Florida International University
Knight Foundation School of Computing and Information Sciences

Software Engineering Focus

Final Deliverable

Enhancing Art Appreciation and Museum Operations: ArtLens for exploration and analysis

Team Members: Niccholas Tim Reiz, Mauricio Piedra, Lorena A. Quincoso-Lugones, Dory Apollon, Ariel Ramos-Perez

Product Owner(s): Niccholas Tim Reiz

Instructor: Masoud Sadjadi

Abstract

This document presents the information necessary to gain a good understanding of the "Enhancing Appreciation and Museum Operations: ArtLens for Exploration and Analysis" project. ArtLens is an innovative application designed to enhance art appreciation and streamline museum operations. Features were developed to ensure personalized guidance, interactive experiences for visitors, and museum analytics for site admins. Specifically, the application is built using a Flask/Django backend with a React/Swift/Streamlit frontend. For the React website, JavaScript/CSS was used to display artwork search, ratings, recommendation, related resources, a gallery tour simulator, and a login function. For artwork ratings and user-admin login credentials, data is stored in a MySQL database. The recommendation system applies a deep learning algorithm using RESNET50/FAISS in Python. The *A** path planning algorithm was used for the gallery tour simulator. An admin dashboard to view database information was built using Streamlit. The educational resource connects to the Google Gemini API to retrieve AI-generated responses. Verification, algorithms, the dashboard, and API requests were made using Python stored in the backend file. The ios-app, coded in Swift, incorporates the artwork search, rating, recommendation system, tour simulation, and a daily featured artwork. These features convey how a modular architecture allows for the easy addition of new features and expansion to other museums or art institutions. ArtLens showcases the power of technology in enhancing cultural heritage appreciation. This project presents opportunities for expanding AI capabilities in artwork curation, like AR/VR technologies for virtual museum experiences.

Table of Contents

INTRODUCTION

ArtLens is a project created with the core idea of using innovative technology to help people, especially those with an interest for art and history, explore cultural treasures such as the works held at The Metropolitan Museum of Art (MET). Our project's foundation lies in the API provided by this museum. The MET API offers a dataset with various details for all of the pieces. ArtLens aims to use this API and combine it with modern technologies to enhance the experience of users discovering art online. We focus on increasing exposure, sparking interest, and fostering deeper engagement to art. To this end, we developed both a website and an iOS app, ensuring ArtLens is accessible through a wide range of devices.

Current System

The MET offers a website with a wide variety of options related to tours, events, galleries, and even research. There, we can see many resources that can generate an interest for exploring this amazing museum, and make it simple for one to visit if they so wish. However, we saw potential for improvement in many of the site's features. When we first started looking into this project, we discussed how this site could be empowered by the use of technologies such as dynamic searches and recommendation systems, and how the methodology for user engagement could be improved. These ideas and knowing about the API offered by the museum is what pushed us to create this project.

Purpose of New System

The purpose of our project is to extend the capabilities of the MET websites by making a system which can be more dynamic, and enticing for users to keep using it and want to learn more about art. One of our main inspirations was Pinterest, and the way it works to keep their users wanting to explore their interests, as well as making new ones. We developed a website capable of providing dynamic searches along with a rating system and an explore page. Additionally, we also built an iOS app that is also capable of performing these tasks; and created a dashboard capable of providing insight about usage and popularity of the system and the artworks.

Incorporated in the backend of these apps, are components which are crucial for achieving the goal of this project. ArtLens uses Computer Vision and Machine Learning to recommend similar artworks when users are checking out artworks and also recommends educational material with more information about the artwork also being generated.

User Stories

The following section provides the detailed user stories that were implemented in this iteration of the ArtLens project. These user stories served as the basis for the implementation of the project's features. This section also shows the user stories that are to be considered for future development.

Implemented User Stories

- As a developer, I want to fully understand Computer Vision Tasks for potential customer recommendations and advanced research capabilities.

- As a developer, I want to research architectures that we might implement in order to connect the frontend and backend for the final product.

- As a developer, I want to explore and familiarize myself with educational media to understand and develop recommendations based on user interactions.

- As a developer, I want toResearch possible frontend designs for the MET API implementation so I can implement a user-friendly design for our website and app.

- Familiarize myself with the iOS app development process for the frontend of our application.

- As a developer, I want to develop an enhanced collection search that uses machine learning to personalize artwork discovery, ensuring it aligns with our goal of improving visitor engagement through tailored recommendations.

- As a developer, I want to build a dashboard that tracks visitor engagement data for different artworks (e.g., time spent, interactions), so that museum curators can make informed decisions about exhibit popularity.

- As a developer, I want to create a recommendation algorithm that suggests personalized educational content based on a user's interactions with specific artworks, so that we can enhance the learning experience for students and families.

- As a developer, I want to build a responsive and interactive museum website using React, so that users can seamlessly browse collections, view educational content, and receive recommendations in a modern and efficient user interface.

- As a developer, I want to build a feature that allows mobile app users to like or favorite specific artworks during their tours, so that users can revisit their selections and receive personalized recommendations.

Pending User Stories

- As a developer, I want to build a picture recognition system so that users can upload an image or take a picture of an artwork and find similar pieces in the collection.

- As a developer, I want to create an AR feature to identify artworks in real-time using the phone camera, so that users can learn more about the pieces during their museum visit.

- As a developer, I want an engaging storytelling feature for children that connects artworks with interesting narratives, so they can enjoy an educational museum visit.

- As a developer, I want to build advanced filters (e.g., by artist, time period, color) powered by the MET API, so users can efficiently find specific types of artworks.

- As a developer, I want to build predictive analytics on visitor trends and preferences, so users can plan future exhibitions more effectively.

PROJECT PLAN

This section describes the planning that went into the realization of this project. This project incorporated the agile development techniques and as such required the sprints to be planned. These sprint plannings are detailed in the section. This section also describes the components, both software and hardware, chosen for this project.

Hardware and Software Resources

The following is a list of all hardware and software resources that were used in this project:

Hardware:
- macOS device
- Windows device

Software:
- React
- Postman
- JavaScript
- HTML
- Swift
- Python
- CSS
- Streamlit
- Google Gemini
- Resnet50
- MySql
- Flask
- PyTorch
- Django
- Pandas
- Pillow
- Numpy
- Scikit-learn
- Joblib
- Faiss
- Matplotlib

Sprints Plan

Sprint 1

After discussion, the velocity of the team was estimated to be 30 (3 hours, 10 days per student) * 5 (total # of students) = 150 hours.

