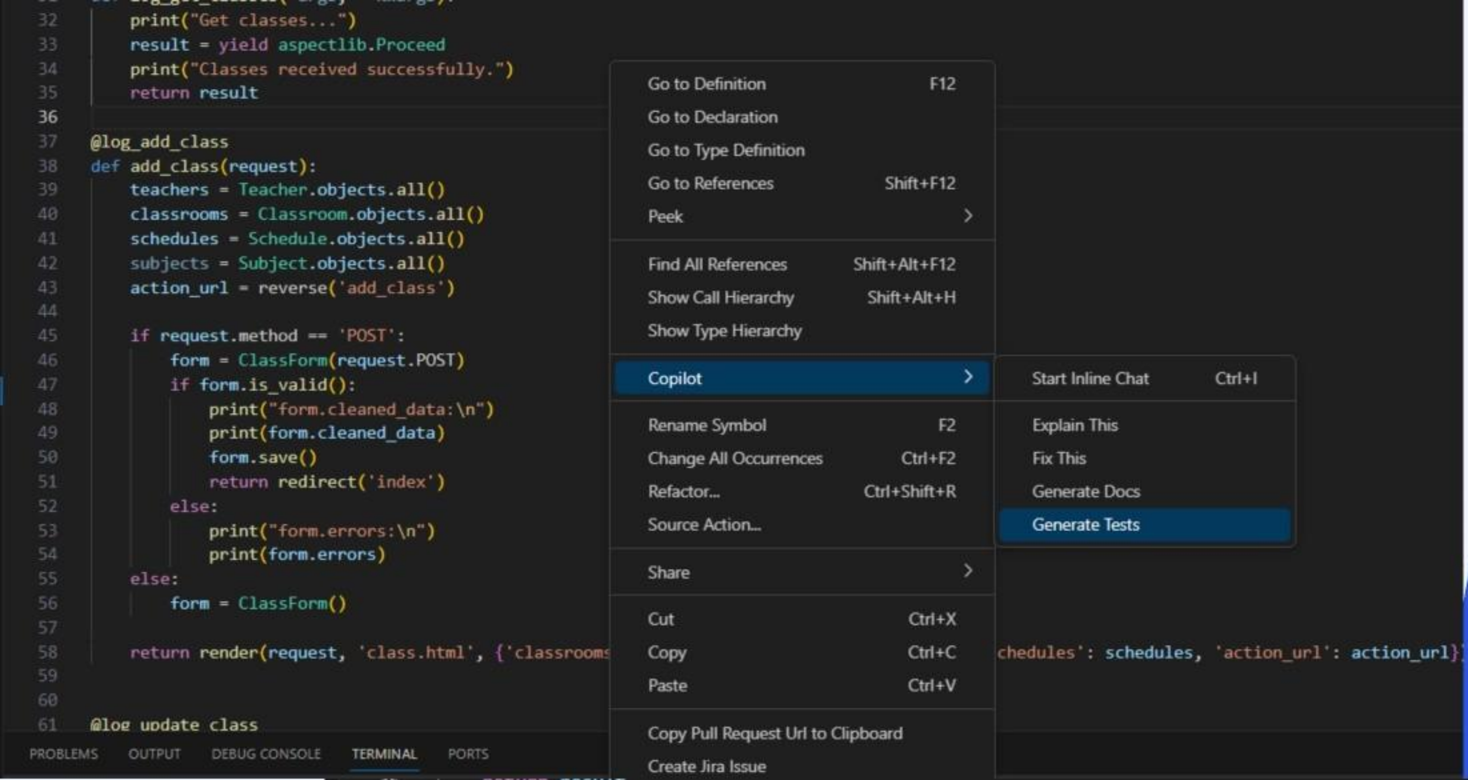
Non-Functional Testing:
For non-functional testing, we used a tool called Locust. Locust is a modern load testing framework that allows us to assess the performance and scalability of our system under various conditions. This ensures that our project can handle a substantial load of users, providing insights into its responsiveness, stability, and overall non-functional attributes.

