DepartmentDirect: Subscription and Event Notification System

Saeideh Kalantari
Dept. of Electrical and Computer Engineering
Concordia University
sdhkalantari@gmail.com

Mobina BonkarDept. of
Concordia University
mobina.bonkar@mail.concordia.ca

Abstract—**This report presents the implementation of subscription management and event notification publishing as a part of the Q&A digital platform called DepartmentDirect to enhance the experience of future applicants and their families during Concordia university's open house events. This report explains software specifications, user stories, and outcomes of the project. In addition, it investigates the rubric sores for two AI suggested prompts and compares them to our proposed prompt.**

Keywords—Digital Platform, Subscription form, Prompt, Rubric score

I. Introduction

Concordia's open house event has a significant role in attracting prospective students. DepartmentDirect is a digital platform that enhances the experience of future applicants and their families during this even. This report investigates the implementation of subscription management and event notification publishing as a part of DepartmentDirect to provide the possibility of subscription to departments updates by the future students and that of publishing event notifications by departments staff.

II. Rubric Definition

A rubric is a type of scoring that assesses specific components and expectations for an assignment, a proposal, etc.

In this section three prompts generated by Copilot, GPT, and us are sored using a defined rubric table to compare these prompts.

In this report, some aspects are considered to define the rubric:

• Are the user stories independent from each other?
• Are they clear to define tests and criteria to verify them?
• Are they properly defined without over the scope? (They state what is needed but not how, not specific details)

A prompt has to be provided for a scenario of a digital platform which is called "DepartmentDirect". This scenario addresses the subscription for event updates by future applicants and publishing event notifications by departments staff. The prompt should describe the context, features, outcomes, process steps, and audience. The objective is to score the prompts generated by two AI agents, and compare them to the scores of our prompt.

Table 1, demonstrates our proposed rubric scoring table for the mentioned prompts. This table simplifies scoring the prompts using clear metrics.

A. Generated Prompt from Copilot

Epic: Subscribe to Future Updates and Publish Event Notifications

Context:
Future applicants need a way to stay informed about department events and updates. Departments need a streamlined method to publish notifications and updates.

Objective:
Enable future applicants to subscribe to updates and receive notifications about events, and allow departments to publish these notifications efficiently.

Structure:
• Feature: Subscription to updates
• Feature: Event notification publishing
• Feature: Notification management

User Stories:

User Story 1: Subscribe to Updates

• Context: Future applicants want to stay informed about department events and updates.
• Objective: Allow users to subscribe to updates.
•Structure: Subscription form, confirmation, and management.
• Tasks:
  o Design a subscription form.
  o Implement email verification.
  o Create a subscription management interface.
• Actions:
  o As a future applicant, I want to subscribe to department updates so that I can stay informed about events and news.
  o As a future applicant, I want to manage my subscription preferences so that I can control the type of updates I receive.
• Results:
  o Users can subscribe to updates.

Criteria	Excellent (5 points)	Good (4 points)	Acceptable(3 points)	Needs refinement (2 points)	Unacceptable (1 point)
Independence of Features and User Stories					
Are the features independent from each other?	Completely independent, no overlap	Highly independent, a few overlap	Some overlap, but mostly separate	Lots of overlap, some independence	Not independent, fully overlapping
Are the user stories independent from each other?	Completely independent, no overlap	Highly independent, small overlap	Some overlap, but mostly separate	Considerable overlap, some independence	Not independent, fully overlapping
Clarity in Defining Tests and Criteria for Verification					
Is the context clear for each user story?	Extremely clear, easy to understand	Mostly clear, small ambiguities	Somewhat clear, some ambiguities	Mostly unclear, significant ambiguities	Very unclear, hard to understand
Is the objective clear for each user story?	Extremely clear, easy to understand	Mostly clear, small ambiguities	Somewhat clear, some ambiguities	Mostly unclear, significant ambiguities	Very unclear, hard to understand
Are the tasks specific and testable for each user story?	Extremely specific and testable	Mostly specific and testable, a few issues	Somewhat specific and testable, some issues	Mostly not specific and testable, lots of issues	Not specific and not testable at all
Are the actions and results measurable for each user story?	Extremely measurable	Mostly measurable, a few issues	Somewhat measurable, some issues	Mostly not measurable, lots of issues	Not measurable at all
Proper Definition Without Over the Scope					
Do the tasks avoid specifying implementation details?	Completely avoids, very high level	Mostly avoids, small details	Somewhat avoids, some specifics	Mostly specifies, significant details	Fully specifies, very detailed
Is the scope limited to the necessary functionalities?	Extremely focused, no extra functionalities	Mostly focused, a few extra functionalities	Somewhat focused, some extra functionalities	Mostly unfocused, significant extra functionalities	Completely unfocused, many extra functionalities

Table 1. Rubric Definition

o Users receive a confirmation email.
o Users can manage their subscription preferences.

User Story 2: Publish Event Notifications

- Context: Departments need to inform future applicants about upcoming events.
- Objective: Allow departments to publish event notifications.
- Structure: Event creation, notification scheduling, and distribution.
- Tasks:
  o Design an event creation interface.
  o Implement notification scheduling.
  o Create a distribution system for notifications.
- Actions:
  o As a department admin, I want to create and schedule event notifications so that future applicants are informed about upcoming events.
  o As a department admin, I want to manage published notifications so that I can update or cancel events if needed.
- Results:
  o Departments can create and schedule event notifications.
  o Notifications are sent to subscribed users.
  o Admins can manage existing notifications.

User Story 3: Notification Management

- Context: Both users and departments need to manage notifications effectively.
- Objective: Provide interfaces for managing notifications.
- Structure: User notification settings, admin notification dashboard.
- Tasks:
  o Design user notification settings interface.
  o Implement admin notification dashboard.
  o Create notification history logs.
- Actions:
  o As a future applicant, I want to view my notification history so that I can keep track of past updates.
  o As a department admin, I want to view and manage all sent notifications so that I can ensure accurate communication.
- Results:
  o Users can view and manage their notification history.
  o Admins have a dashboard to manage all notifications.
  o Notification history is logged for reference.

Table 2 shows the rubric score of the above explained prompt. In overall, User stories and features demonstrate high level of independence with minor overlap. Overall, all user stories are clear though minor refinements are required in specificity and scope definition of publish event notification and notification management.

Features	Independence Assessment	Explanation
Subscribe to Updates	Highly independent, a few overlaps (4)	The subscription process is independent from event notification publishing but may share some interface elements or user preferences.
Publish Event Notifications	Highly independent, a few overlaps (4)	Event notification publishing has different capabilities from subscription, while in both of them user interaction and messaging systems are required.
Notification Management	Highly independent, a few overlaps (4)	Notification management only concentrates on managing notifications, it is independent from both subscription and event publishing.

