

CS 492
Senior Design Project
2022 Spring



Final Report
PolliVidis

Ömer Ünlüsoy - 21702136
Elif Gamze Güliter - 21802870
İrem Tekin - 21803267
Ece Ünal - 21703149
Umut Ada Yürüten - 21802410

Supervisor: Ercüment Çiçek
Jury Members: Shervin Arashloo and Hamdi Dibeklioglu

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Table of Contents

1. Introduction.....	5
1.1 Overview	6
1.2 Interface Documentation Guidelines	6
1.3 Engineering Standards.....	7
1.4 Definitions, Acronyms, and Abbreviations.....	8
2.Requirement Details.....	9
2.1 Functional Requirements.....	9
2.1.1 System Functionality	9
2.1.2 User Functionality.....	9
2.2 Nonfunctional Requirements.....	10
2.2.1 Usability.....	10
2.2.2 Reliability	10
2.2.3 Privacy and Security.....	10
2.2.4 Efficiency	10
2.2.5 Accessibility	10
2.2.6 Extensibility	11
2.3 Pseudo Requirements:	11
2.3.1 Version Control and Management:	11
2.3.2 Implementation.....	11
3. Final Architecture and Design Details.....	12
3.1. Packages.....	12
3.1.0 Introduction.....	12
3.1.1 Client.....	14
3.1.2 Server.....	16
3.1.2.1 Logic Tier.....	16
3.1.2.2 Data Tier	21
3.2. Class Interfaces.....	23
3.2.0 Introduction.....	23
3.2.1 Client.....	23
3.2.1.1 Presentation Tier	23
3.2.2 Server.....	29
3.2.2.1 Logic Tier.....	29
3.2.2.2 Data Tier	41
3.3 APIs and Tools Used.....	46
3.3.1 Django REST API Framework.....	46
3.3.2 Google Firebase Storage Service.....	46
3.3.3 PyTorch.....	46
3.3.4 ScKit Learn	47
3.3.5 Cv2	47

3.3.6 Matplotlib.....	47
3.3.7 MySQL.....	47
3.3.8 Dijkstra Server.....	47
3.3.9 React.....	48
3.3.10 Google Maps API.....	48
4. Development and Implementation Details.....	49
4.1 Image Processing Implementation.....	49
4.2 Machine Learning Implementation.....	52
4.3 Backend and Database Implementation.....	56
4.4 UI Implementation.....	58
5. Testing Details.....	60
5.1 Testing of Image Processing Procedure.....	60
5.2 Testing of Machine Learning Procedure.....	60
5.3 Testing of Backend and Frontend.....	60
6. Maintenance Plan and Details.....	61
7. Other Project Elements.....	61
7.1 Consideration of Various Factors in Engineering Design.....	61
7.2 Ethics and Professional Responsibilities.....	64
7.3 Judgements and Impacts to Various Contexts.....	64
7.4 Teamwork Details.....	67
7.4.1 Ömer Ünlüsoy.....	67
7.4.2 Gamze Güliter.....	67
7.4.3 İrem Tekin.....	67
7.4.4 Ece Ünal.....	67
7.4.5 Umut Ada Yürüten.....	67
7.5 New Knowledge Acquired and Applied.....	68
8. Conclusion and Future Work.....	68
9. User Manual and Installation.....	69
9.1 About Installation.....	69
9.2 Pollen Map.....	69
9.3 Registration (Sign Up).....	70
9.4 Academic Login.....	71
9.5 Navigation Menu without Academic Login.....	72
9.6 Navigation Menu with Academic Login.....	72

9.7 Analyze Sample Page	73
9.7.1 Upload Image Page.....	75
9.7.2 Analysis Report Page	75
9.8 Previous Analyses Page	76
9.9 Profile Page	77
9.10 About Us Page	78
9.11 Download Dataset Page.....	79
9.12 Feedback Page	79
9.13 How PolliVidis Works Page.....	80
<i>10.References.....</i>	<i>81</i>

1. Introduction

Along with the developments in machine learning, most systems in natural sciences are transforming into automated models. Nevertheless, currently there is no available model to identify pollen species for palynology, the branch of biology which examines pollens. Methodology to identify and count pollen species in a light microscope sample has not changed since the invention of microscopy, the brute force method requiring one palynologist to count each pollen on the sample one by one. Additionally, training new students to identify pollens requires further effort, especially distinguishing pollens with similar granular cell structures.

There are millions of people having seasonal pollen allergies. That's why most weather reports try to give the current pollen densities in the air regionally. However, such information requires palynologists to take samples from the air and analyze the sample with brute force daily, which takes hours. Such restriction in the pollen analysis hinders a proper weather report and decreases the life quality of the people with pollen allergies.

Although there have been attempts to create such a machine learning model in order to identify pollen in a given data, currently there is no widely available model to identify pollen types for palynology. This conclusion created PolliVidis, the first and only website for palynologists, palynology students, and people with pollen allergies to analyze their pollen sample with a trained deep learning architecture and share their pollen analysis with everyone, accelerating the developments in palynology. With the system we will develop, we aim to fill the role of a publicly available web application for pollen identification in Turkey which would help academics to easily classify the pollen they research and furthermore aid students trying to learn the details of palynology.

Beside the trained deep learning architecture, PolliVidis offers a Pollen Map where every analysis shows up to create an accumulated regional pollen density information for the people with pollen allergies.

We have decided to take the challenging road and to create our own pollen dataset from scratch. A little research showed that the largest pollen research in Turkey is being conducted by Ankara University. Our team has contacted Researcher Aydan Acar at Ankara University and convinced them to do a joint study. She has taught us how to use a light microscope and given us access to her pollen species collection. We have selected the most famous and allergenic pollen species in Turkey for this dataset. Over six months, our team has collected 23 different species and more than 6.000 single pollen samples. Hence, along with the web application, PolliVidis publishes a dataset of 6000 pollen samples which was not available before.

1.1 Overview

PolliVidis is a pollen classification platform for academics, students and people with pollen allergies. It is a web application which makes it widely accessible to interested users. PolliVidis provides a pollen map built by the academic community's pollen analyses. Moreover, it allows data sharing between academics which will contribute to building a vast pollen database and help palynology research. Furthermore, PolliVidis has three main promises: classification and analysis of given pollen samples with a trained ML model, creation of a PolliVidis database which can help further palynology research and academic information exchange, and construction of a pollen map of Turkey. In addition, using the pollen map people with pollen allergies can track the frequency dates, and locations of these pollen and schedule their visits. In PolliVidis everyone, without the need of an account, is able to access the pollen map and learn our analysis, allergenic pollen information based on time and location. Moreover, everyone can use our model to analyze their own pollen sample via the website. However, only academics can update the pollen map with their samples and uploading data to the map requires an academic account. Login into these academic accounts is required to protect the accuracy and the reliability of the map. Furthermore, the pollen analysis consists of pollen classification and counting. After uploading a sample, the user can learn the pollen types and their ratio (number) in their samples. If academics agree to share their sample analysis, the pollen map will be updated with their analysis. Moreover, students are also able to use the PolliVidis for educational purposes, such as uploading pollen photos and learning their type without consulting their instructor.

1.2 Interface Documentation Guidelines

The following template is used for each class definition in this report. The class name and the description are stated first, then the attributes of the class are given and finally methods of the class are explained.

Class Name	
Class Description	
Attributes	
Attribute Type : Attribute Name	
Methods	
MethodName (Parameters) : Return Type	Method Description

1.3 Engineering Standards

Final Report diagrams follow the UML guidelines [1] as the previous reports of PolliVidis.

Until this report; PolliVidis has used UML Use Case Diagram to describe user interactions with the web application [2], UML Sequence Diagram to describe the lifeline of an object [3], UML Activity Diagram to describe the control flow of the system [4], and UML Deployment Diagram to define the hardware communication of the system [5] and UML Class Diagram to describe statically the structure of the system [6]. As the usage of UML is standard in the industry, all diagrams follow the UML guidelines.

For the citations in the reports, IEEE Citation guidelines are followed as it is the standard in engineering [7].

1.4 Definitions, Acronyms, and Abbreviations

Palynology: Branch of biology studying pollen.