Let's see how it's working

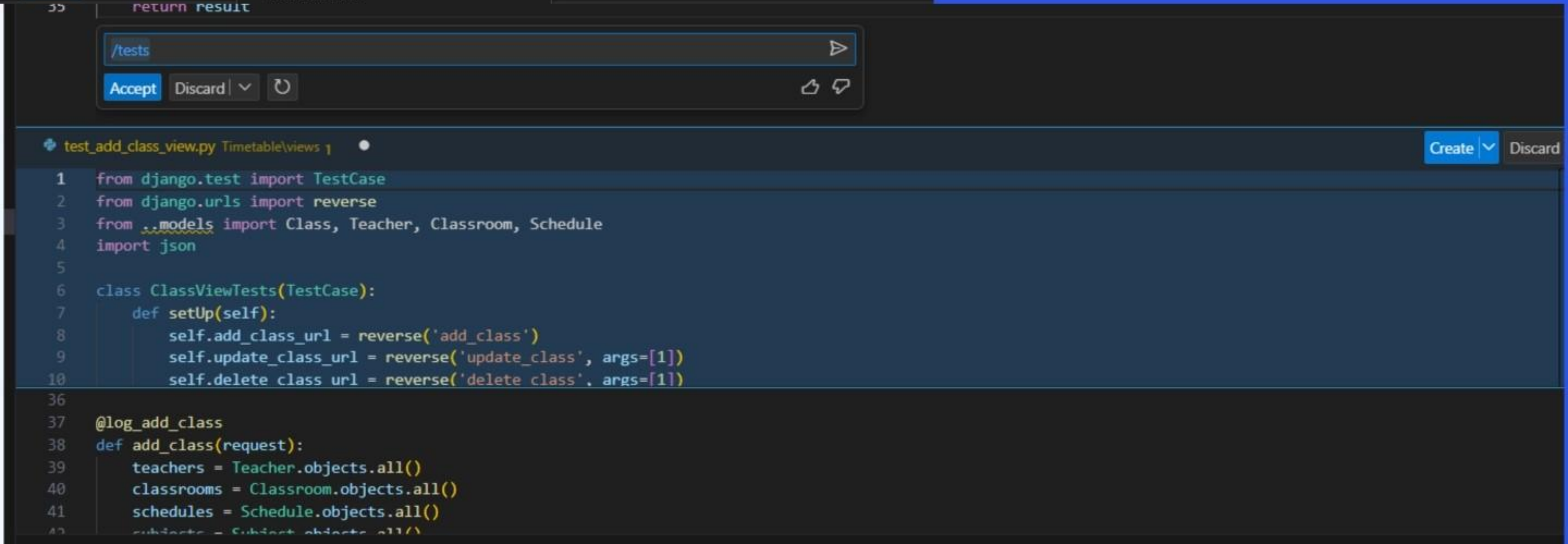

LOCUST
7.2
Total Requests per Second
Response Times (ms)

GitHub copilot

GitHub copilot



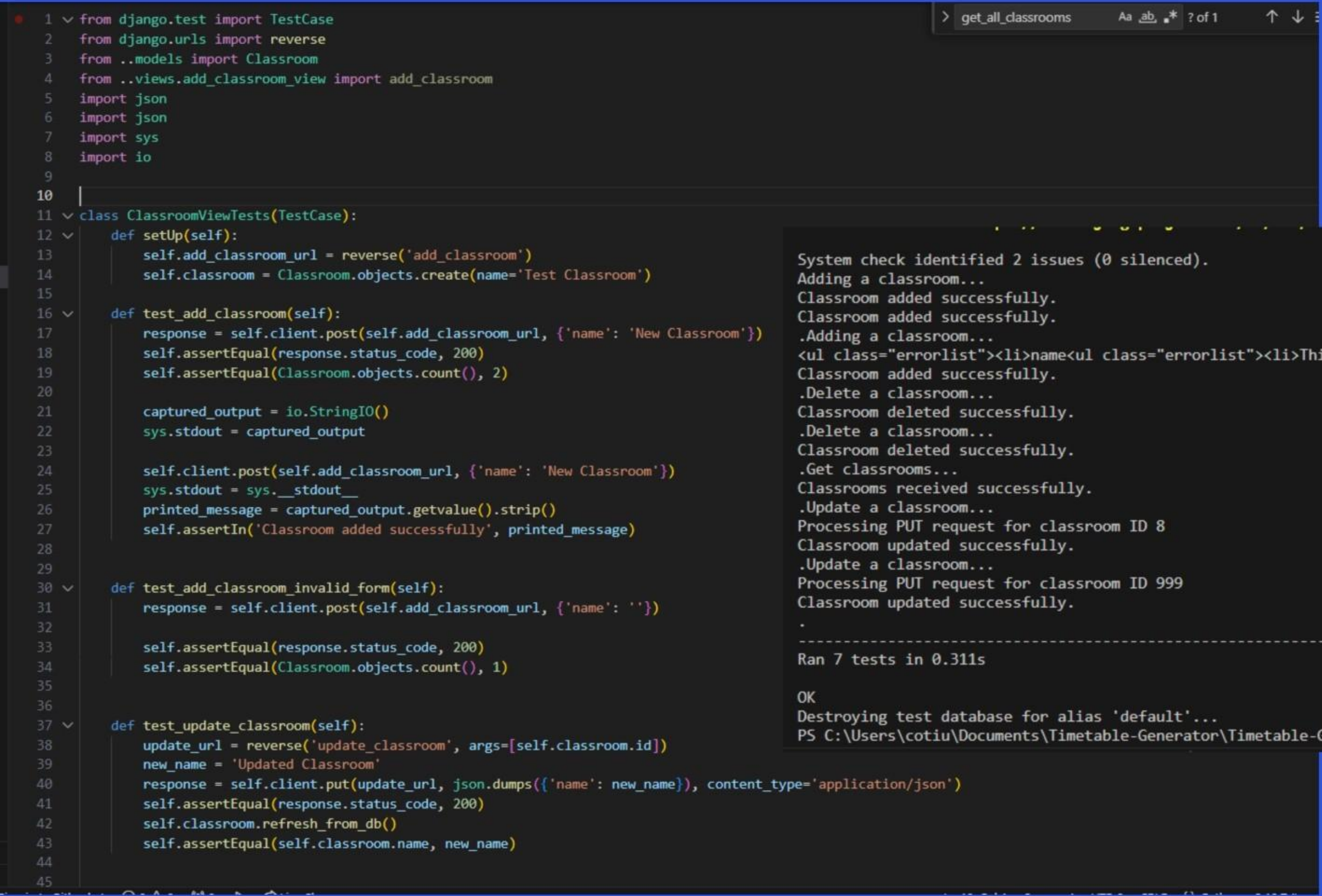
```python
    print("Get classes...")
    result = yield aspectlib.Proceed
    print("Classes received successfully.")
    return result

@log_add_class
def add_class(request):
    teachers = Teacher.objects.all()
    classrooms = Classroom.objects.all()
    schedules = Schedule.objects.all()
    subjects = Subject.objects.all()
    action_url = reverse('add_class')

    if request.method == 'POST':
        form = ClassForm(request.POST)
        if form.is_valid():
            print("form.cleaned_data:\n")
            print(form.cleaned_data)
            form.save()
            return redirect('index')
        else:
            print("form.errors:\n")
            print(form.errors)
    else:
        form = ClassForm()

    return render(request, 'class.html', {'classrooms chedules': schedules, 'action_url': action_url}

@log_update_class
```