- **Lorena A. Quincoso Lugones** (Scrum Master)

- User Story 4: As a Developer, I want to ensure I fully understand Computer Vision Tasks for potential customer recommendations. I'll follow along with courses/playlists such as the following: https://youtube.com/playlist?list=PLTgRMOcmRb3NuchXdEjbg5yiD3Bx6LpFg&si=XRDqUbaRD3sP4OuQ.

- **Ariel Ramos Perez** (Developer Team)
- User Story 3: As a developer, I would like to explore and learn Python, and more specifically, Pytorch libraries, so that I can have a good understanding of which ones could prove useful in our project.

- **Niccholas Reiz**: (Project Owner)
- User Story 1: As a Product Owner, I would like to research public MET API implementation/data, so that I will have ideas for features to present to the team within this particular week.

- **Dory Apollon**(Developer Team)
    - User Story 5: As a developer, I want to familiarize myself with the tools we will be using throughout the project so that I can work as effectively as possible. IDE's, API's, packages, etc...

- **Mauricio Piedra**(Developer Team)
    - User Story 2: As a developer, I would like to research on the MET API implementation so I can fully understand further improvements or features needed given the data distribution.

Sprint 2

After discussion, the velocity of the team was estimated to be 30 (3 hours, 10 days per student) * 5 (total # of students) = 150 hours.

- **Lorena A. Quincoso Lugones**(Developer Team)
    - User Story 4: As a Developer, I want to ensure I fully understand Computer Vision Tasks for potential customer recommendations and advanced research capabilities.
- **Ariel Ramos Perez**(Project Owner)
    - User Story 1: As a Product Owner, I would like to research architectures that we might implement in order to connect the frontend and backend for the final product.
- **Niccholas Reiz**: (Developer Team)
    - User Story 5: As a developer, I want to explore and familiarize myself with educational media to understand and develop recommendations based on user interactions.
- **Dory Apollon**(Scrum Master)
    - User Story 2: As a developer, I would like to research possible frontend designs for the MET API implementation so I can implement a user friendly design for our website and app.
- **Mauricio Piedra**(Developer Team)
    - User Story 3: As a developer, I would like to understand how to familiarize myself with the iOS app development process for the frontend of our application.

Sprint 3

After discussion, the velocity of the team was estimated to be 30 (3 hours, 10 days per student) * 5 (total # of students) = 150 hours.

- **Lorena A. Quincoso Lugones:**(Project Owner)
    - User Story 1: As a Product Owner, I want to prioritize developing an enhanced collection search that uses machine learning to personalize artwork discovery, ensuring it aligns with our goal of improving visitor engagement through tailored recommendations.

- **Ariel Ramos Perez**(Scrum Master)
    - User Story 2: As a developer, I want to build a dashboard that tracks visitor engagement data for different artworks (e.g., time spent, interactions), so that museum curators can make informed decisions about exhibit popularity.

- **Niccholas Reiz**: (Developer Team)
    - User Story 4: As a developer, I want to create a recommendation algorithm that suggests personalized educational content based on a user's interactions with specific artworks, so that we can enhance the learning experience for students and families.

- **Dory Apollon:** (Developer Team)
    - User Story 5: As a developer, I want to build a responsive and interactive museum website using React, so that users can seamlessly browse collections, view educational content, and receive recommendations in a modern and efficient user interface.

- **Mauricio Piedra** (Developer Team)
    - User Story 3: As a developer, I want to build a feature that allows mobile app users to like or favorite specific artworks during their tours, so that users can revisit their selections and receive personalized recommendations.

Sprint 4

After discussion, the velocity of the team was estimated to be 30 (3 hours, 10 days per student) * 5 (total # of students) = 150 hours.

- **Lorena A. Quincoso Lugones:**(Developer Team)
    - User Story 1: As a Developer, I want to continue developing an enhanced collection search that uses machine learning to personalize artwork discovery, ensuring it aligns with our goal of improving visitor engagement through tailored recommendations

- **Ariel Ramos Perez**(Developer Team)
    - User Story 2: As a developer, I want to continue building a dashboard that tracks visitor engagement data for different artworks (e.g., time spent, interactions), so that museum curators can make informed decisions about exhibit popularity.

- **Niccholas Reiz**:(Developer Team)
    - User Story 4: As a developer, I want to continue creating a recommendation algorithm that suggests personalized educational content based on a user's interactions with specific artworks, so that we can enhance the learning experience for students and families.

- **Dory Apollon:**(Project Owner)
    - User Story 5: As a developer, I want to continue building a responsive and interactive museum website using React, so that users can seamlessly browse collections, view educational content, and receive recommendations in a modern and efficient user interface.

- **Mauricio Piedra**(Scrum Master)
    - User Story 3: As a developer, I want to continue building a feature that allows mobile app users to like or favorite specific artworks during their tours, so that users can revisit their selections and receive personalized recommendations.

Sprint 5

After discussion, the velocity of the team was estimated to be 30 (3 hours, 10 days per student) * 5 (total # of students) = 150 hours.

- **Lorena A. Quincoso Lugones:**(Project Owner)
    - User Story 1: As a Developer, I want to continue developing an enhanced collection search that uses machine learning to personalize artwork discovery, ensuring it aligns with our goal of improving visitor engagement through tailored recommendations

- **Ariel Ramos Perez**(Developer Team)
    - User Story 2: As a developer, I want to continue building a dashboard that tracks visitor engagement data for different artworks (e.g., time spent, interactions), so that museum curators can make informed decisions about exhibit popularity.

- **Niccholas Reiz**: (Developer Team)
    - User Story 4: As a developer, I want to continue creating a recommendation algorithm that suggests personalized educational content based on a user's interactions with specific artworks, so that we can enhance the learning experience for students and families.

- **Dory Apollon:**(Developer Team)
    - User Story 5: As a developer, I want to continue building a responsive and interactive museum website using React, so that users can seamlessly browse collections, view educational content, and receive recommendations in a modern and efficient user interface.

- **Mauricio Piedra**(Scrum Master)
    - User Story 3: As a developer, I want to continue building a feature that allows mobile app users to like or favorite specific artworks during their tours, so that users can revisit their selections and receive personalized recommendations.

Sprint 6

After discussion, the velocity of the team was estimated to be 30 (3 hours, 10 days per student) * 5 (total # of students) = 150 hours.