Academic: User with a pollen or biology related background such as palynologists, biologists, and palynology or biology students.

Allergenic Pollen: Specific pollen types that humans can develop allergies to.

Sample / Pollen Sample: Pollen image shot by a light microscope containing a few pollens.

Noise: All other shapes in the sample rather than pollen itself such as spores.

Sample Analysis: Procedure of identifying pollen types, classification, from a sample using machine learning.

Analysis Report: Report containing pollen information generated by the pollen analysis.

Pollen Map: Google Maps supported map showing pollen information and distribution of Turkey.

Pollen Extraction: Process of extracting a single pollen image from the sample with few pollens using the Pollen Extraction Algorithm coded by us.

(Ankara) Dataset: Pollen dataset, collection of pollen images, created by us in Ankara University.

PolliVidis Database: MySQL based database to store all (allowed) uploaded pollen samples with their analyses in order to construct the Pollen Map.

CNN: Convolutional Neural Networks

Transfer Learning: Using pre-trained networks such as AlexNet or VGG-19 to boost the classification.

Data Augmentation: Manipulating dataset to avoid overfitting and increasing accuracy.

Google Maps API: used API for the Pollen Map of PolliVidis.

Django: Python package for website backend.

React: JavaScript library for website frontend.

MySQL: Relational database management system for SQL.

2.Requirement Details

2.1 Functional Requirements

2.1.1 System Functionality

The system should:

- detect and classify allergen pollen types found in Turkey.
- should count the pollen types and density in the uploaded pollen samples.
- display the sample analysis report after the sample upload.
- require academic registration for academic usage.
- not require registration for pollen analysis.
- not require registration for pollen map usage.
- allow any user to upload pollen sample to analyze.
- should store the pollen samples that are uploaded by academic users.
- display pollen map supported by Google Maps.
- display pollen samples and their analyses reports on the map.
- allow academics to look through the pollen analyses with its publisher information.

2.1.2 User Functionality

The academic user should:

- register to the system with required information.
- login in order to contribute to the Pollen Map.
- view other pollen samples and their analysis with its publisher information.
- upload pollen samples to analyze.
- able to see the count, density, and the species of the sample from the analysis.
- look through the pollen analyses with its publisher information.

The anonymous user should:

- not register to the system to analyze any pollen samples.
- not register to the system to look through the Pollen Map.

2.2 Nonfunctional Requirements

2.2.1 Usability

The system should:

- be able to work on most search engines such as Safari, Chrome, Firefox, and Mozilla.
- yield an analysis with the pollen samples taken under a light microscope within 10 seconds.
- allow pollen analysis without any registration or login.
- allow Pollen Map usage without any registration or login

2.2.2 Reliability

The system should

- ensure that the pollen map data that it offers is reliable and obtained from a palynologist or academic's samples.
- ensure reliable results (more than %98 accuracy) for the pollen classification.
- should not lose any pollen data unless the user deletes or doesn't let data to be added to the database

2.2.3 Privacy and Security

The system should:

- require passwords that contain uppercase and lowercase letters, and have at least 8 characters with a mixture of both numbers and letters.
- ensure that the user's data is safe by not storing their password directly but hashing it with the Google recommended hashing algorithm SHA-256 [8]
- get permission from academic users to share the location and the date of the samples with other users.
- not require any personal information from unregistered users to analyze pollen and look through the pollen map.

2.2.4 Efficiency

- The webpage's loading time should not exceed 2 seconds which is the maximum loading time recommended by Google [9].
- Analysis will be conducted on the server rather than the user's own computer which should decrease memory usage.

2.2.5 Accessibility

The system should be:

- available on most used internet browsers (Safari, Chrome, Firefox, Microsoft Edge, and Mozilla).

2.2.6 Extensibility

- The ML model can be improved in the future with more datasets available.
- The allergenic pollen in Turkey will be examined to scale the project. However, the pollen types can be increased in the future.
 - Pollen Map is based on Turkey where the examined pollen is located. This map can be extended as global in the future.

2.3 Pseudo Requirements:

2.3.1 Version Control and Management:

- GitHub and Git will be used for version control.
- Google Docs will be used and will be used for project management and source sharing.
- Google CoLab will be used for collaboratory implementations
- GitHub pages will be used as a project website.

2.3.2 Implementation

- PolliVidis will be implemented as a website.
- The system will have server-client architecture.
- Python will be used in the back-end.
- For the Machine Learning model, CNN will be used.
- PyTorch package will be used for implementing convolutional neural networks.
- Python Django and React frameworks will be used for building the website.
- MySQL will be used in database design.
- AlexNet pre-trained model was used as transfer learning strategy
- All dataset will be collected by us, in Ankara University Science Faculty labs.
- Dataset will be stored in the database of the project written in MySQL.
- Object Oriented Programming (OOP) paradigms was followed during the implementation

3. Final Architecture and Design Details

3.1. Packages

3.1.0 Introduction

PolliVidis takes advantage of some external packages, libraries, which allow the system to be more dynamic and optimized while taking away the burden to code everything from scratch.

The first and the most significant package PolliVidis uses is PyTorch [10]. This package is a machine learning package developed by Facebook, allowing its users to construct and train neural networks easily. PolliVidis will use this package to architect its CNN and train it with the dataset we created.

The second package is SCikit-Image which PolliVidis uses for pollen extraction from sample images. It is basically an image processing toolbox [11].

For the frontend, PolliVidis uses React Library [12]. React is a JavaScript library for building user interfaces easily.

To connect the UI with the database and ML model, PolliVidis has taken advantage of the Django framework which allows its users to create web apps and connect them with their server [13].

Lastly, Google Maps API is used for PolliVidis Pollen Map [14]. This API allowed us to construct and modify a web app in PolliVidis website.

After explaining the external packages used, let's see the internal structure of the web app. PolliVidis mimics 3-Tier Client/Server Architecture.

On the client side, the system implements the Presentation Tier which contains the UI Subsystem. This subsystem is implemented with React and manages UI components. This subsystem sends queries to the server to handle user requests and shows the results of the queries to the user.

On the server side of the system, the system implements Logic and Data Tiers. In the Logic Tier, there are two subsystems, namely Backend Subsystem and ML Subsystem. Backend Subsystem is implemented with the Python Django Framework. It directs queries coming from the client side to the Data Tier and handles user requests. This subsystem uses the ML Subsystem to extract pollen images from the sample image and analyze them one by one. It returns the analyses to the user.

ML Subsystem is the core of PolliVidis and implements two main functionalities, pollen extraction from the sample image and pollen classification with PyTorch. The classification is done by a CNN and pollen extraction uses Image Processing with the SCikit-Image package.

In the Logic Tier, a single subsystem named Database Subsystem handles database interactions of PolliVidis. It implements the MySQL database and all queries within it and supplies Python Model classes for ease of use.

3.1.1 Client

3.1.1.1 Presentation Tier

3.1.1.1.1 UI Subsystem

UI Subsystem consists of user interface front-end views.