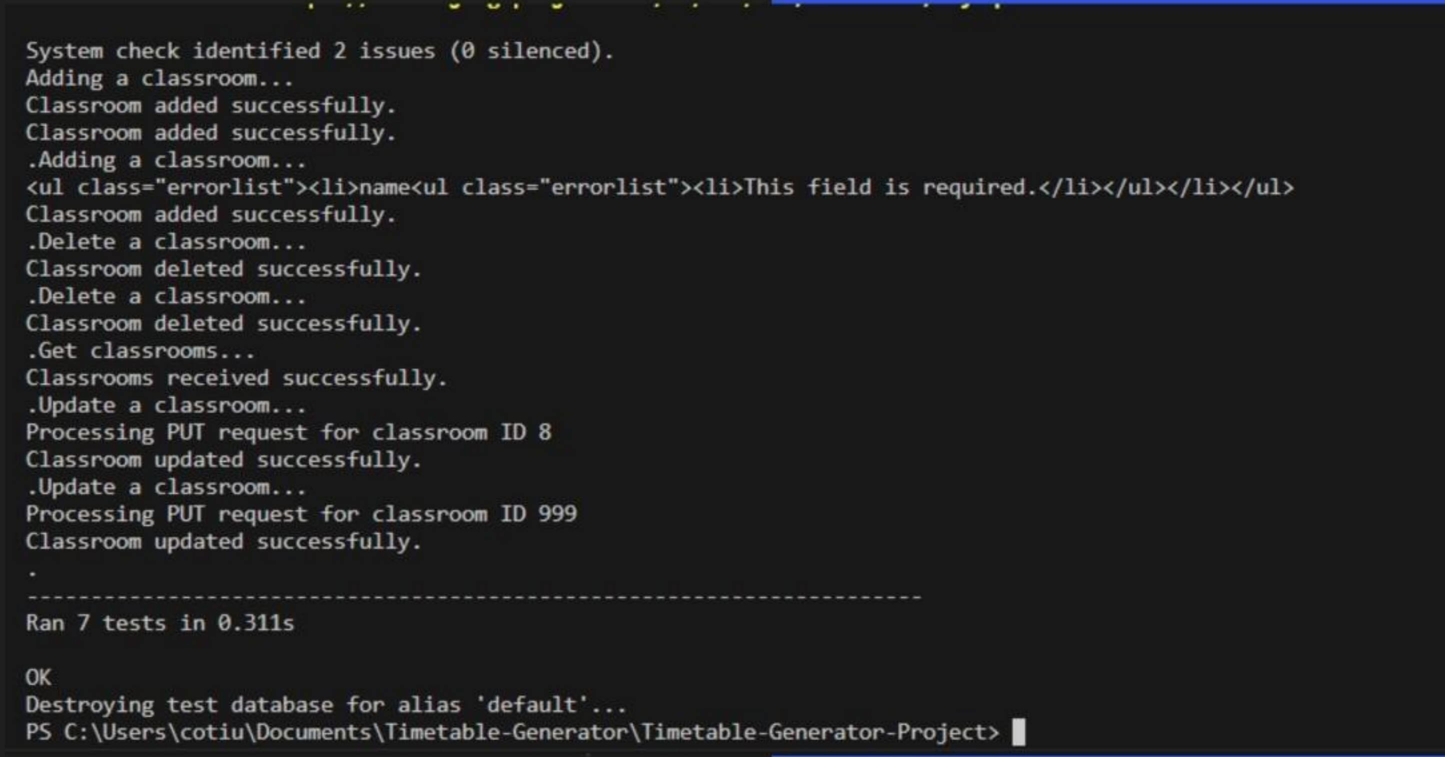
```python
    return result
```

```python
from django.test import TestCase
from django.urls import reverse
from ..models import Class, Teacher, Classroom, Schedule
import json

class ClassViewTests(TestCase):
    def setUp(self):
        self.add_class_url = reverse('add_class')
        self.update_class_url = reverse('update_class', args=[1])
        self.delete_class_url = reverse('delete_class', args=[1])
```

```python
@log_add_class
def add_class(request):
    teachers = Teacher.objects.all()
    classrooms = Classroom.objects.all()
    schedules = Schedule.objects.all()
```



```python
from django.test import TestCase
from django.urls import reverse
from ..models import Classroom
from ..views.add_classroom_view import add_classroom
import json
import json
import sys
import io


class ClassroomViewTests(TestCase):
    def setUp(self):
        self.add_classroom_url = reverse('add_classroom')
        self.classroom = Classroom.objects.create(name='Test Classroom')

    def test_add_classroom(self):
        response = self.client.post(self.add_classroom_url, {'name': 'New Classroom'})
        self.assertEqual(response.status_code, 200)
        self.assertEqual(Classroom.objects.count(), 2)

        captured_output = io.StringIO()
        sys.stdout = captured_output

        self.client.post(self.add_classroom_url, {'name': 'New Classroom'})
        sys.stdout = sys.__stdout__
        printed_message = captured_output.getvalue().strip()
        self.assertIn('Classroom added successfully', printed_message)


    def test_add_classroom_invalid_form(self):
        response = self.client.post(self.add_classroom_url, {'name': ''})

        self.assertEqual(response.status_code, 200)
        self.assertEqual(Classroom.objects.count(), 1)


    def test_update_classroom(self):
        update_url = reverse('update_classroom', args=[self.classroom.id])
        new_name = 'Updated Classroom'
        response = self.client.put(update_url, json.dumps({'name': new_name}), content_type='application/json')
        self.assertEqual(response.status_code, 200)
        self.classroom.refresh_from_db()
        self.assertEqual(self.classroom.name, new_name)
```



```
System check identified 2 issues (0 silenced).
Adding a classroom...
Classroom added successfully.
Classroom added successfully.
.Adding a classroom...
<ul class="errorlist"><li>name<ul class="errorlist"><li>This field is required.</li></ul></li></ul>
Classroom added successfully.
.Delete a classroom...
Classroom deleted successfully.
.Delete a classroom...
Classroom deleted successfully.
.Get classrooms...
Classrooms received successfully.
.Update a classroom...
Processing PUT request for classroom ID 8
Classroom updated successfully.
.Update a classroom...
Processing PUT request for classroom ID 999
Classroom updated successfully.
.
----------------------------------------------------------------------
Ran 7 tests in 0.311s

OK
Destroying test database for alias 'default'...
PS C:\Users\cotiu\Documents\Timetable-Generator\Timetable-Generator-Project>
```

Summary

The Timetable Generator project excels in scheduling and project planning using advanced technologies like Monitoring-Oriented Programming (MOP) and Aspect-Oriented Programming (AOP). It ensures a dynamic and secure user experience with BPMN, focusing on modularity, maintainability, and security. The project proactively addresses scheduling conflicts through MOP, and extensive testing validates scalability. Looking ahead, it lays the foundation for future advancements, exploring potential optimizations with genetic algorithms and constraint fulfillment.