Features	Context Clarity	Objective Clarity	Tasks Specificity and Testability	Actions and Results Measurability	Proper Definition Without Over the Scope	Scope Limited to Necessary Functionalities
Subscribe to Updates	Ext. clear, easy to understand (5)	Ext. clear, easy to understand (5)	Ext. spec. and testable (5)	Ext. measurable (5)	Comp.avoids, very high level (5)	Ext. focused, no extra functionalities (5)
Publish Event Notifications	Mostly clear, small ambiguities (4)	Mostly clear, small ambiguities (4)	Mostly specific and testable, a few issues (4)	Mostly measurable, a few issues (4)	Mostly avoids, small details (4)	Mostly focused, a few extra functionalities (4)
Notification Management	Mostly clear, small ambiguities (4)	Mostly clear, small ambiguities (4)	Mostly specific and testable, a few issues (4)	Mostly measurable, a few issues (4)	Mostly avoids, small details (4)	Somewhat focused, some extra functionalities (3)

Table 2. Rubric scores of Copilot's prompts

B. Generated Prompt from ChatGPT

Epics and User Stories

Epic 1: Subscription Management

- As a future applicant, I want to subscribe to updates so that I can receive notifications about events and news.
  - o User Story 1.1: As a future applicant, I want to see a subscription form on the homepage so that I can sign up for updates easily.
  - o User Story 1.2: As a future applicant, I want to choose the types of updates I receive (e.g., event notifications, news updates) so that I only get relevant information.
  - o User Story 1.3: As a future applicant, I want to manage my subscription preferences so that I can update my choices or unsubscribe if needed.

Epic 2: Event Notification Publishing

- As a department staff member, I want to publish event notifications so that future applicants are informed about upcoming events.
  - o User Story 2.1: As a department staff member, I want to create and schedule event notifications so that they are automatically sent to subscribers at the appropriate time.
  - o User Story 2.2: As a department staff member, I want to customize the content of event notifications so that they are specific to my department's events.
  - o User Story 2.3: As a department staff member, I want to view the list of scheduled notifications so that I can manage and update them as needed.

Prioritization

1. High Priority: User Story 1.1, User Story 1.2, User Story 2.1
2. Medium Priority: User Story 1.3, User Story 2.2
3. Low Priority: User Story 2.3

Prompts for Creating Epics and User Stories

Epic 1: Subscription Management

- Context: "You are defining the features for subscription management in DepartmentDirect."
- Objective: "Create epics and user stories that allow future applicants to subscribe to updates and manage their preferences."
- Structure: "Break down the subscription process into specific tasks: viewing the subscription form, selecting update types, managing preferences."
- Tools: "Project management tools like Jira or Trello."
- Audience: "Future applicants who will subscribe to updates."
- Relevance: "Ensuring users can easily subscribe to relevant updates will improve engagement with DepartmentDirect."

Epic 2: Event Notification Publishing

- Context: "You are defining the features for publishing event notifications in DepartmentDirect."
- Objective: "Create epics and user stories that allow department staff to publish and manage event notifications."
- Structure: "Break down the notification process into specific tasks: creating and scheduling notifications, customizing content, managing scheduled notifications."
- Tools: "Project management tools like Jira or Trello."
- Audience: "Department staff who will publish notifications."
- Relevance: "Ensuring staff can efficiently publish event notifications will keep future applicants informed and engaged."

Table 3 describes the rubric scores of the prompt generated by ChatGPT. This tables shows that user stories and features demonstrate high level of independence with minor overlap. Overall, all user stories are clear though minor refinements are required in specificity and scope definition of publish event notification and notification management.

Criteria	Independence Assessment	Explanation
Subscription Management *Features: viewing the subscription form, selecting update types, managing preferences.	Highly independent, a few overlap (4)	The subscription form and preferences management are distinct functionalities, but they may have minor overlap.
Event Notification Publishing *Features: Event creation, Notification scheduling, Distribution system	Highly independent, a few overlap (4)	The event creation and notification management are distinct functionalities, but they may have minor overlap.

Features	Context Clarity	Objective Clarity	Tasks Specificity and Testability	Actions and Results Measurability	Proper Definition Without Over the Scope	Scope Limited to Necessary Functionalities
Subscription Management	Mostly clear, small ambiguities (4)	Mostly clear, small ambiguities (4)	Mostly specific and testable, a few issues (4)	Mostly measurable, a few issues (4)	Mostly avoids, small details (4)	Mostly focused, a few extra functionalities (4)
Event Notification Publishing	Mostly clear, small ambiguities (4)	Mostly clear, small ambiguities (4)	Mostly specific and testable, a few issues (4)	Mostly measurable, a few issues (4)	Mostly avoids, small details (4)	Mostly focused, a few extra functionalities (4)

Table 3. Rubric Scores of ChatGPT's Prompts

C. Our proposed Prompt

Key features of the subscription management and event notification publishing platform are as the following:

1- ***Scenario 1:***

This scenario deals with future applicants who want to subscribe to departments updates and management of their subscription preferences.

2- ***Scenario 2:***

This scenario deals with departments staff who want to publish event notifications and updates.

Epics and User Stories

This section discusses the epics of the project and their user stories.

Epic 1: Subscription Management:

User story1- Subscribing to Updates

As a Future student, I would like to subscribe updates and event notifications, so that I can be informed about important updates and upcoming events of the university departments.

Acceptance Criteria:

- A subscription form must be available to enter email address by a future student.
- After submission the subscription form, the subscriber receives a verification email.
- The subscriber's email address is stored in a subscription list.

User story2- Email Verification

As a Future student, I would like to verify my email address, so that I can confirm my subscription.

Acceptance Criteria:

- The subscriber receives an email containing a verification link.
- After clicking the verification link, the subscriber's email address is marked as verified.

User story 3- Managing Subscription Preferences

As a future applicant, I want to manage my subscription preferences so that I can control the type of updates I receive.

Acceptance Criteria:

- The subscriber can access a preferences page after logging in.
- The preferences are stored and updated in a database.

User story 4- Unsubscribing from Notifications

As a future applicant, I want to be able to cancel my subscription from notifications and updates, so that I no longer receive notifications.

Acceptance Criteria:

- The subscriber receives an unsubscribe link by email.
- After clicking on the unsubscribe link, the email address is removed from the subscription list.

Epic 1: Event Notification Publishing

User story 1- Creating Event Notification

As a department staff, I want to be able to create notifications so that I can inform subscribers about upcoming events or update news.

Acceptance Criteria:

- A form is available to enter a notification title, contents, etc.
- The system sends the notification at the scheduled time.

User story 2- Managing Event Notifications

As a department staff, I want to be able to manage notifications so that I can update event details or cancel an event.

Acceptance Criteria:

- Department staff can see a list of published notifications.
- Department staff can edit the details of a scheduled notification.

Prompt for our proposed epics and user stories.

- Prompt for "Subscription management":

Context: You are a software developer who is creating the possibility of subscription management in "DepartmentDirect".

Outcomes: Users can subscribe to updates; Users can manage their subscription preferences; Users can unsubscribe from notifications.

Steps: Breakdown subscription process into request for submitting a subscription form, receiving a verification email, managing notification preferences, and unsubscribing option.

Tools: Jira for project management integrated with GitHub repository

Audience: Future applicants who subscribe to update notifications.

Relevance: Ensuring applicants can easily subscribe or unsubscribe to/from updates and manage their subscription preferences.