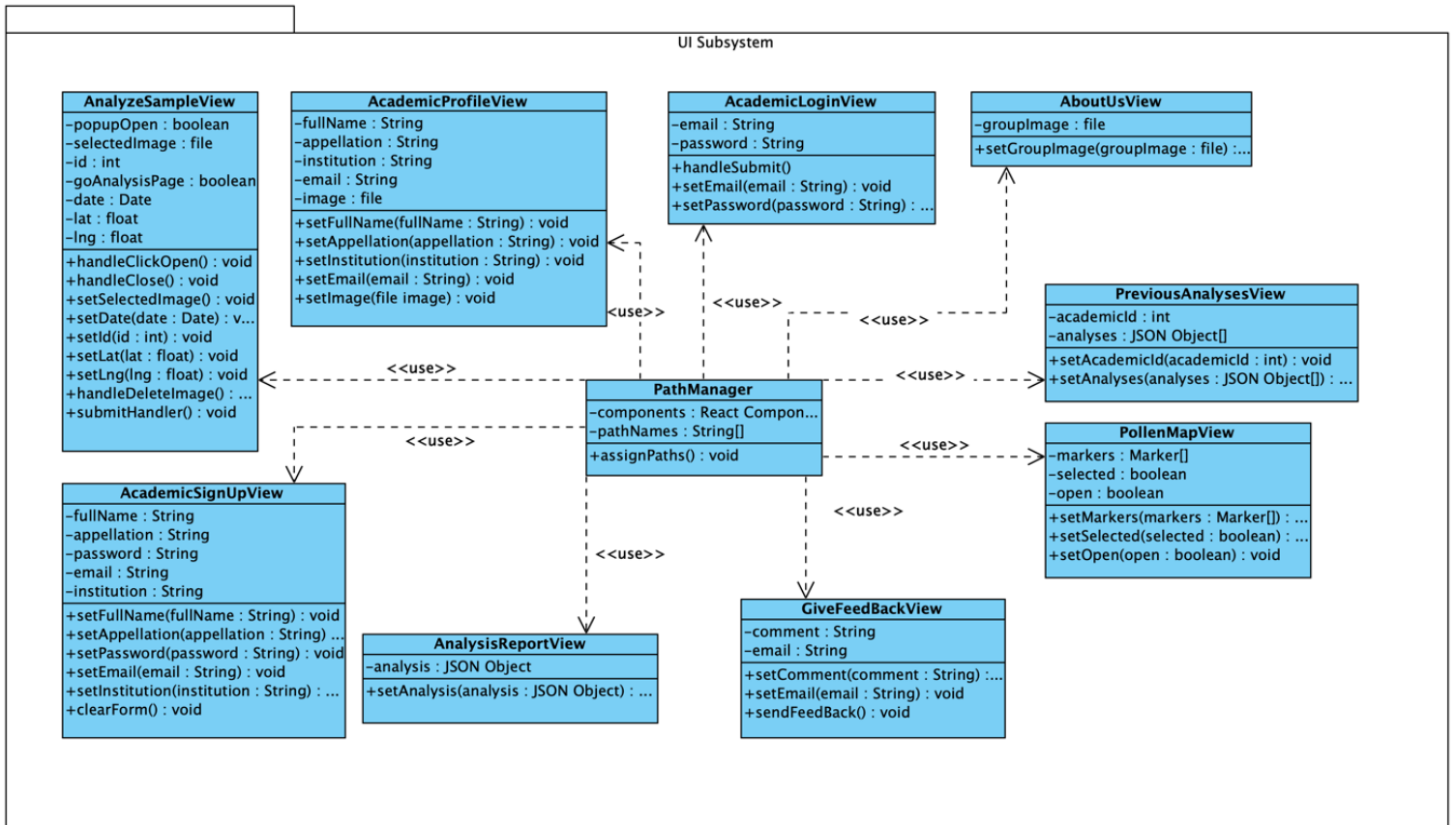


Figure 1: UI Subsystem

3.1.1.1.1.1 Path Manager

Path Manager handles the navigation and main structure of the frontend. It controls the presentation tier.

3.1.1.1.1.2 AnalyzeSampleView

AnalyzeSampleView handles the user interface of the screen in which users upload sample images and request an analysis.

3.1.1.1.1.3 AnalysisReportView

AnalysisReportView handles the user interface of the screen in which the analysis report of the users' samples is shown.

3.1.1.1.1.4 PollenMapView

PollenMapView handles the user interface of the Google Maps pollen map which contains the pollen analyses as markers. When clicked on one, the analysis report of the analysis is shown.

3.1.1.1.1.5 PreviousAnalysisView

PreviousAnalysisView handles the user interface of the screen in which an academic's previous analysis reports are shown.

3.1.1.1.1.6 AcademicLoginView

AcademicLoginView handles the user interface of the screen in which an academic can login.

3.1.1.1.1.7 AcademicSignUpView

AcademicSignUpView handles the user interface of the screen in which a user can sign up as an academic.

3.1.1.1.1.8 AcademicProfileView

AcademicProfileView handles the user interface of the screen that shows the profile information of an academic.

3.1.1.1.1.9 AboutUsView

AboutUsView handles the user interface of the screen that shows PolliVidis developers' information.

3.1.1.1.1.10 GiveFeedBackView

GiveFeedBackView handles the user interface of the screen in which users can send feedback about PolliVidis.

3.1.2 Server

3.1.2.1 Logic Tier

3.1.2.1.1 Backend Subsystem

Backend Subsystem mainly consists of firstly model objects used both in data and client level, secondly the serializer classes of the model objects for Http responses to the front end, and lastly the handler classes that process the request and respond accordingly.

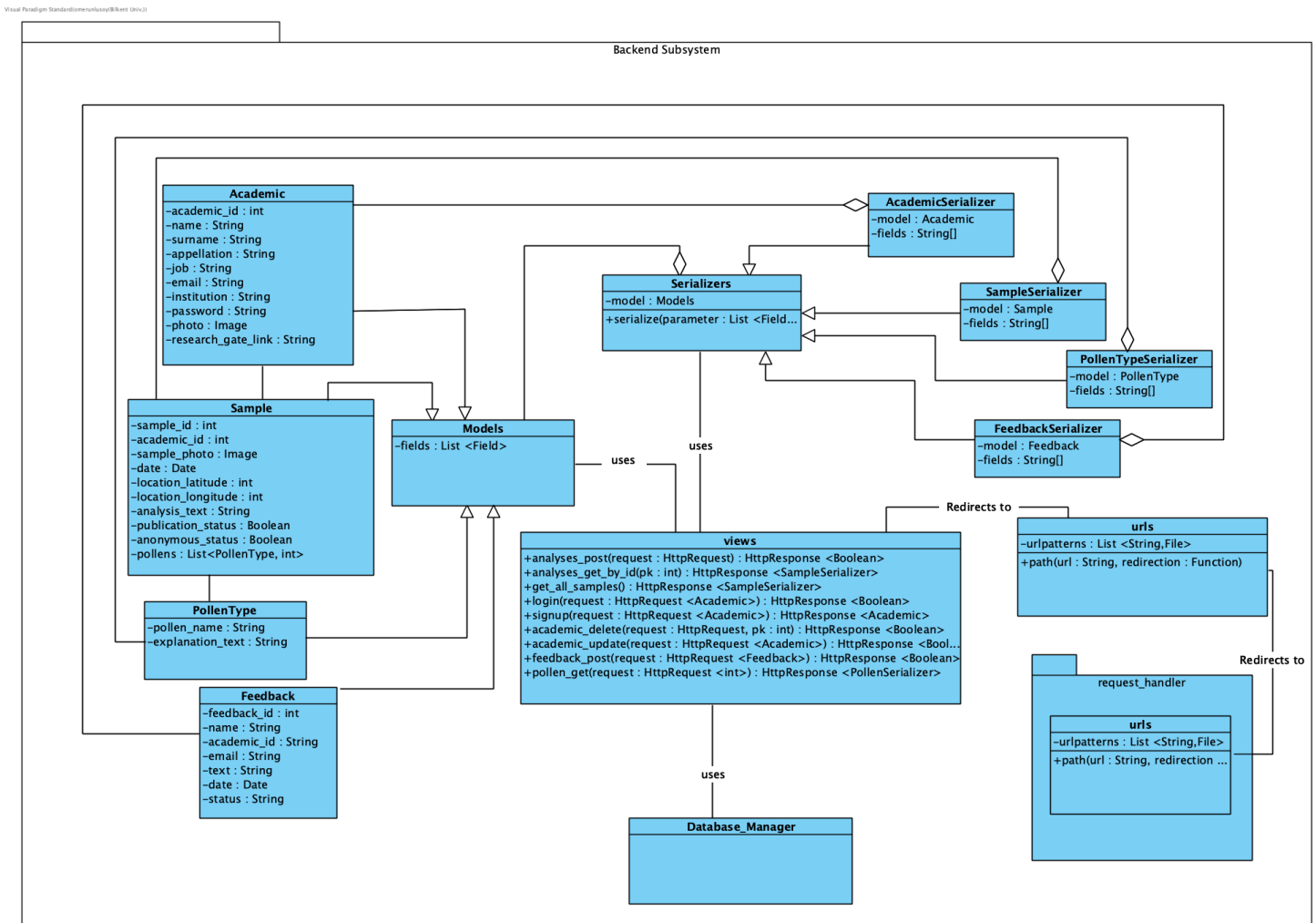


Figure 2: Backend Subsystem

3.1.2.1.1.1 Academic

Contains the class of Academic model which has been used both in client and data level. It is used for keeping consistency in all levels as well as defining the Http response for the client side.