- **Lorena A. Quincoso Lugones:**(Developer Team)
    - User Story 1: As a Developer, I want to finish developing an enhanced collection search that uses machine learning to personalize artwork discovery, ensuring it aligns with our goal of improving visitor engagement through tailored recommendations.

- **Ariel Ramos Perez**(Developer Team)
    - User Story 2: As a developer, I want to finish building a dashboard that tracks visitor engagement data for different artworks (e.g., time spent, interactions), so that museum curators can make informed decisions about exhibit popularity.

- **Niccholas Reiz**: (Project Owner)
    - User Story 4: As a developer, I want to finish creating a recommendation algorithm that suggests personalized educational content based on a user's interactions with specific artworks, so that we can enhance the learning experience for students and families.

- **Dory Apollon:**(Scrum Master)
    - User Story 5: As a developer, I want to finish building a responsive and interactive museum website using React, so that users can seamlessly browse collections, view educational content, and receive recommendations in a modern and efficient user interface.
    - User Story 6: As a developer, I want to put all components together under the same website/mobile app, ensuring consistency and functionality.

- **Mauricio Piedra**(Developer Team)
    - User Story 3: As a developer, I want to finish building a feature that allows mobile app users to like or favorite specific artworks during their tours, so that users can revisit their selections and receive personalized recommendations.

Sprint 7

After discussion, the velocity of the team was estimated to be 30 (3 hours, 10 days per student) * 5 (total # of students) = 150 hours.

- **Lorena A. Quincoso Lugones:**
  - User Story 1: As a Developer, I want to start working on the corresponding PowerPoint presentation for showcase day and video recording.
  - User Story 6: As a developer, I want to get started with the poster representing my particular feature, and therefore my contributions to the project.

- **Ariel Ramos Perez**
  - User Story 4: As a developer, I want to get started with the Sequence Diagram in order to accurately reflect the timing and order of all system components.
  - User Story 6: As a developer, I want to get started with the poster representing my particular feature, and therefore my contributions to the project.

- **Niccholas Reiz**:
  - User Story 5: As a Product Owner, I'm expecting to record intro and demo videos along with the Developer's team, so we can effectively showcase our contributions.
  - User Story 6: As a developer, I want to get started with the poster representing my particular feature, and therefore my contributions to the project.

- **Dory Apollon:**
  - User Story 3: As a developer, I want to start working on the Use Case diagram, depicting user-system interactions.
  - User Story 6: As a developer, I want to get started with the poster representing my particular feature, and therefore my contributions to the project.

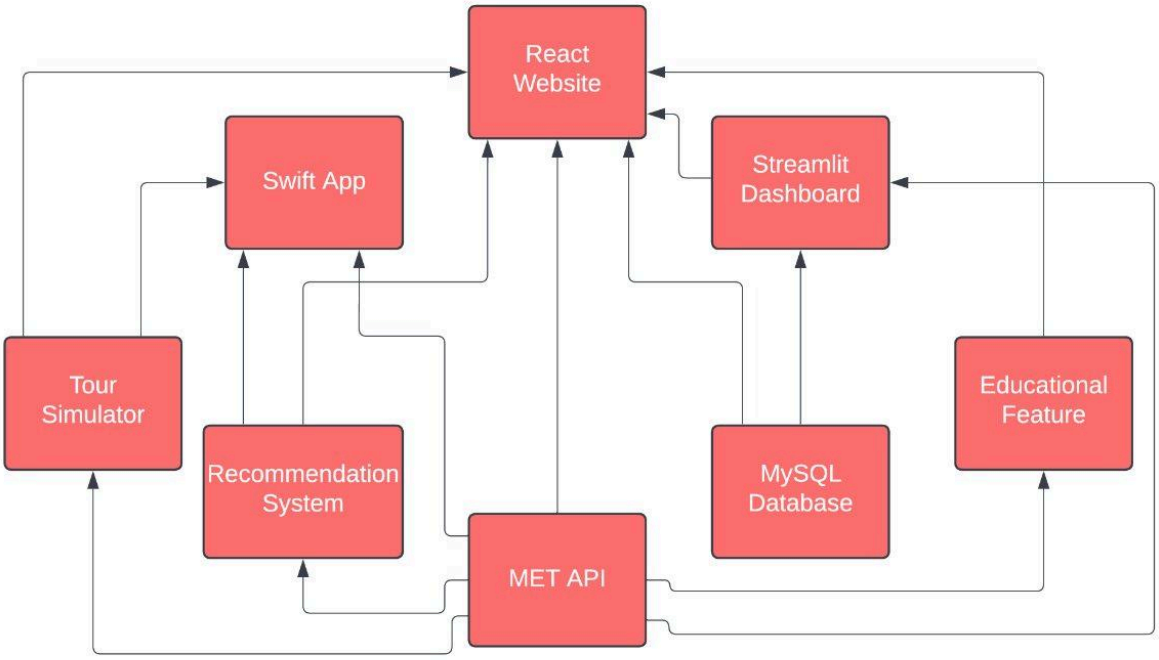
- **Mauricio Piedra**
  - User Story 2: As a developer, I want to start working on the UML diagram, representing all classes and their relationships within the entire project.
  - User Story 6: As a developer, I want to get started with the poster representing my particular feature, and therefore my contributions to the project.