- Prompt for "Event Notification Publishing":

Context: You are a software developer who is creating the possibility of publishing notifications in "DepartmentDirect".

Outcomes: create event notifications, Notifications are sent to subscribed users, Departments staff can manage existing notifications.

Steps: Breakdown publishing notifications into create new notifications, managing previously created notifications

Tools: Jira for project management integrated with GitHub repository

Audience: Department staff who publish update notifications.

Relevance: Ensuring departments staff can easily publish and manage event notifications

Table 4 demonstrates the rubric scores of our proposed prompt.

Feature	Independence Assessment	Justification
Subscription Management	Highly independent, minimal overlap (4)	Subscription management focuses on user actions related to subscribing and managing preferences, distinct from event notification publishing.
Publishing Event Notifications	Highly independent, minimal overlap (4)	Event notification publishing involves tasks related to creating and managing notifications, separate from subscription management.

Feature	Context Clarity	Objective Clarity	Tasks Specificity and Testability	Actions and Results Measurability	Proper Definition Without Over the Scope	Scope Limited to Necessary Functionalities
Subscription Management	Mostly clear, minor ambiguities (4)	Mostly clear, minor ambiguities (4)	Mostly specific and testable, a few issues (4)	Mostly measurable, a few issues (4)	Mostly avoids, minor specifics (4)	Mostly focused, minor extra functionalities (4)
Publishing Event Notifications	Mostly clear, minor ambiguities (4)	Mostly clear, minor ambiguities (4)	Mostly specific and testable, a few issues (4)	Mostly measurable, a few issues (4)	Mostly avoids, minor specifics (4)	Mostly focused, minor extra functionalities (4)

Table 4. Rubric scores of our prompt

Based on the rubric table, both "subscription management" and the" event notification publishing" have high level of independence. Also, both epics are mostly clear and require a few improvements in specificity and scope definition.

To answer part 4 of the assignment, one of the user stories have been broken down into three tasks which one of them is non-functional.

creating 3 satisfactions for this user story: 2- As a future applicant, I want to manage my subscription preferences so that I can control the type of updates I receive.

4-1- Subscribers can easily access their subscription preferences, so that they are able to select relevant updates as notifications.

4-2- Future applicants have the flexibility in managing their preferences at any time.

4-3- System should ensure the preferences updating and confirming changes in less than 500 ms even the network traffic is very high.

III. Implementation

For implementation of this assignment, some technologies have been used such as express.js, SQLite, and Nodemailer.

1- Express.js:

Express.js is a web application framework that provides a set of features for web applications. This technology allows you to define to handle HTTP requests. It provides a straightforward way to define routes for an application.

Express.js easily integrates with various databases and other web frameworks. Supports asynchronous request handling, making it suitable for high-performance applications.

In this assignment, Express.js is used to defined routes for handling subscription and verification requests. It utilizes body-parser for parsing JSON request bodies and cors for handling cross-origin requests.

2- SQLite:

SQLite is a database that doesn't require a separate server process and allows access to the database using a nonstandard variant of the SQL query language.

By using SQLite all the data is stored in a single file, and makes it easy to deploy. This database doesn't require a server to operate which reduces overhead. Besides, it doesn't need to set up or administration.

In this assignment, SQLite is used for storing subscriber data, including email addresses and verification statuses. It provides a simple schema to create and manage the subscribers table and the events table.

3- Nodemailer:

Nodemailer is a module for Node.js applications to allow sending mail easily. It is designed to be easy to set up and use. It Supports multiple transport methods, including SMTP, AWS SES, and custom transports.

In this assignment, Nodemailer technology is used to send verification emails to subscribers, to set up email transport using Gmail's SMTP service, and to create email templates for verification and notification messages.

Also, Jira has been used as the project management tool and it has been integrated to a repository on GitHub. The link to our repository is : https://github.com/sdklntri/coen-6311-summer-2024-demo

Screen shots of our project in Jira are illustrated in the appendix. Table 5 is a demonstration of overall rubric scores of all prompts explained in this report. Additionally, cosine similarity norm has been calculated for considering the similarity between these prompts.

User story	Agent 1 definition	Agent 2 definition	our final definition
Norm Rubric Score	4/5	4/5	4/5
Norm Cosine Similarity	With Agent 2: 0.677 with my prompt: 0.661	With Agent 1: 0.677 with my prompt: 0.684	with GPTt prompt: 0.684 with copilot prompt: 0.661

Table 5. Overall Rubric scores and cosine similarity norm

IV. C4 Model

In this section three levels of C4 model diagrams are demonstrated for event-subscription-notification scenario of the DepatmentDirect platform. Figures 1, 2, and 3 respectively show context, container, and component diagrams of this scenario.

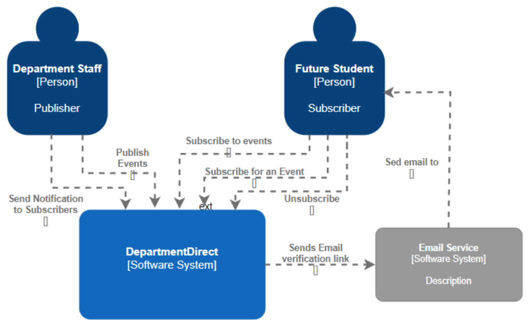


Figure 1. Context Diagram for Event-Subscription-Notification Scenario

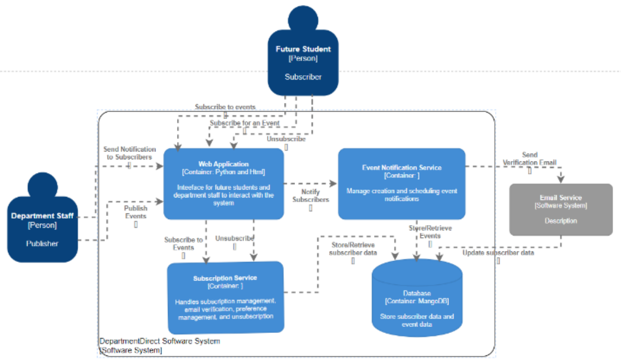

Future Student
[Person]
Subscriber
Subscribe to events
Subscribe for an Event
Unsubscribe
Send Notification to Subscribers
Department Staff
[Person]
Publisher
Publish Events
Web Application
[Container: Python and Html]
Inteeface for future students and department staff to interact with the system
Notify Subscribers
Event Notification Service
[Container:]
Manage creation and scheduling event notifications
Send Verification Email
Email Service
[Software System]
Description
Subscribe to Events
Unsubscribe
Store/Retrieve subscriber data
Store/Retrieve Events
Update subscriber data
Subscription Service
[Container:]
Handles subscription management, email verification, preference management, and unsubscription
Database
[Container: MangoDB]
Store subscriber data and event data
DepartmentDirect Software System
[Software System]