3.1.2.1.1.2 Sample

Contains the class of Sample model which has been used both in client and data level. It is used for keeping consistency in all levels as well as defining the Http response for the client side.

3.1.2.1.1.3 PollenType

Contains the class of PollenType model which has been used both in client and data level. It is used for keeping consistency in all levels as well as defining the Http response for the client side.

3.1.2.1.1.4 Feedback

Contains the class of Feedback model which has been used both in client and data level. It is used for keeping consistency in all levels as well as defining the Http response for the client side.

3.1.2.1.1.5 Models

Contains the general class for models that is recognized by the REST django API.

3.1.2.1.1.6 AcademicSerializer

Contains a serializable version of the Academic class which is used to convert the Academic objects into proper Http Response bodies.

3.1.2.1.1.7 SampleSerializer

Contains a serializable version of the Sample class which is used to convert the Academic objects into proper Http Response bodies.

3.1.2.1.1.8 PollenTypeSerializer

Contains a serializable version of the PollenTypeclass which is used to convert the Academic objects into proper Http Response bodies.

3.1.2.1.1.9 FeedbackSerializer

Contains a serializable version of the Feedback class which is used to convert the Academic objects into proper Http Response bodies.

3.1.2.1.1.10 Serializer

Contains the general class for serializer classes that is recognized by the REST django API.

3.1.2.1.1.11 request_handler.urls

This class handles requests that arrive from the client side of the project. This class maps the given url to the respective handler, and redirects to that file. Although most of the requests are directed to the main API, this handler allows debugging via redirecting to admin pages. Furthermore, this file could be expanded to include different types of requests from different types of users.

3.1.2.1.1.12 urls

Much like the `request_handler` package in 2.2.1.1.11, this class maps request urls. However, unlike its counterpart, this url handler maps the Http requests to their respective functions in views file as it is explained in 2.2.1.1.13.

3.1.2.1.1.13 views

This class is the central part of the backend subsystem and contains functions that handles, processes and responds to the requests made by the client level. As explained in 2.2.1.1.12, `urls` class redirects a request to a proper function in this class. In each function, the request is transformed and acknowledged with proper Model classes. Next, methods from `Database_Manager` are used for database operations. Details of the `Database_Manager` are further explained in 2.2.2.1, hence it is represented as a blackbox class. After acquiring the results from the database, an `HttpResponse` is formed via `Serializer` classes and then sent back to client side.

3.1.2.1.2 ML Subsystem

ML Subsystem has two main functionalities; pollen classification with PyTorch and pollen image extraction from the incoming sample image.

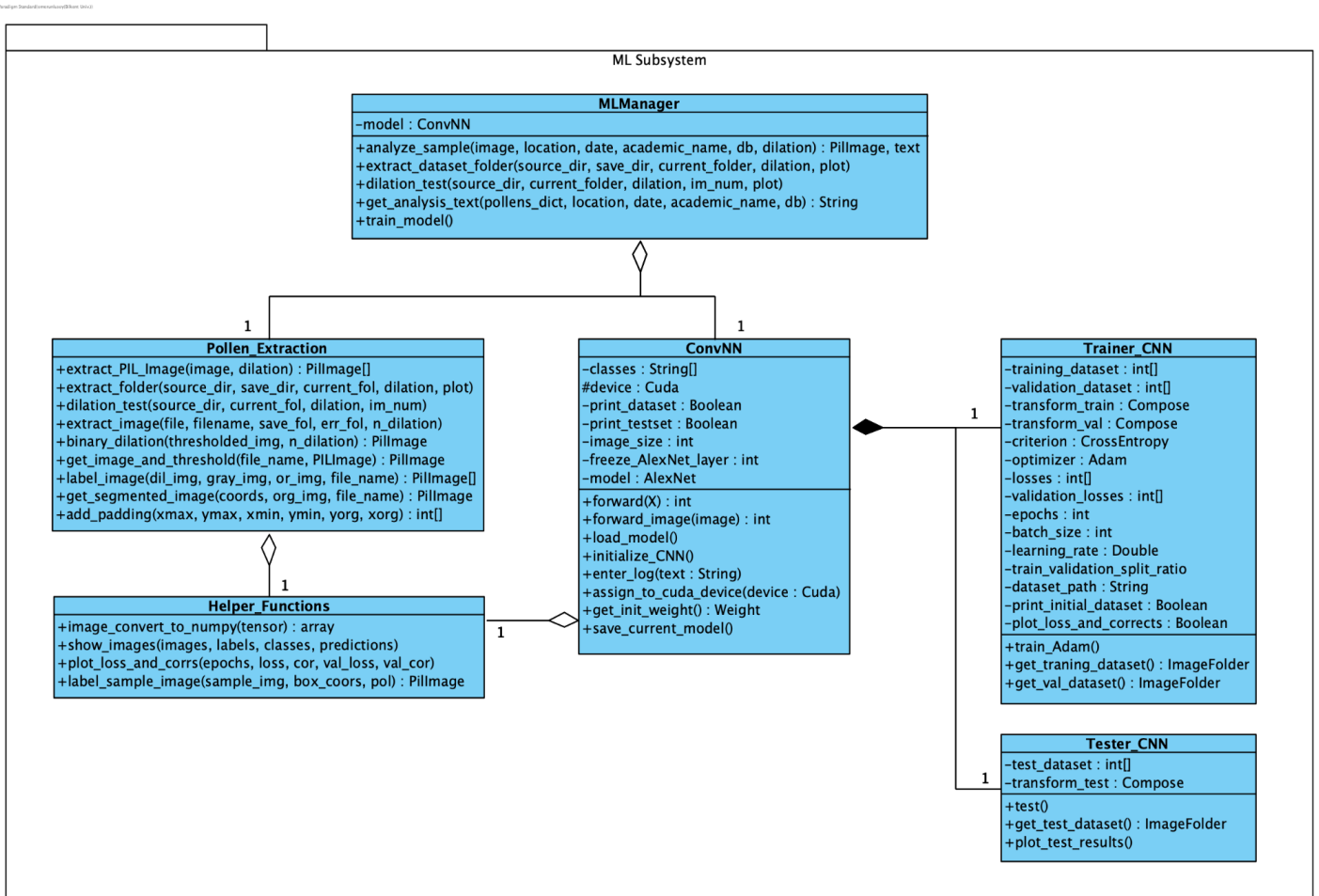


Figure 3: ML Subsystem

3.1.2.1.2.1 ML Manager

ML Manager class is the driver class of this subsystem, it uses Pollen Extraction and ConvNN classes to respond to a client request. Thus, it handles the two main functionalities of this subsystem, namely pollen classification and pollen image extraction. This manager class holds the trained model and uses ConvNN class to predict and classify the incoming pollens from the client. Moreover, it processes the sample image and extracts pollen images using the Pollen Extraction class.

3.1.2.1.2.2 Pollen Extraction

This class implements the pollen extraction algorithm, using image processing and dilation with Python SCikit-Image package. This extraction algorithm is used in two scenarios; when the pollen dataset of PolliVidis is prepared and ready to be pre-processed before going into the training algorithm, and when the client sends a sample image with a few pollens in it required to be pre-processed before the classification. Thus, this class can process a single image and folders of images at the same time. The procedure of this algorithm is explained in detail in another section of this report.

3.1.2.1.2.3 ConvNN

ConvNN is the class of the ML model which implements the Convolutional Neural Network. This class holds the hyperparameters of the architecture, uses Trainer class to train its model, and saves the trained model for later use. The predictions of the model are made in this class.

3.1.2.1.2.4 Trainer_CNN

This class implements the training procedure of the model. The sole reason for this functionality to be implemented as a separate class is ease of use.

3.1.2.1.2.5 Tester_CNN

This class implements the testing procedure of the model and calculates the evaluation matrices.

3.1.2.1.2.6 Helper_Functions

This class is a helper class used by most classes in this subsystem. It implements general purpose functionalities such as printing, plotting, and converting images. Its implementation simplifies the subsystem.