System Design

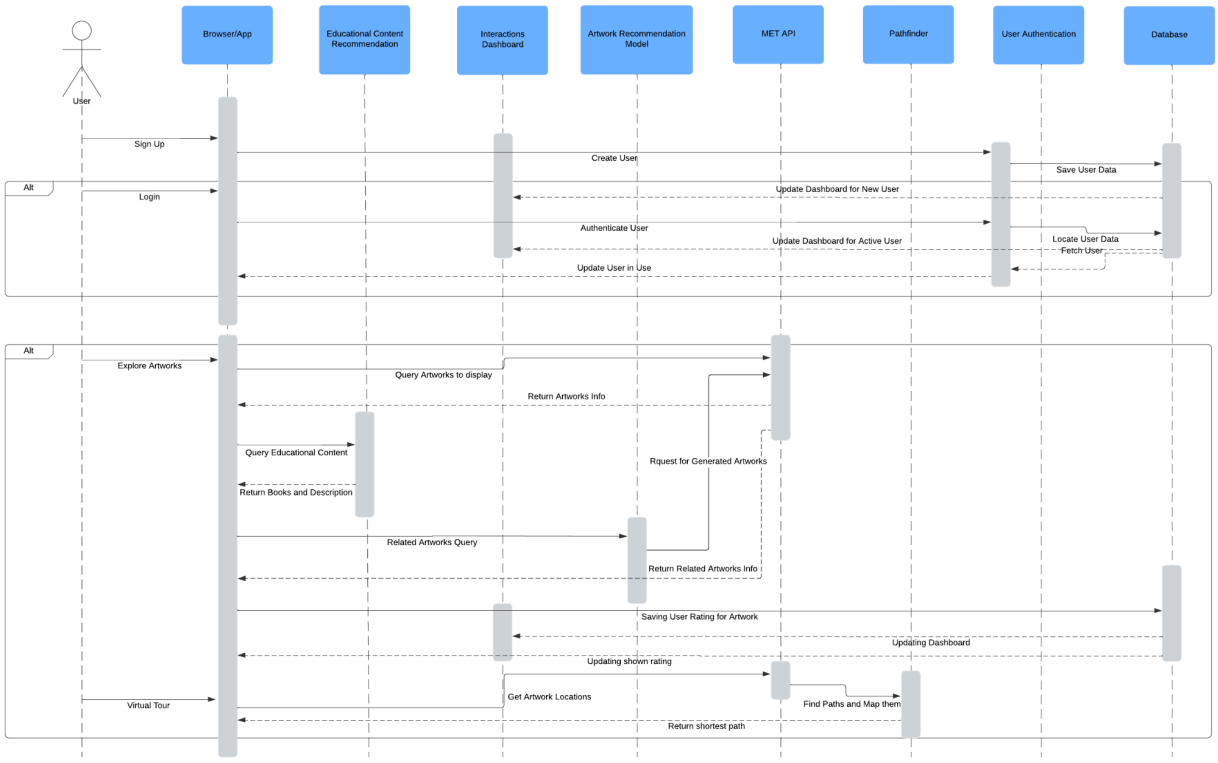
This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

Architectural Patterns

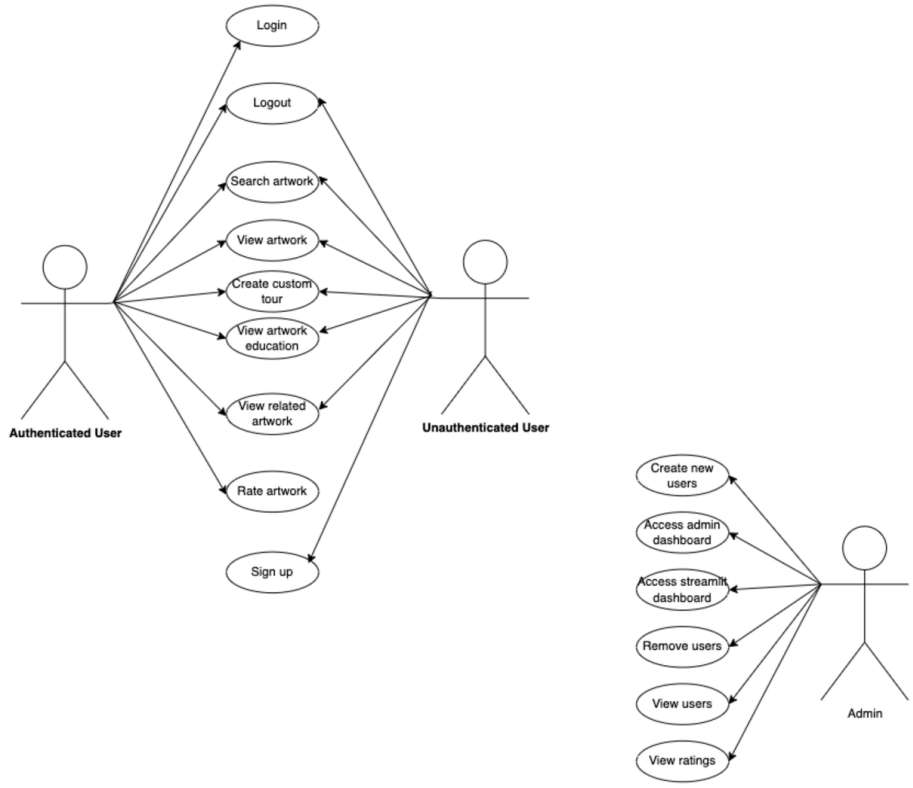
System Decomposition



Sequence Diagram



Use Case Diagram



System Validation

In the scope of the ArtLens project, system validation tests can closely be related to performance tests for a variety of components in the system to ensure that all components operate correctly and steadily. Recommendation System's tests were mainly concerned with the request for photos, i.e. how legitimate image data is fetched when there are issues like bad responses, timeouts, or 404's. All these prevent the situation where picture data is either wrongly fetched or faults occur and picture data is recovered without damage.

The Streamlit Dashboard tests were related to User analytics and user actions on the dashboard. In particular, the tests were to validate whether the data such as total number of users, users both active and inactive, counts of interactions and rank on the leaderboard are sources and presented correctly. The tests uses dummy data that mimics realities of the outside world so as to check for integration of the dashboard with the database as well as normal operating conditions of the dashboard.

For Testing Educational Features, the focus is the Gemini API endpoint. It ensures that appropriate education information is generated continuously, that there are steady appropriate status codes and that satisfactory structured JSON answers are provided. This phase accompanies all others in ensuring effective use of resources for teaching and producing relevant resources for learners.

When it comes to the iOS App, the tests focusing on whether artworks' information can be fetched and presented accurately. They ensure appropriate rendering of JSON responses and that relevant artworks are simply placed.