Figure 2. Container Diagram for Event-Subscription-Notification Scenario

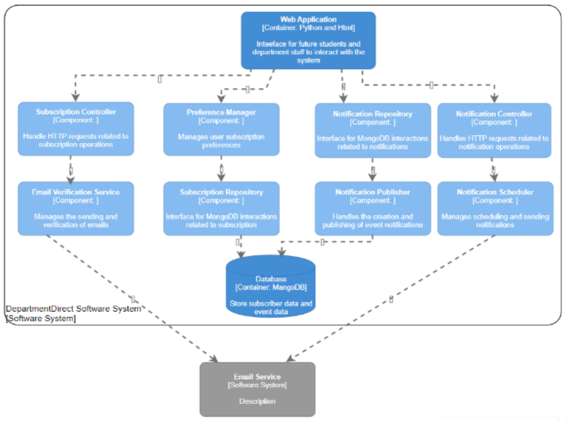

Web Application
[Container: Python and Html]
Inteeface for future students and department staff to interact with the system
Subscription Controller
[Component:]
Handle HTTP requests related to subscription operations
Preference Manager
[Component:]
Manages user subscription preferences
Notification Repository
[Component:]
Interface for MongoDB interactions related to notifications
Notification Controller
[Component:]
Handles HTTP requests related to notification operations
Email Verification Service
[Component:]
Manages the sending and verification of emails
Subscription Repository
[Component:]
Interface for MongoDB interactions related to subscription
Notification Publisher
[Component:]
Handles the creation and publishing of event notifications
Notification Scheduler
[Component:]
Manages scheduling and sending notifications
Database
[Container: MangoDB]
Store subscriber data and event data
DepartmentDirect Software System
[Software System]
Email Service
[Software System]
Description

Figure 3. Component Diagram for Event-Subscription-Notification Scenario

V. Development

This project followed the Agile methodology, enabling iterative progress and continuous feedback. The development was divided into two sprints, each focusing on specific features outlined in the earlier sections of this report.

The development process utilized tools such as Visual Studio as the IDE, project management application like JIRA, version control systems like Git, and programming languages such as Java, Python, HTML.

Requirements were identified through meetings with team members and discuss the needs of potential users. The gathered requirements were documented and prioritized based on their impact on the project goals.

A- Design Architecture

The system was designed using MVC architecture to separate the concerns, achieve modularity, and enhance maintainability.

In our project, modules like Subscription Management, Event Notification, and User Interface are developed independently which helps to reduce the impact of changes in one part on the entire system. This architecture facilitates the unit testing and improves flexibility and scalability of the project.

1- *Model:*

The project's Model contains the main organizational and cognitive aspects of business logic and data. It manages the state of the application and interacts with the database. There are two databases in this project, Events and subscribers. The subscription and Event classes handle data validation and store subscription and event details.

2- *View:*

The View is responsible for presenting data to the users. We implemented the View using HTML template which dynamically renders content based on the data provided by the Model. View in our system include a subscription form.

3- *Controller*

The Controller acts as a link between the Model and the View. It processes user inputs, such as form submissions, and updates the Model accordingly. For instance, upon subscribing for updates by a user, input validation follows by the Controller. It updates the subscription list in the Model and handles the View to render the confirmation message.

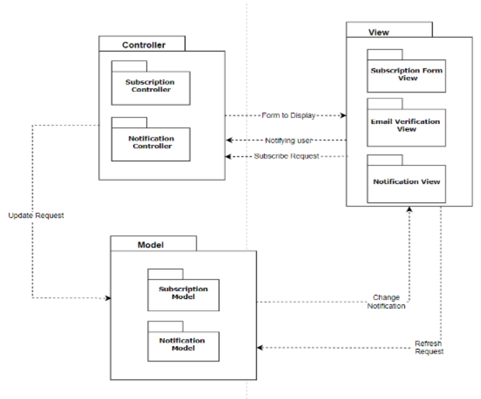


Figure 4. Architecture Design for Event-Subscription-Notification System

Figure 4 demonstrates architecture design of our project. In this figure:

- The View consists of three elements; 1- Subscription form view which allows future students to enter their email address to subscribe.
  2- Email verification view that shows the verification status.
  3- Notification management view which allows department staff to create and manage event notifications.
- The Controller is made up of two parts; 1- Subscription controller to handle user requests related to subscription.
  2- Notification controller to manage the actions related to notifications.is the Application Layer.
- The Model includes two model blocks; 1- Subscription model to manage subscription data such as storing and retrieving information.
  2- Notification model to handle data of event notifications such as content and schedules.
  Data is stored in two databases called subscribers and events. MongoDB was employed as the database to store the required data for this project. MongoDB is a document-oriented database program that utilizes JSON-like documents.

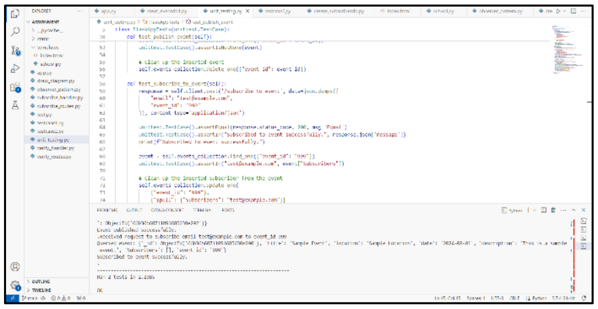
B- Implementation and Testing

The implementation was conducted using coding standards and Codes were reviewed regularly to ensure code quality.

Unit testing was performed on the system to ensure that each component is working correctly both in isolation and as part of the larger system.

Pyunit (Unittest) framework was employed to automate the testing process. JIRA helped us to track bugs and follow a systematic approach to prioritize and fix issues. This process played an important role in maintaining the system's reliability and performance.

Figure 5 shows the results of the unit testing. Each module, has been evaluated to ensure that every part of the system works properly.



```
class FlaskAppTests(unittest.TestCase):
    def test_publish_event(self):
        unittest.TestCase().assertIsNotNone(event)

        # clean up the inserted event
        self.events_collection.delete_one({"event_id": event_id})

    def test_subscribe_to_event(self):
        response = self.client.post('/subscribe_to_event', data=json.dumps({
            "email": "test@example.com",
            "event_id": "999"
        }), content_type='application/json')

        unittest.TestCase().assertEqual(response.status_code, 200, msg='Equal')
        unittest.TestCase().assertIn("Subscribed to event successfully.", response.json['message'])
        print(f"Subscribed to event successfully.")

        event = self.events_collection.find_one({"event_id": "999"})
        unittest.TestCase().assertIn("test@example.com", event["Subscribers"])

        # clean up the inserted subscriber from the event
        self.events_collection.update_one(
            {"event_id": "999"},
            {"$pull": {"subscribers": "test@example.com"}}
```

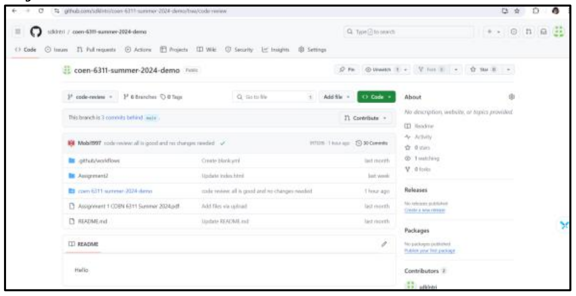
```
': ObjectId('[illegible]')})
Event published successfully.
.Received request to subscribe email test@example.com to event_id 999
Queried event: {'_id': ObjectId('[illegible]'), 'title': 'Sample Event', 'location': 'Sample Location', 'date': '2024-08-01', 'description': 'This is a sample event.', 'Subscribers': [], 'event_id': '999'}
Subscribed to event successfully.
.
----------------------------------------------------------------------
Ran 2 tests in 2.238s

OK
```

Figure 5. Unit-Testing

C- Code Review

In the first round, we engaged to perform pair programming and it helped us to uncover some over looked issues. One of the issues was clarity of the codes. It brings a serious maintainability issue it may cause that the developers get confused in understanding the code and making necessary updates. After discussion, we agreed on a solution. Some clear comments should be added to the codes to make it more understandable.