3.1.2.2 Data Tier

3.1.2.2.1 Database Subsystem

Database Subsystem deals with the usage and the management of the database for the application.

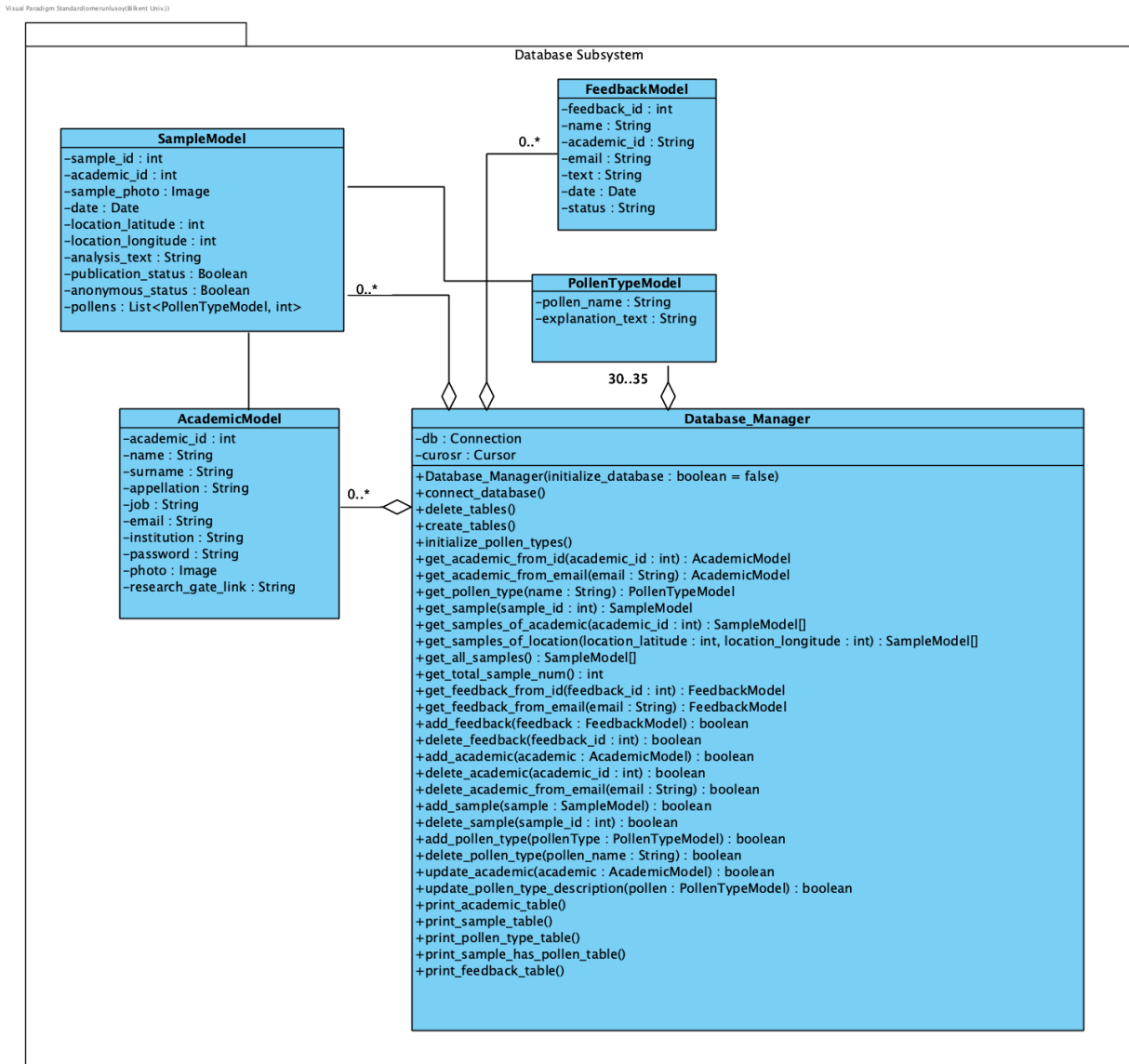


Figure 4: Database Subsystem

3.1.2.2.1.1 AcademicModel

Contains the class of AcademicModel, which has been used both in client and data level. It is used to acknowledge the data received from the backend as well as sent and acquired from the database.

3.1.2.2.1.2 SampleModel

Contains the class of SampleModel, which has been used both in client and data level. It is used to acknowledge the data received from the backend as well as sent and acquired from the database.

3.1.2.2.1.3 PollenTypeModel

Contains the class of PollenTypeModel, which has been used both in client and data level. It is used to acknowledge the data received from the backend as well as sent and acquired from the database.

3.1.2.2.1.4 FeedbackModel

Contains the class of FeedbackModel, which has been used both in client and data level. It is used to acknowledge the data received from the backend as well as sent and acquired from the database.

3.1.2.2.1.5 Database_Manager

This class is the main processor of the Database Subsystem. It is used to connect to the database, initialize it and then execute commands for utilizing the database. It uses other Model classes to send and receive data from the database, in which the tables correspond with the model objects.

3.2. Class Interfaces

3.2.0 Introduction

In the Class Interfaces section, attributes and methods of each class will be given with the method signatures and detailed explanations.

3.2.1 Client

3.2.1.1 Presentation Tier

3.2.1.1.1 UI Subsystem

Class PathManager	
Path Manager handles the navigation and main structure of the frontend. It controls the presentation tier.	
Attributes	
private React Component[] components	
private String[] pathNames	
Methods	
void assignPaths()	Assigns paths to components (pages)

Class AnalyzeSampleView	
AnalyzeSampleView handles the user interface of the screen in which users upload sample images and request an analysis.	
Attributes	
private boolean popupOpen	
private file selectedImage	
Private int id	

Private boolean goAnalysisPage	
Private Date date	
Private float lng	
Private float lat	
Methods	
void handleClickOpen()	Opens the popup screen for selecting image
void handleClose()	Closes the popup screen for selecting image
void setSelectedImage()	Sets the attribute selected image
void setDate(Date date)	Sets the attribute date
void setId(int id)	Sets the attribute id
void setLat(float lat)	Sets the attribute lat
void setLng(float lng)	Sets the attribute lng
Void handleDeleteImage()	Deletes the selected image, sets selected image null
Void submitHandler()	Sends the selected image, date, lat, lng, id to the server

Class AnalysisReportView	
AnalysisReportView handles the user interface of the screen in which the analysis report of the users' samples is shown.	
Attributes	
Private JSON Object analysis	
Methods	
void setAnalysis(JSON Object analysis)	Sets the analysis information coming from the server to the attribute analysis

Class PollenMapView	
PollenMapView handles the user interface of the Google Maps pollen map which contains the pollen analyses as markers. When clicked on one, the analysis report of the analysis is shown.	
Attributes	
Private Marker[] markers	
Private boolean selected	
Private boolean open	
Methods	
Void setMarkers(Marker[] markers)	Sets the marker locations coming from the server to the markers array
Void setSelected(boolean selected)	Sets the attribute selected. Used when a marker is clicked by the user.
Void setOpen(boolean open)	Sets the attribute open. Opens a left drawer when a marker is clicked and shows analysis information corresponding to that marker.