Recommendation System

```python
@pytest.mark.asyncio
@patch('aiohttp.ClientSession.get')
async def test_fetch_image_success(mock_get):
    # Testing a successful response with an image
    img = Image.new('RGB', (10, 10), color='red')
    img_byte_arr = io.BytesIO()
    img.save(img_byte_arr, format='PNG')
    img_data = img_byte_arr.getvalue()

    mock_response = MagicMock()
    mock_response.__aenter__.return_value.status = 200
    mock_response.__aenter__.return_value.read = AsyncMock(return_value=img_data)
    mock_get.return_value = mock_response

    async with ClientSession() as session:
        url = 'https://fakeurl.com/fakeimage.png'
        result = await fetch_image(session, url)
        assert result is not None
        assert isinstance(result, Image.Image)
        assert result.mode == 'RGB'
```

```python
@pytest.mark.asyncio
@patch('aiohttp.ClientSession.get')
async def test_fetch_image_timeout(mock_get):
    # Testing a timeout during image fetching
    mock_get.side_effect = Exception('Timeout occurred')

    async with ClientSession() as session:
        url = 'https://fakeurl.com/timeout.png'
        result = await fetch_image(session, url)
        assert result is None
```

```python
@pytest.mark.asyncio
@patch('aiohttp.ClientSession.get')
async def test_fetch_image_invalid_data(mock_get):
    # Testing a successful response but with invalid image data
    mock_response = AsyncMock()
    mock_response.status = 200
    mock_response.read.return_value = b'not_an_image'
    mock_get.return_value = mock_response

    async with ClientSession() as session:
        url = 'https://fakeurl.com/invalidimage.png'
        result = await fetch_image(session, url)
        assert result is None
```

```python
@pytest.mark.asyncio
@patch('aiohttp.ClientSession.get')
async def test_fetch_image_not_found(mock_get):
    # Testing a 404 Not Found response
    mock_response = AsyncMock()
    mock_response.status = 404
    mock_response.read.return_value = None
    mock_get.return_value = mock_response

    async with ClientSession() as session:
        url = 'https://fakeurl.com/notfound.png'
        result = await fetch_image(session, url)
        assert result is None
```

```
app\tests\test_recommender.py ....                                                    [100%]

============================== warnings summary ===============================

============================== warnings summary ===============================

============================== warnings summary ===============================
C:\Users\oryou\anaconda3\envs\deep_learning\lib\site-packages\faiss\loader.py:49
  C:\Users\oryou\anaconda3\envs\deep_learning\lib\site-packages\faiss\loader.py:49: DeprecationWarning: numpy.core._multiarray_um
th is deprecated and has been renamed to numpy._core._multiarray_umath. The numpy._core namespace contains private NumPy internal
 and its use is discouraged, as NumPy internals can change without warning in any release. In practice, most real-world usage of
umpy.core is to access functionality in the public NumPy API. If that is the case, use the public NumPy API. If not, you are usin
 NumPy internals. If you would still like to access an internal attribute, use numpy._core._multiarray_umath.__cpu_features__.
    from numpy.core._multiarray_umath import __cpu_features__

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
================== 4 passed, 1 warning in 765.03s (0:12:45) ===================
```

Streamlit Dashboard

```python
def test_get_total_users():
    with patch('dashboard.pymysql.connect') as mock_connect:
        mock_conn = MagicMock()
        mock_cursor = MagicMock()

        mock_cursor.fetchall.return_value = [(100, 80, 20)]

        mock_cursor.description = (
            ('total_users',), ('active_users',), ('inactive_users',)
        )

        mock_conn.cursor.return_value = mock_cursor
        mock_connect.return_value = mock_conn

        dashboard = InteractionsDashboard()


        result = dashboard.get_total_users()


        pd.testing.assert_frame_equal(result, mock_total_users)

        dashboard.close()
        mock_cursor.close.assert_called_once()
        mock_conn.close.assert_called_once()
```

```python
mock_total_users = pd.DataFrame({
    'total_users': [100],
    'active_users': [80],
    'inactive_users': [20]
})

mock_most_active_hour = pd.DataFrame({
    'hour_numeric': [14],
    'interactions': [150]
})

mock_ratings_leaderboard = pd.DataFrame({
    'artwork_id': [1, 2, 3],
    'average_rating': [4.5, 4.0, 3.5],
    'total_ratings': [100, 150, 200]
})
```

```
database_test.py ........                                    [ 61%]
graph_test.py ..                                             [ 76%]
tests\streamlit_test.py .                                    [ 84%]
utility_test.py ..                                           [100%]

================================== 13 passed in 3.81s ==========================
```

Educational Feature

```python
class TestGeminiAPI(unittest.TestCase):

    def setUp(self):
        self.app = app.test_client()
        self.app.testing = True

    def test_generate_response_success(self):
        response = self.app.post(
            "/api/gemini",
            data=json.dumps({"prompt": "Tell me a story about a robot."}),
            content_type="application/json",
        )
        self.assertEqual(response.status_code, 200)
        response_data = json.loads(response.data)
        self.assertIn("response", response_data)
        self.assertIsInstance(response_data["response"], str)
```

```
Niccs-Macbook-Pro:tests niccholasreiz$ pytest test_app.py
============================================== test session starts ==============================================
platform darwin -- Python 3.12.0, pytest-8.3.4, pluggy-1.5.0
rootdir: /Users/niccholasreiz/Documents/GitHub/ArtLens-MET-Insights2/backend/education/tests
plugins: cov-6.0.0, anyio-4.6.2.post1
collected 2 items

test_app.py ..                                                                                          [100%]

============================================== 2 passed in 5.19s ==============================================
```

Tour Simulator

```python
def test_get_galleries(self):
    # Testing the '/api/galleries' endpoint to get all galleries
    response = self.client.get('/api/galleries')
    self.assertEqual(response.status_code, 200)

    data = json.loads(response.data)
    self.assertIsInstance(data, dict)
    self.assertIn('100', data)
```

```python
def test_get_shortest_path_valid(self):
    # Testing the '/api/shortest-path' endpoint with valid gallery IDs
    response = self.client.get('/api/shortest-path', query_string={'start': '100', 'end': '101'})
    self.assertEqual(response.status_code, 200)

    data = json.loads(response.data)
    self.assertIn('path', data)
    self.assertIsInstance(data['path'], list)
    self.assertGreater(len(data['path']), 0)
```

```python
def test_get_shortest_path_invalid(self):
    # Testing the '/api/shortest-path' endpoint with invalid gallery IDs
    response = self.client.get('/api/shortest-path', query_string={'start': '999', 'end': '888'})
    self.assertEqual(response.status_code, 400)

    data = json.loads(response.data)
    self.assertIn('error', data)
    self.assertEqual(data['error'], "Invalid start or end point")
```