After pair programming, we decided to request one of the team members to review our code and give comments. Since all of the members were busy due to the final exam, we performed the code review by ourselves. Mobina reviewed the code developed by Saeideh. The refined code was submitted and was approved by member responsible for repository. Consequently, the collaboration through pair programming and second round review enhanced the performance and quality of the project.



sdkhtri / coen-6311-summer-2024-demo

coen-6311-summer-2024-demo Public

code-review · 6 Branches · 0 Tags

This branch is 3 commits behind main.

Mobi1997 code review: all is good and no changes needed ✓ · 30 Commits

.github/workflows	Create blank.yml	last month
Assignment2	Update index.html	last week
coen 6311-summer-2024-demo	code review: all is good and no changes needed	1 hour ago
Assignment 1 COEN 6311 Summer 2024.pdf	Add files via upload	last month
README.md	Update README.md	last month

README

Hello

About

No description, website, or topics provided.

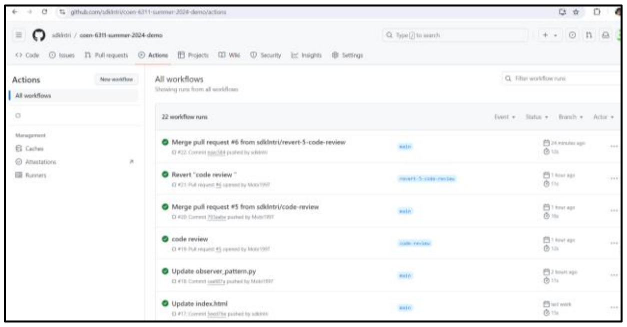
Readme · Activity · 0 stars · 1 watching · 0 forks

Releases: No releases published

Packages: No packages published

Contributors 2

6- (a)



22 workflow runs

Merge pull request #6 from sdklntri/revert-5-code-review

Revert "code review"

Merge pull request #5 from sdklntri/code-review

code review

Update observer_pattern.py

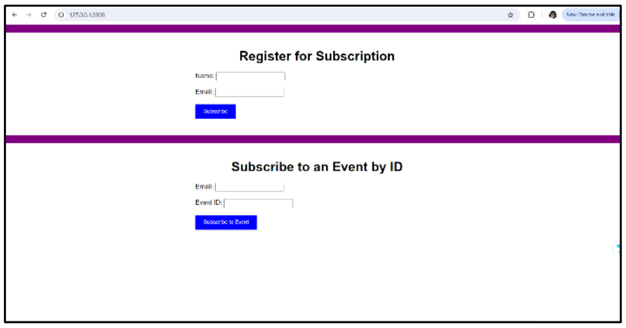
Update index.html

(b)

Figure 6. Code Review Screenshots

VI. RESULTS

The output result of Event-Subscription-Notification module of the DepartmentDirect platform is shown in Figure 6.



Register for Subscription

Name:

Email:

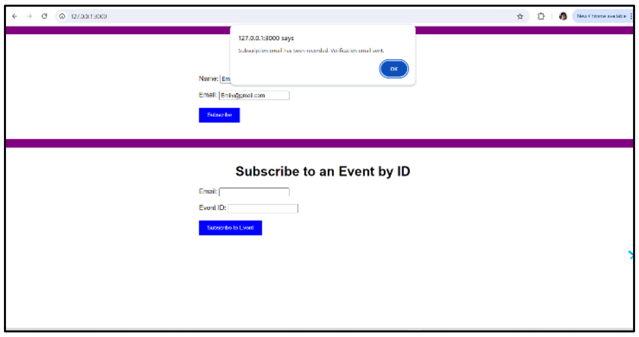
Subscribe to an Event by ID

Email:

Event ID:

Figure 7. User Interface

When an applicant sends a subscription request, the system renders a confirmation message and stores the new subscriber's data in the subscribers' data-base. Figure 7 shows the confirmation message.



127.0.0.1:3000 says

Subscription email has been recorded. Verification email sent.

Name: Em

Email: Emily@gmail.com

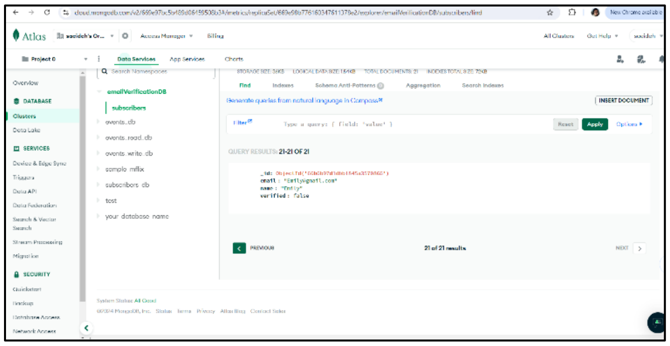
Subscribe to an Event by ID

Email:

Event ID:

Figure 8. Confirmation Message after Subscription Request

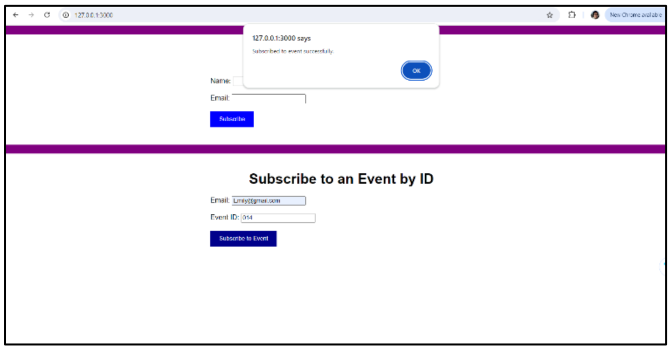
Figure 8 illustrates that Emily's data has been stored in the subscribers' data-base when Emily sends a subscription request.



```
QUERY RESULTS: 21-21 OF 21

_id: ObjectId('66b6b97d1d0b1845a2578665')
email : "Emily@gmail.com"
name : "Emily"
verified : false
```

Figure 9. Subscriber data recorded in the Data-base

When, Emily wants to subscribe to an event she enters her email and the desired event id in the form and receives a confirmation message which is shown in Figure 9.



127.0.0.1:3000 says

Subscribed to event successfully.