Class PreviousAnalysesView	
PreviousAnalysisView handles the user interface of the screen in which an academic's previous analysis reports are shown.	
Attributes	
Private int academicId	
Private JSON Object[] analyses	
Methods	
void setAnalyses(JSON Object[] analyses)	Sets the previous analyses information coming from the server to the attribute analyses

Void setAcademicId(int academicId)	Sets the attribute academicId
------------------------------------	-------------------------------

Class AcademicLoginView	
AcademicLoginView handles the user interface of the screen in which an academic can login.	
Attributes	
Private String email	
Private String password	
Methods	
void handleSubmit()	Send email and password information to the server
Void setEmail(String email)	Sets the attribute email
Void setPassword(String password)	Sets the attribute password

Class AcademicSignUpView	
AcademicSignUpView handles the user interface of the screen in which a user can sign up as an academic.	
Attributes	
Private String fullName	
Private String appellation	
Private String password	
Private String email	
Private String institution	
Methods	
void setFullName(String fullName)	Sets the attribute fullName

void setAppellation(String appellation)	Sets the attribute appellation
void setPassword(String password)	Sets the attribute password
void setEmail(String email)	Sets the attribute email
void setInstitution(String institution)	Sets the attribute institution
Void clearForm()	Sets all the attributes to null

Class AcademicProfileView	
AcademicProfileView handles the user interface of the screen that shows the profile information of an academic.	
Attributes	
Private String fullName	
Private String appellation	
Private file image	
Private String email	
Private String institution	
Methods	
void setFullName(String fullName)	Sets the attribute fullName
void setAppellation(String appellation)	Sets the attribute appellation
void setEmail(String email)	Sets the attribute email
void setInstitution(String institution)	Sets the attribute institution
Void setImage(file image)	Sets the attribute image

Class AboutUsView	
AboutUsView handles the user interface of the screen that shows PolliVidis developers' information.	
Attributes	
Private file groupImage	
Methods	
void setGroupImage(file groupImage)	Sets the attribute groupImage

Class GiveFeedBackView	
GiveFeedBackView handles the user interface of the screen in which users can send feedback about PolliVidis.	
Attributes	
Private String comment	
Private String email	
Methods	
void setComment(String comment)	Sets the attribute comment
void setEmail(String email)	Sets the attribute email
Void sendFeedBack()	Sends the comment and email information to the server

3.2.2 Server

3.2.2.1 Logic Tier

3.2.2.1.1 Backend Subsystem

Class Academic	
Contains the class of Academic model which has been used both in client and data level. It is used for keeping consistency in all levels as well as defining the Http response for the client side.	
Attributes	
private int	academic_id
private String	name
private String	surname
private String	appellation
private String	job
private String	mail
private String	institution
private String	password
private Image	photo
private String	research_gate_link

Class Sample

Contains the class of Sample model which has been used both in client and data level. It is used for keeping consistency in all levels as well as defining the Http response for the client side.

Attributes

private int sample_id

private int academic_id

private Image sample_photo

private Date date

private int location_latitude

private int location_longitude

private String analysis_text

private Boolean publication_status

private Boolean anonymous_status

private String research_gate_link

private List<PollenTypeModel,int> pollens

Class PollenType

Contains the class of PollenType model which has been used both in client and data level. It is used for keeping consistency in all levels as well as defining the Http response for the client side.

Attributes

private String pollen_name

private String explanation_text

Class Feedback

Contains the class of Feedback model which has been used both in client and data level. It is used for keeping consistency in all levels as well as defining the Http response for the client side.

Attributes

private int feedback_id

private String name

private int academic_id

private String email

private String text

private Date date

private String status

Class Models

Contains the general class for models that is recognized by the REST django API.

Attributes

```
private List<Field> fields
```

Class AcademicSerializer

Contains a serializable version of the Academic class which is used to convert the Academic objects into proper Http Response bodies.

Attributes

```
private Academic model
```

```
private String[] fields
```

Class SampleSerializer

Contains a serializable version of the Sample class which is used to convert the Academic objects into proper Http Response bodies.

Attributes

```
private Sample model
```

```
private String[] fields
```

Class PollenTypeSerializer	
Contains a serializable version of the PollenType class which is used to convert the Academic objects into proper Http Response bodies.	
Attributes	
private PollenType model	
private String[] fields	

Class FeedbackSerializer	
Contains a serializable version of the Feedback class which is used to convert the Academic objects into proper Http Response bodies.	
Attributes	
private Feedback model	
private String[] fields	

Class Serializer	
Contains the general class for serializer classes that is recognized by the REST django API.	
Attributes	
private Models model	
Methods	
serialize(parameter : List <Fields>)	Serializes the given model according to the its fields

Class request_handler.urls

This class handles requests that arrive from the client side of the project. This class maps the given url to the respective handler, and redirects to that file. Although most of the requests are directed to the main API, this handler allows debugging via redirecting to admin pages. Furthermore, this file could be expanded to include different types of requests from different types of users.

Attributes

private List<String, File> urlpatterns

Methods

path(url : String, redirection : File)

Redirects given string url to mapped file

Class urls

Much like the request_handler package in 2.2.1.1.11, this class maps request urls. However, unlike its counterpart, this url handler maps the Http requests to their respective functions in views file as it is explained previously.

Attributes

private List<String, File> urlpatterns

Methods

path(url : String, redirection : Function)

Redirects given string url to the mapped function

Class views

This class is the central part of the backend subsystem and contains functions that handles, processes and responds to the requests made by the client level. As explained in 2.2.1.1.12, urls class redirects a request to a proper function in this class. In each function, the request is transformed and acknowledged with proper Model classes. Next, methods from Database_Manager are used for database operations. Details of the Database_Manager are further explained in 2.2.2.1, hence it is represented as a blackbox class. After acquiring the results from the database, an HttpResponse is formed via Serializer classes and then sent back to the client side.

Methods

analyses_post(request : HttpRequest) : HttpResponse <Boolean>	Inserts the given analysis to the database. Returns true if successful, otherwise false
analyses_get_by_id(pk : int) : HttpResponse <SampleSerializer>	Returns the analysis with the given id
get_all_samples() : HttpResponse <SampleSerializer>	Returns all uploaded samples from the database
login(request : HttpRequest <Academic>) : HttpResponse <Boolean>	Attempts to find matching Academic in the database. Returns the user if found, null otherwise.
signup(request : HttpRequest <Academic>) : HttpResponse <Academic>	Attempts to insert a user into the database. Returns the user if successful, null otherwise.
academic_delete(request : HttpRequest, pk : int) : HttpResponse <Boolean>	Attempts to remove an Academic user with the given properties. Returns true if successful, false otherwise.
academic_update(request : HttpRequest <Academic>) : HttpResponse <Boolean>	Changes the attributes of an Academic tuple in the database. Returns true if successful, false otherwise.
feedback_post(request : HttpRequest <Feedback>) : HttpResponse <Boolean>	Inserts new feedback into the database. Returns true if successful, false otherwise.
pollen_get(request : HttpRequest <int>) : HttpResponse <PollenSerializer>	Gets a pollen of PollenType from the database with the given parameters. Returns the pollen if successful, null otherwise.

3.2.2.1.2 ML Subsystem

Class ML_Manager	
<p>ML Manager class is the driver class of this subsystem, it uses Pollen Extraction and ConvNN classes to respond to a client request. Thus, it handles the two main functionalities of this subsystem, namely pollen classification and pollen image extraction. This manager class holds the trained model and uses ConvNN class to predict and classify the incoming pollens from the client. Moreover, it processes the sample image and extracts pollen images using the Pollen Extraction class.</p>	
Attributes	
private ConvNN model	
Methods	
analyze_sample(image, location, date, academic_name, db, dilation) : Pillmage, text	This function gets the image from the client side via backend and calls ConvNN forward function to classify each pollen after extracting them from the sample. It returns the analyzed image with the analysis text.
extract_dataset_folder(source_dir, save_dir, current_folder, dilation, plot)	This function calls the Pollen_Extraction class to process a dataset folder.
dilation_test(source_dir, current_folder, dilation, im_num, plot)	This function calls the dilation test of the extractor.
get_analysis_text(pollens_dict, location, date, academic_name, db) : String	This function constructs the analysis text from the classification results.
train_model()	This function calls the training procedure of Training_CNN.

Class Pollen_Extraction	
<p>This class implements the pollen extraction algorithm, using image processing and dilation with Python SCikit-Image package. This extraction algorithm is used in two scenarios; when the pollen dataset of PolliVidis is prepared and ready to be pre-processed before going into the training algorithm, and when the client sends a sample image with a few pollens in it required to be pre-processed before the classification. Thus, this class can process a single image and folders of images at the same time. The procedure of this algorithm is explained in detail in another section of this report.</p>	
Methods	
extract_PIL_Image(image, dilation) : PillowImage []	This function extracts pollens from the single given image and is used for the client sample images.
extract_folder(source_dir, save_dir, current_fol, dilation, plot)	This function processes the entire folder for pollen extraction for the pre-processing for the training.
dilation_test(source_dir, current_fol, dilation, im_num)	This function tests for the best dilation value for a given image.
extract_image(file, filename, save_fol, err_fol, n_dilation)	This function extracts a single image, and is called by extract_PIL_Image.
binary_dilation(thresholded_img, n_dilation) : PillowImage	This function applies binary dilation to given image.
get_image_and_threshold(file_name, PillowImage) : PillowImage	This function loads the image from a filepath and applies thresholding to the image.
label_image(dil_img, gray_img, or_img, file_name) : PillowImage []	This function labels the binary thresholded image to separate regions for pollen extraction.
get_segmented_image(coords, org_img, file_name) : PillowImage	This image applies segmentation.
add_padding(xmax, ymax, xmin, ymin, yorg, xorg) : int []	This function adds padding to the segmented image before extracting it.