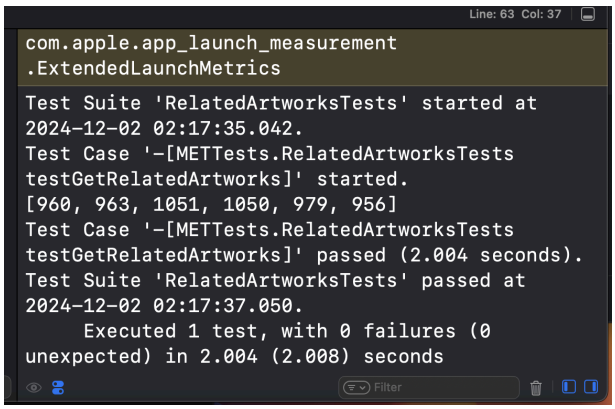
```
============================================= test session starts =============================================
platform win32 -- Python 3.12.4, pytest-8.3.3, pluggy-1.5.0
rootdir: D:\SCHOOL\Semester 12 - Fall 2024\CIS - Caps 2\Project\MET-Museum-ML\backend\tour
plugins: anyio-4.2.0, asyncio-0.24.0
asyncio: mode=Mode.STRICT, default_loop_scope=None
collected 3 items

tests\test_app.py ...                                                                                [100%]

============================================== 3 passed in 0.14s ==============================================
```

iOS App

```swift
final class FetchArtworkTests: XCTestCase {

    var fetchArtwork: FetchArtwork!

    override func setUp() {
        super.setUp()
        fetchArtwork = FetchArtwork()
    }

    func testFetchArtworkSuccess() {
        let expectation = self.expectation(description: "FetchArtwork completes")

        FetchArtwork.getImageData(imageID: 1234) { result in
            switch result {
            case .success(let artworks):
                XCTAssertNotNil(artworks)
                XCTAssertGreaterThan(artworks.count, 0)
            case .failure(let error):
                XCTFail("Fetching failed with error: \(error)")
            }
            expectation.fulfill()
        }

        waitForExpectations(timeout: 5, handler: nil)
    }
}
```

```
Line: 66 Col: 48
Test Suite 'FetchArtworkTests' started at
2024-12-02 01:36:58.096.
Test Case '-[METTests.FetchArtworkTests
testFetchArtworkSuccess]' started.
["title": Optional("Candlestick"), "imageUrl":
Optional("https://images.metmuseum
.org/CRDImages/ad/original/136084.jpg")]
Test Case '-[METTests.FetchArtworkTests
testFetchArtworkSuccess]' passed (0.500 seconds).
Test Suite 'FetchArtworkTests' passed at
2024-12-02 01:36:58.597.
     Executed 1 test, with 0 failures (0
unexpected) in 0.500 (0.501) seconds
Filter
```

```swift
final class ImageDataTests: XCTestCase {

     func testImageDataDecoding() throws {
         let jsonData = """
         {
             "primaryImage": "http://example.com/sunset.jpg",
             "title": "Sunset"
         }
         """.data(using: .utf8)!

         do {
             let imageData = try JSONDecoder().decode(ImageData.self, from: jsonData)
             XCTAssertEqual(imageData.primaryImage, "http://example.com/sunset.jpg")
             XCTAssertEqual(imageData.title, "Sunset")
         } catch {
             XCTFail("Decoding failed with error: \(error)")
         }
     }
}
```



GLOSSARY

(List of tems, words to clarify)