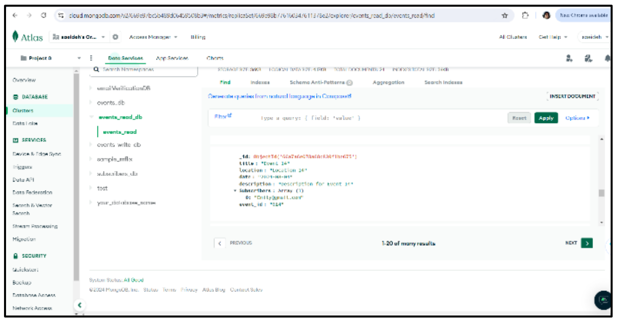
Subscribe to an Event by ID

Email: Emily@gmail.com

Event ID: 014

Figure 10. Subscription to an Event using Event-ID

As can be seen in Figure 9, Emily has requested to subscribe to Event 014. Figure 10 shows that Emily has been added to the subscribers list of event 014.



```
_id: ObjectId('66a7a6e078a60c638f1be675')
title : "Event 14"
location : "Location 14"
date : "2024-08-01"
description : "Description for Event 14"
subscribers : Array (1)
  0: "Emily@gmail.com"
event_id : "014"
```

Figure 11. Updated Event Data-Base

VII. CONCLUSION

In conclusion, the developed project serves a necessary part of the DepartmentDirect platform that is required for subscription to events and updates by the prospected students to enable them get informed about the news and upcoming events at Concordia University.

VIII. REFERENCES

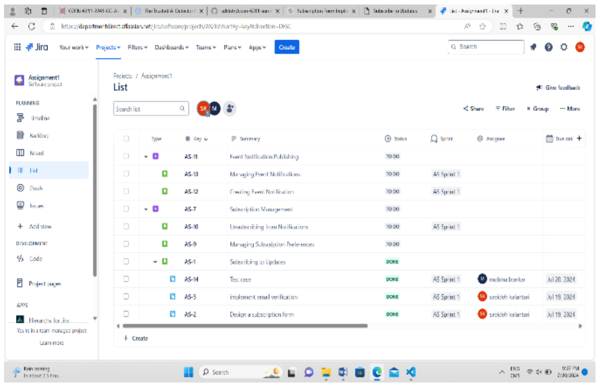
[1] https://github.com/.
[2] https://www.atlassian.com/software/jira.
[3] https://draw.io/.

[4] https://cloud.mongodb.com/.

IX. APENDIX

1- Screen shots of Jira:

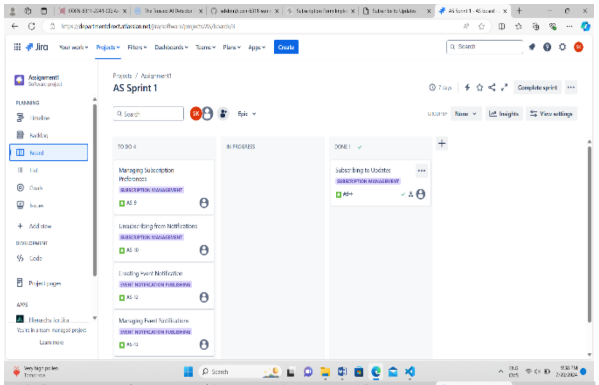
A. Assignment 1:



Projects / Assignment1

List

Type	Key	Summary	Status	Sprint	Assignee	Due date
	AS-11	Event Notification Publishing	TO DO			
	AS-13	Managing Event Notifications	TO DO	AS Sprint 1		
	AS-12	Creating Event Notification	TO DO	AS Sprint 1		
	AS-7	Subscription Management	TO DO			
	AS-10	Unsubscribing from Notifications	TO DO	AS Sprint 1		
	AS-9	Managing Subscription Preferences	TO DO			
	AS-1	Subscribing to Updates	DONE			
	AS-14	Test case	DONE	AS Sprint 1	mobina bonkar	Jul 20, 2024
	AS-3	implement email verification	DONE	AS Sprint 1	saeideh kalantari	Jul 19, 2024
	AS-2	Design a subscription form	DONE	AS Sprint 1	saeideh kalantari	Jul 19, 2024



Projects / Assignment1

AS Sprint 1

TO DO 4

Managing Subscription Preferences
SUBSCRIPTION MANAGEMENT
AS-9

Unsubscribing from Notifications
SUBSCRIPTION MANAGEMENT
AS-10

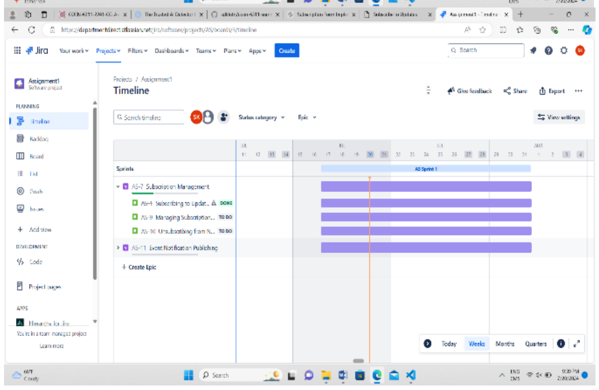
Creating Event Notification
EVENT NOTIFICATION PUBLISHING
AS-12

Managing Event Notifications
EVENT NOTIFICATION PUBLISHING
AS-13

IN PROGRESS

DONE 1

Subscribing to Updates
SUBSCRIPTION MANAGEMENT
AS-1

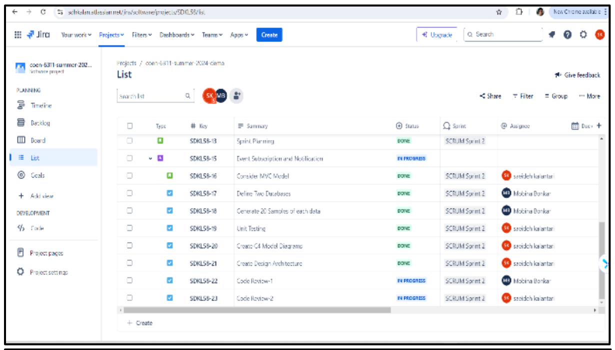


Projects / Assignment1

Timeline

AS-7 Subscription Management
AS-1 Subscribing to Updat... DONE
AS-9 Managing Subscription... TO DO
AS-10 Unsubscribing from N... TO DO
AS-11 Event Notification Publishing

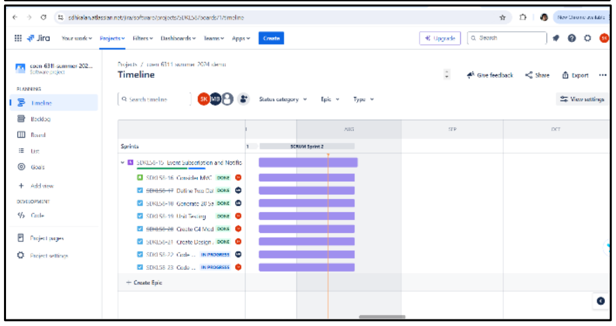
B. Assignment 2:



Projects / coen-6311-summer-2024-demo

List

Type	Key	Summary	Status	Sprint	Assignee
	SDKL58-13	Sprint Planning	DONE	SCRUM Sprint 2	
	SDKL58-15	Event Subscription and Notification	IN PROGRESS		
	SDKL58-16	Consider MVC Model	DONE	SCRUM Sprint 2	saeideh kalantari
	SDKL58-17	Define Two Databases	DONE	SCRUM Sprint 2	Mobina Bonkar
	SDKL58-18	Generate 20 Samples of each data	DONE	SCRUM Sprint 2	Mobina Bonkar
	SDKL58-19	Unit Testing	DONE	SCRUM Sprint 2	saeideh kalantari
	SDKL58-20	Create C4 Model Diagrams	DONE	SCRUM Sprint 2	saeideh kalantari
	SDKL58-21	Create Design Architecture	DONE	SCRUM Sprint 2	saeideh kalantari
	SDKL58-22	Code Review-1	IN PROGRESS	SCRUM Sprint 2	Mobina Bonkar
	SDKL58-23	Code Review-2	IN PROGRESS	SCRUM Sprint 2	saeideh kalantari



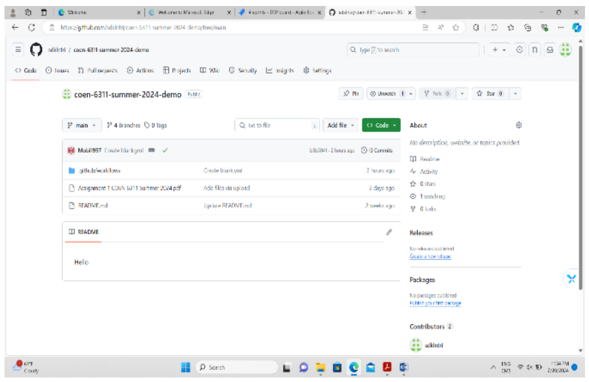
Projects / coen-6311-summer-2024-demo

Timeline

SDKL58-15 Event Subscription and Notific...

2- Screen shots of GitHub:

A. Assignment 1:



sdklntri / coen-6311-summer-2024-demo

coen-6311-summer-2024-demo Public

main · 4 Branches · 0 Tags

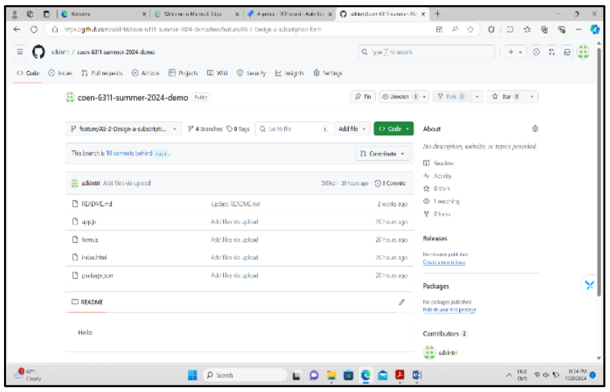
Mobi1997 Create blank.yml · b3b5041 · 2 hours ago · 13 Commits

.github/workflows	Create blank.yml	2 hours ago
Assignment 1 COEN 6311 Summer 2024.pdf	Add files via upload	2 days ago
README.md	Update README.md	2 weeks ago

README

Hello

About: No description, website, or topics provided.



sdklntri / coen-6311-summer-2024-demo

coen-6311-summer-2024-demo Public

feature/AS-2-Design-a-subscripti... · 4 Branches · 0 Tags

This branch is 10 commits behind main.

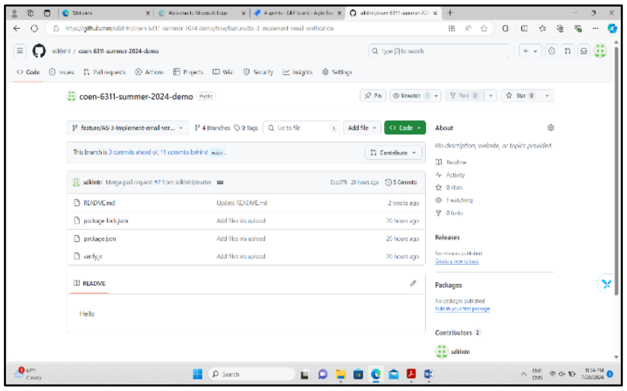
sdklntri Add files via upload · 20 hours ago · 3 Commits

README.md	Update README.md	2 weeks ago
app.js	Add files via upload	20 hours ago
form.js	Add files via upload	20 hours ago
index.html	Add files via upload	20 hours ago
package.json	Add files via upload	20 hours ago

README

Hello

About: No description, website, or topics provided.


sdklntri / coen-6311-summer-2024-demo

coen-6311-summer-2024-demo Public

feature/AS-3-implement-email-ver... 4 Branches 0 Tags

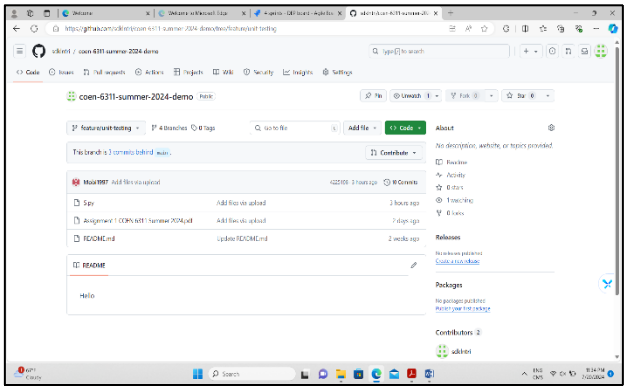
This branch is 3 commits ahead of, 11 commits behind main.

sdklntri Merge pull request #7 from sdklntri/feature 5 Commits

README.md	Update README.md	2 weeks ago
package-lock.json	Add files via upload	20 hours ago
package.json	Add files via upload	20 hours ago
verify.js	Add files via upload	20 hours ago

README

Hello


sdklntri / coen-6311-summer-2024-demo

coen-6311-summer-2024-demo Public

feature/unit-testing 4 Branches 0 Tags

This branch is 3 commits behind main.