Class ConvNN	
ConvNN is the class of the ML model which implements the Convolutional Neural Network. This class holds the hyperparameters of the architecture, uses Trainer class to train its model, and saves the trained model for later use. The predictions of the model are made in this class.	
Attributes	
private String[] classes	
protected Cuda device	
private Boolean print_dataset	
private Boolean print_testset	
private int image_size	
private int freeze_AlexNet_layer	
private torch.AlexNet model	
Methods	
forward(X) : int	This function is the classic forward method of CNN, predicts the class of the given image.
forward_image(image) : int	This function applies transformations before calling the forward method.
load_model()	This function loads the model for the later use by ML_Manager.
initialize_CNN()	This function is the main driver function of this class, it is a procedure of creating transformations, datasets, and calling training function.
enter_log(text : String)	This general function enters log to the log file.
assign_to_cuda_device(device : Cuda)	This function assigns each object to the Cuda.
get_init_weight() : NN.Weight	Returns to the initial weights of the CNN for easy convergence.

save_current_model()	Saves the current model.
----------------------	--------------------------

Class Trainer_CNN	
This class implements the training procedure of the model. The sole reason for this functionality to be implemented as a separate class is ease of use.	
Attributes	
private int[] training_dataset	
private int[] validation_dataset	
private Compose transform_train	
private Compose transform_val	
private CrossEntropy criterion	
private optim.Adam optimizer	
private int[] losses	
private int[] validation_losses	
private int epochs	
private int batch_size	
private Double learning_rate	
private Double train_validation_split_ratio	
private String dataset_path	
private Boolean print_initial_dataset	
private Boolean plot_loss_and_corrects	

Methods	
train_Adam()	implements the entire training procedure of the model.
get_training_dataset() : ImageFolder	Returns the training set.
get_val_dataset() : ImageFolder	Returns the validation set.

Class Tester_CNN	
This class implements the testing procedure of the model and calculates the evaluation matrices.	
Attributes	
private int[] test_dataset	
private Compose transform_test	
Methods	
test()	Tests the model with the test set and calculates the evaluation matrices.
get_test_dataset() : ImageFolder	Returns the test set.
plot_test_results()	Plots the evaluation metrics.

Class Helper_Functions	
This class is a helper class used by most classes in this subsystem. It implements general purpose functionalities such as printing, plotting, and converting images. Its implementation simplifies the subsystem.	
Methods	
image_convert_to_numpy(tensor) : np.array	Converts PILImage (tensor) to the numpy array.
show_images(images, labels, classes, predictions)	Displays the given image.
plot_loss_and_corrs(epochs, loss, cor, val_loss, val_cor)	Plots the loss and corrects of the training procedure.
label_sample_image(sample_img, box_coors, pol) : PIlImage	Draws rectangular boxes around the labeled pollens.

3.2.2.2 Data Tier

3.2.2.2.1 Database Subsystem

Class AcademicModel	
Contains the class of AcademicModel, which has been used both in client and data level. It is used to acknowledge the data received from the backend as well as sent and acquired from the database.	
Attributes	
private int	academic_id
private String	name
private String	surname
private String	appellation
private String	job
private String	mail
private String	institution
private String	password
private Image	photo
private String	research_gate_link

Class SampleModel

Contains the class of SampleModel, which has been used both in client and data level. It is used to acknowledge the data received from the backend as well as sent and acquired from the database.

Attributes

private int sample_id

private int academic_id

private Image sample_photo

private Date date

private int location_latitude

private int location_longitude

private String analysis_text

private Boolean publication_status

private Boolean anonymous_status

private String research_gate_link

private List<PollenTypeModel,int> pollens

Class PollenTypeModel

Contains the class of PollenTypeModel, which has been used both in client and data level. It is used to acknowledge the data received from the backend as well as sent and acquired from the database.

Attributes

private String pollen_name

private String explanation_text

Class FeedbackModel

Contains the class of FeedbackModel, which has been used both in client and data level. It is used to acknowledge the data received from the backend as well as sent and acquired from the database.

Attributes

private int feedback_id

private String name

private int academic_id

private String email

private String text

private Date date

private String status

Class Databas_Manager	
<p>This class is the main processor of the Database Subsystem. It is used to connect to the database, initialize it and then execute commands for utilizing the database. It uses other Model classes to send and receive data from the database, in which the tables correspond with the model objects.</p>	
Attributes	
private Connection db	
private Cursor cursor	
Methods	
Database_Manager(initialize_database : boolean = false)	Initializes database
connect_database()	Connects to database.
delete_tables()	Drops all tables in the database.
create_tables()	Creates tables in the database with predetermined attributes and table names.
initialize_pollen_types()	Populates the pollen table with tuples.
get_academic_from_id(academic_id : int) : Database Subsystem.AcademicModel	Returns the academic user tuple in the database with the given id.
get_academic_from_email(email : String) : Database Subsystem.AcademicModel	Returns the academic user tuple in the database with the given email.
get_pollen_type(name : String) : Database Subsystem.PollenTypeModel	Returns the pollen with the given name in the database.
get_sample(sample_id : int) : Database Subsystem.SampleModel	Returns the sample tuple in the database with the given id.
get_samples_of_academic(academic_id : int) : Database Subsystem.SampleModel []	Returns all the samples uploaded by the academic user with the given id.
get_samples_of_location(location_latitude : int, location_longitude : int) : Database Subsystem.SampleModel []	Returns all the samples in the database with the given coordinates.
get_all_samples() : Database	Return all the samples in the database.

Subsystem.SampleModel []	
get_total_sample_num() : int	Returns the number of samples in the database
get_feedback_from_id(feedback_id : int) : Database Subsystem.FeedbackModel	Returns the feedback tuple with the given id.
get_feedback_from_email(email : String) : Database Subsystem.FeedbackModel	Returns the feedback tuple with the given email.
add_feedback(feedback : Database Subsystem.FeedbackModel) : boolean	Creates and adds a new feedback tuple into the database. Returns true if successful, otherwise false.
delete_feedback(feedback_id : int) : boolean	Deletes the feedback tuple with the given id. Returns true if successful, otherwise false.
add_academic(academic : Database Subsystem.AcademicModel) : boolean	Creates and adds a new academic tuple into the database. Returns true if successful, otherwise false.
delete_academic(academic_id : int) : boolean	Deletes the academic tuple with the given id. Returns true if successful, otherwise false.
delete_academic_from_email(email : String) : boolean	Deletes the academic tuple with the given email. Returns true if successful, otherwise false.
add_sample(sample : Database Subsystem.SampleModel) : boolean	Creates and adds a new sample tuple into the database. Returns true if successful, otherwise false.
delete_sample(sample_id : int) : boolean	Deletes the sample tuple with the given id. Returns true if successful, otherwise false.
add_pollen_type(pollenType : Database Subsystem.PollenTypeModel) : boolean	Creates and adds a new pollen type tuple into the database. Returns true if successful, otherwise false.
delete_pollen_type(pollen_name : String) : boolean	Deletes the pollen type tuple with the given name. Returns true if successful, otherwise false.
update_academic(academic : Database Subsystem.AcademicModel) : boolean	Updates academic tuple with the given parameters. Returns true if successful, otherwise false.
update_pollen_type_description(pollen : Database Subsystem.PollenTypeModel) :	Updates pollen type tuple with the given parameters. Returns true if successful,

boolean	otherwise false.
print_academic_table()	Prints the academic table and its tuples
print_sample_table()	Prints the sample table and its tuples
print_pollen_type_table()	Prints the pollen type table and its tuples
print_sample_has_pollen_table()	Prints the sample_has_pollen table and its tuples
print_feedback_table()	Prints the feedback table and its tuples

3.3 APIs and Tools Used

3.3.1 Django REST API Framework

REST API is an architectural style for an application program interface that uses HTTP requests to access and use data and Django allows the implementation and utilization of REST API by allowing connection to a database and frontend.