APPENDIX

Appendix A - UML Diagrams

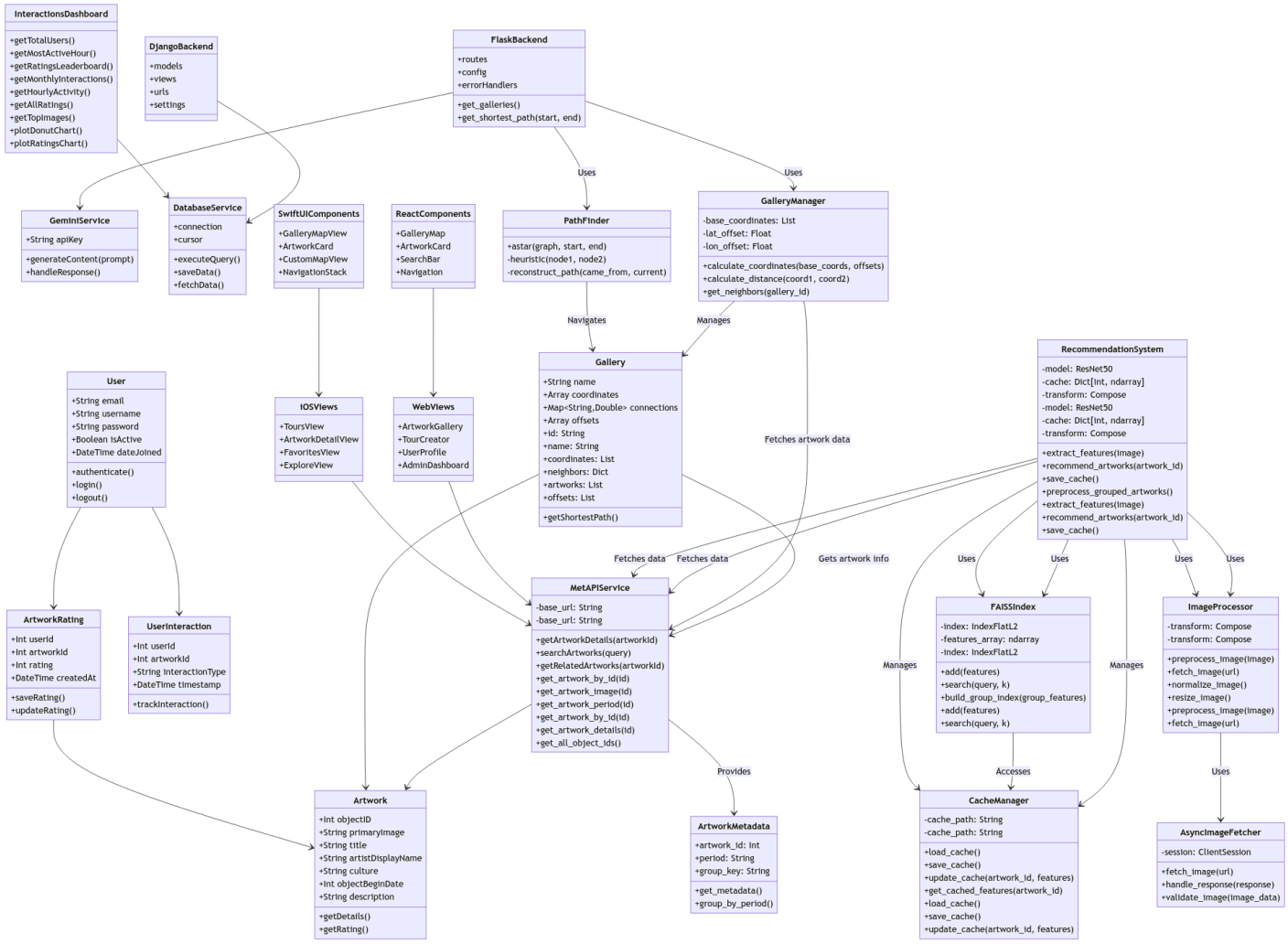


Fig 1. UML Diagram for all project classes/components and its interactions

Appendix B - User Interface Design

React

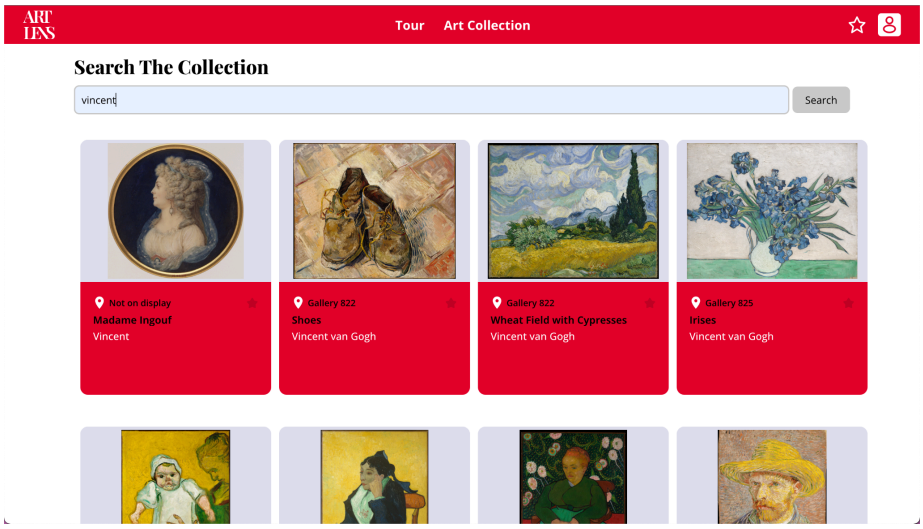


Fig 1. Search Page

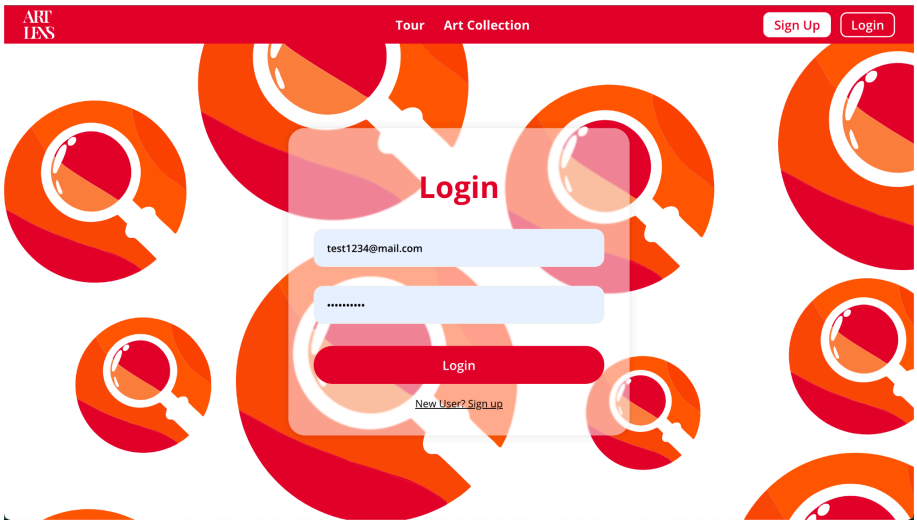


Fig 2. Login Page

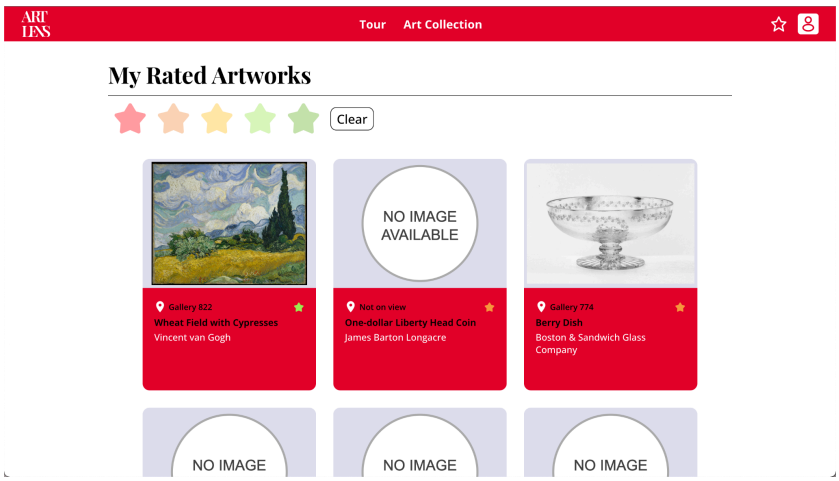
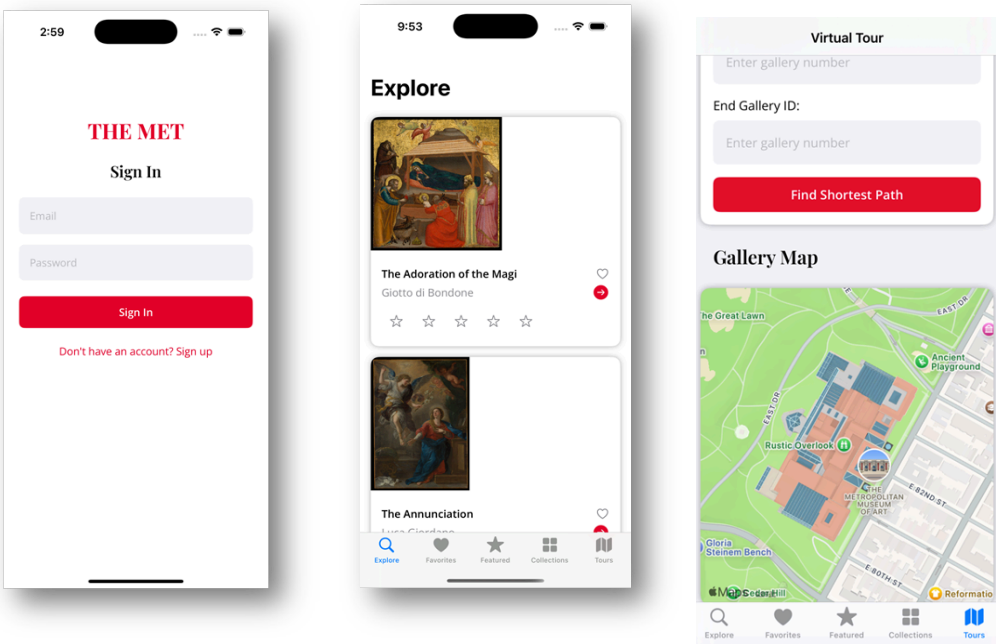


Fig 3. Related Artworks Page

iOS



Appendix C - Sprint Review Reports

Overview

This appendix shows highlights of the Sprint Review Meetings carried out during the development of the ArtLens application. Progressive cumulative sprints were concentrated in accomplishing set targets towards the ultimate desired end result.