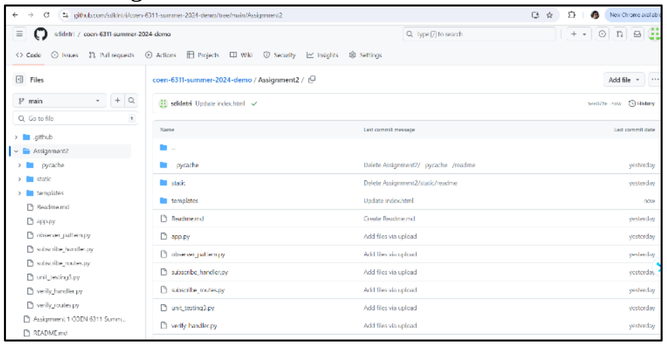
Mobi1997 Add files via upload 10 Commits

5.py	Add files via upload	3 hours ago
Assignment 1 COEN 6311 Summer 2024.pdf	Add files via upload	2 days ago
README.md	Update README.md	2 weeks ago

README

Hello

B. Assignment 2:


github.com/sdklntri/coen-6311-summer-2024-demo/tree/main/Assignment2

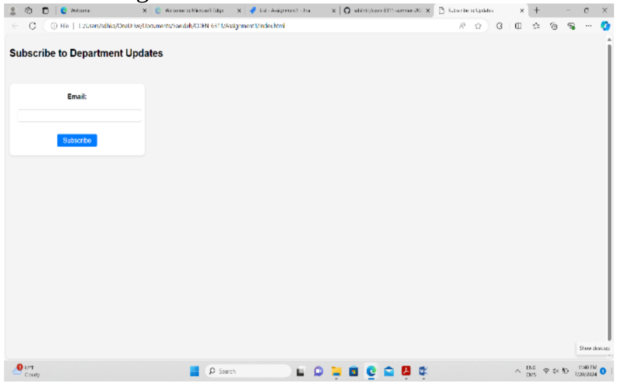
coen-6311-summer-2024-demo / Assignment2 /

sdklntri Update index.html

Name	Last commit message	Last commit date
..		
pycache	Delete Assignment2/ pycache /readme	yesterday
static	Delete Assignment2/static/readme	yesterday
templates	Update index.html	now
Readme.md	Create Readme.md	yesterday
app.py	Add files via upload	yesterday
observer_pattern.py	Add files via upload	yesterday
subscribe_handler.py	Add files via upload	yesterday
subscribe_routes.py	Add files via upload	yesterday
unit_testing3.py	Add files via upload	yesterday
verify handler.py	Add files via upload	yesterday

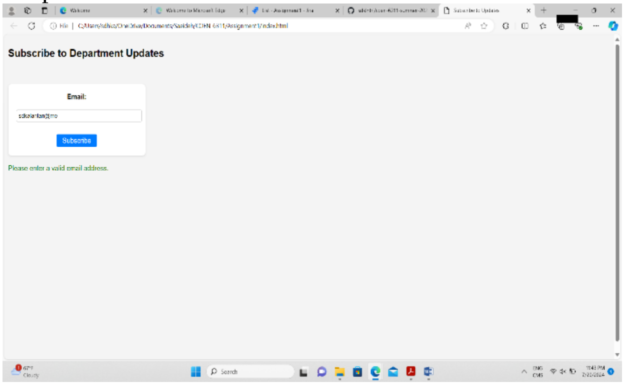
3- Screen shots of the outcome:

C. Assignment 1:


Subscribe to Department Updates

Email:

This screen shot shows the subscription form designed to allow the future applicants send their email address to the publisher.

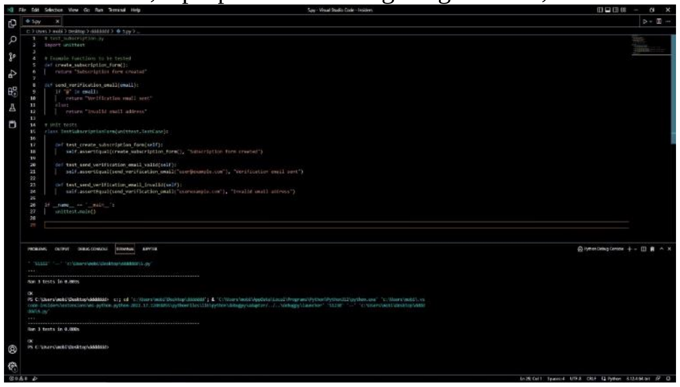

C:/Users/sdkia/OneDrive/Documents/Saeideh/COEN 6311/Assignment1/index.html

Subscribe to Department Updates

Email: sdkalantari@mo

Please enter a valid email address.

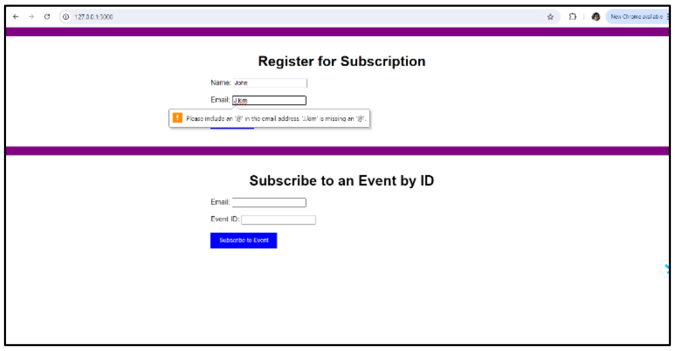
This screen shot shows that upon entering an invalid email address, a proper error message is generated,


```
# test_subscription.py
import unittest

# Example functions to be tested
def create_subscription_form():
    return "Subscription form created"

def send_verification_email(email):
    if "@" in email:
        return "Verification email sent"
    else:
        return "Invalid email address"

# Unit tests
class TestSubscriptionForm(unittest.TestCase):

    def test_create_subscription_form(self):
        self.assertEqual(create_subscription_form(), "Subscription form created")

    def test_send_verification_email_valid(self):
        self.assertEqual(send_verification_email("user@example.com"), "Verification email sent")

    def test_send_verification_email_invalid(self):
        self.assertEqual(send_verification_email("userexample.com"), "Invalid email address")

if __name__ == '__main__':
    unittest.main()
```

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
PS C:\Users\mobi\Desktop\ddddddd> c:; cd 'c:\Users\mobi\Desktop\ddddddd'; & 'C:\Users\mobi\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\mobi\.vscode-insiders\extensions\ms-python.python-2022.17.12800955\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '51238' '--' 'C:\Users\mobi\Desktop\ddddddd\5.py'
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
PS C:\Users\mobi\Desktop\ddddddd>
```

Assignment 2:

Invalid Email Address:


127.0.0.1:3000

Register for Subscription

Name: John

Email: J.kim

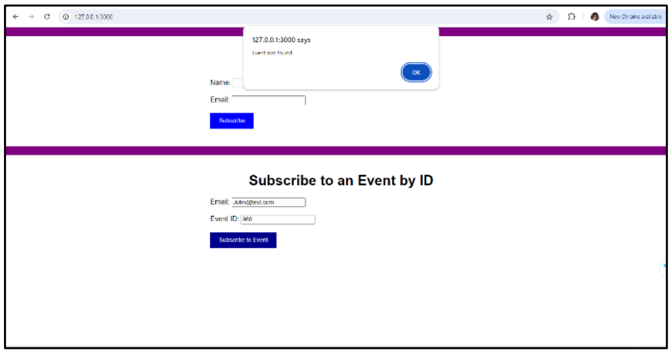
Please include an '@' in the email address. 'J.kim' is missing an '@'.

Subscribe to an Event by ID

Email:

Event ID:

Invalid Event-ID:


127.0.0.1:3000

127.0.0.1:3000 says

Event not found.

Name:

Email:

Subscribe to an Event by ID

Email: John@test.com

Event ID: 980