Sprint Summaries

Sprint 1: Research and Planning

- **Goal**: Establish foundational knowledge for the MET API and identify potential features for ArtLens.
- **Completed User Stories**:
    1. Research public MET API implementation.
    2. Understand data distribution and explore feature possibilities.
- **Outcome**: Developed a clear roadmap for the integration of the MET API into the application.

Sprint 2: Architecture Development

- **Goal**: Define the system architecture and develop initial frontend components.
- **Completed User Stories**:
    1. Research frontend-backend architecture integration.
    2. Create user-friendly frontend designs for MET API features.
- **Outcome**: Delivered a scalable system design and initial prototypes for the user interface.

Sprint 3: Machine Learning Integration

- **Goal**: Implement machine learning-based features for artwork recommendations.
- **Completed User Stories**:
    1. Build a ResNet50-based recommendation system.

2. Develop a Streamlit Dashboard for museum analytics.
- **Outcome**: Machine learning and analytics prototypes were successfully developed.

Sprint 4: Enhancing User Engagement

- **Goal**: Develop features to improve user interactions on mobile and web platforms.
- **Completed User Stories**:
    1. Add mobile app functionality for favoriting artworks.
    2. Design a responsive React-based website.
- **Outcome**: Enhanced user engagement features were implemented.

Sprint 5: Personalized Learning

- **Goal**: Integrate personalized educational content for users.
- **Completed User Stories**:
    1. Add the “Learn More” feature for AI-generated content.
    2. Expand the dashboard with visitor engagement metrics.
- **Outcome**: Improved educational and analytical capabilities of ArtLens.

Sprint 6: Refinement and Testing

- **Goal**: Ensure system stability and optimize performance.
- **Completed User Stories**:
    1. Test the recommendation system, dashboard, and UI components.
    2. Ensure all features meet the product requirements.
- **Outcome**: Delivered a reliable and user-friendly system.

Sprint 7: Final Integration

- **Goal**: Integrate all components and prepare for deployment.
- **Completed User Stories**:
    1. Combine web, mobile, backend, and API systems into a unified application.

2. Prepare for the final product demonstration.

- **Outcome**: Successfully delivered a fully integrated and functional application.

Achievements

- **Successful Completion**: Effective user story planning has led to the effective completion of all the user stories within the designated sprint timeline.
- **Agile Practices**: All team roles in the project participated in and contributed to regular sprint planning, executed daily stand-ups, and had retrospectives focusing on the future of the project.
- **Impact**: The development process was creative and effective such that it enabled the development of a better solution for boosting art appreciation and enhancing the operations of the museum.

Conclusion

The sprint reviews are consistent with the structured and target driven approach of the entire project. This appendix demonstrates the work that has been done by the team to achieve the necessary quality degrees which fulfilled the aims of the project.

Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

-Troubleshooting- Torch not installing on macOs.docx

1. Check Installed Python Versions

To see which versions are available, run in terminal:
python3 –version
which python3

If the desired Python version is not installed, use Homebrew to install it:

brew install python@3.10

Verify:
python3.10 –version

2. Configure Poetry to Use a Specific Python Version

Step A: Remove Existing Environment

If a Poetry virtual environment is already created for the project:
poetry env remove python

Step B: Set the Python Interpreter

Tell Poetry to use the specific Python interpreter by providing the full path to the Python version:
poetry env use /path/to/python3.10

You can find the path to the Python interpreter using:
which python3.10

Step C: Verify the Environment

Check that the correct Python interpreter is being used:
poetry env info

3. Install Project Dependencies

After switching the interpreter, install the dependencies:
poetry install

An error might pop here:

```
The specified Python version (3.10.15) is not supported by the project (^3.12).
Please choose a compatible version or loosen the python constraint specified in the pyproject.toml file.
```

To fix this:
Go to the pyproject.toml file, and change the python version to 3.10

backend > artwork_recommendation > pyproject.toml

```
4   description = "A Machine Learning system for recommending artworks from the MET Museum"
5   authors = ["lquincoso <122310652+lquincoso@users.noreply.github.com>"]
6   readme = "README.md"
7
8   [tool.poetry.dependencies]
9   python = "^3.10"
10  flask = "^3.1.0"
11  torch = "^2.5.1"
12  torchvision = "^0.20.1"
13  pillow = "^11.0.0"
14  requests = "^2.32.3"
15  numpy = "^2.1.3"
16  scikit-learn = "^1.5.2"
17  joblib = "^1.4.2"
18  faiss-cpu = "^1.9.0.post1"
19  diskcache = "^5.6.3"
20  aiohttp = "^3.11.7"
21  flask-cors = "^5.0.0"
22
23
24  [build-system]
25  requires = ["poetry-core"]
```

After this, torch should be installed correctly when running:
poetry install

References

- *Latest updates*. The Metropolitan Museum of Art Collection API. (n.d.). https://metmuseum.github.io/

- Google. (n.d.). *Gemini API | google AI for developers*. Google. http://ai.google.dev/gemini-api/docs

- *React reference overview*. React. (n.d.). https://react.dev/reference/react

- Inc., A. (n.d.). *Documentation*. Swift.org. https://www.swift.org/documentation/

- *Welcome to Python.org*. Python.org. (n.d.). https://www.python.org/doc/

- MozDevNet. (n.d.). *JavaScript: MDN*. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript

- MozDevNet. (n.d.-a). *CSS: Cascading style sheets: MDN*. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/CSS

- MozDevNet. (n.d.-b). *HTML: Hypertext markup language: MDN*. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/HTML

- *Streamlit docs*. Streamlit documentation. (n.d.). https://docs.streamlit.io/