3.3.2 Google Firebase Storage Service

In order to achieve image transfer between frontend, backend and machine learning model, Google Firebase Storage Service was utilized. It also allowed the storage of the desired analysis images as well as processed images.

3.3.3 PyTorch

PyTorch is an open source machine learning framework that accelerates the path from research prototyping to production deployment, and it was used to implement the main machine learning model of the system.

3.3.4 ScKit Learn

For the machine learning model to work properly, a considerable image preprocessing was necessary. SciKit Learn is a simple and efficient tools for predictive data analysis which was the most suitable for this task.

3.3.5 Cv2

In the image processing as well as in the machine learning processes, simple and complex operations were needed. OpenCV provides a real-time optimized Computer Vision library, tools, and hardware, and allow the aforementioned tasks to be achieved.

3.3.6 Matplotlib

Especially in the testing stage of the machine learning implementation, data need to be represented and visualized for correct assessment and improvement of the machine learning model. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python, and was utilized as a great tool for testing the machine learning model and its performance.

3.3.7 MySQL

PolliVidis needs to store the uploaded samples, their analysis, user information, and pollen details. In order to achieve this a database system is required. MySQL enables users to meet the database challenges of next generation web, cloud, and communications services with great scalability and uptime. Furthermore, most of the members were familiar with MySQL, and hence, it was the choice for the database system of the project.

3.3.8 Dijkstra Server

In order for the data in the database be synchronous for each user, an online server was required. Although the database system of choice, MySQL, could be run on a local machine, a synchronous server which could assist the projects development and testing

was needed. For this purpose, it was decided to build the MySQL database on the Dijkstra Server in Bilkent, which allowed the data be secure in the university environment, synchronous and connected with all machines, while also being reliable server.

3.3.9 React

PolliVidis uses the open source Javascript library React for the front-end part. React is known as the most popular web framework due to its highly flexible structure, and has a wide range of documentation and sources for developers to use. With the help of React and some of its packages, all of the components that require user interactions are developed.

3.3.10 Google Maps API

One of the core functionalities of PolliVidis is to show pollen sample upload locations. This requires great precision on the location, ease of use and familiarity for the users, and automatic local mapping. One of the prominent API's for this task is Google Maps API, which is familiar with most of the users while being easy to use for them. In the implementation and development stage, Google Maps API was also simple to integrate with the overall project. Hence, it was chosen and utilized strongly.

4. Development and Implementation Details

4.1 Image Processing Implementation

PolliVidis requires some procedure of image processing in two different but closely related areas; raw dataset has to be processed to obtain single pollen images can be fed into ML model, and user supplied image to be analyzed has to be processed before forwarding into ML model. The main necessity of this image processing procedure comes from the fact that neither the raw dataset images nor the user supplied images consist of single pollen images. However, ML model can only process single pollen images to train itself or classify the given image. To supply ML model with single pollen images, some image processing procedure has to be applied on these raw dataset images and user supplied images.

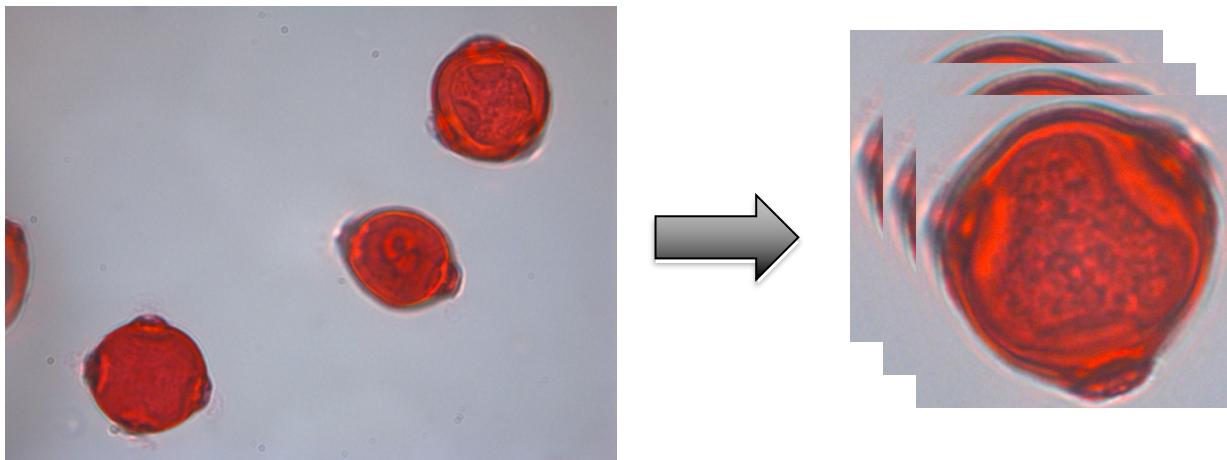


Figure 5: Image Extraction Procedure

This procedure will detect and extract pollen images from the supplied image. For example; when the user supplies PolliVidis a pollen image including 3 Betula pollens, this procedure will detect that there are three pollens in this image and extract each pollen creating its own image. Then, the procedure will forward each extracted image to the ML model to learn its type (classification). ML model will tell that the supplied images are all Betula. Finally, this procedure will label each pollen in the original supplied image with its classification.

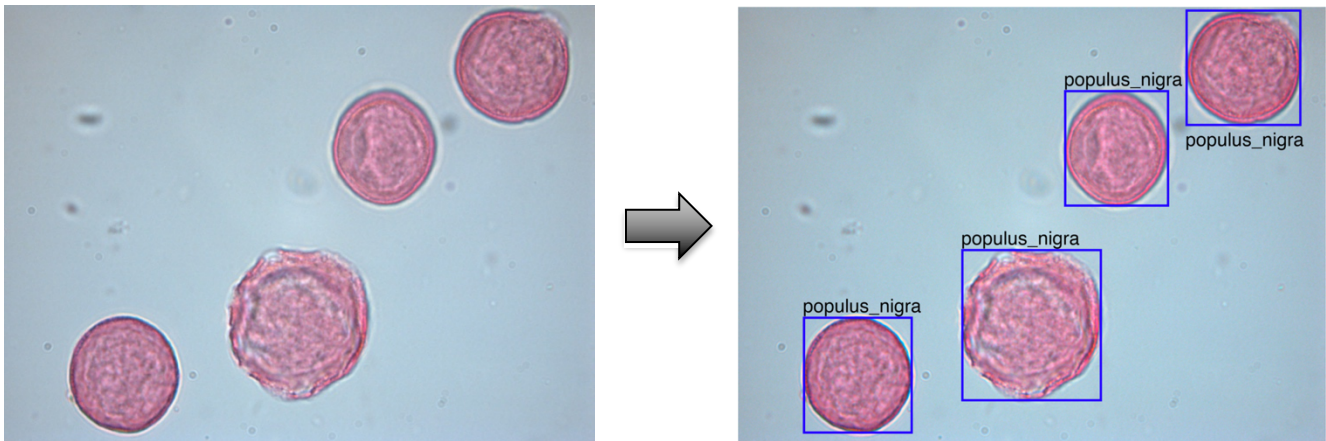


Figure 6: Classification Outcome

Implementation of such procedure came out to be much harder than we initially participated in. Its implementation took as long as the entire ML model implementation. General procedure can be divided into subroutines as follows;

1. Folder Iteration Procedure
2. Load Image
3. Transform into GrayScale
4. Obtain Otsu Threshold
5. Morphology Sequence
6. Label Image according to Otsu Threshold to obtain Regions
7. Extract Single Pollens from the Original Image using Label Coordinates
8. Forward to the Model (details in ML Implementation Section)
9. Label Original Image with Classification Information

In the first step, we have implemented a folder iteration algorithm that directs each image of each species in the dataset to the image processing procedure. This step directs a single image in the case of a user supplied image. Its implementation was pretty straightforward.

In the second step, the given image is loaded to the memory as a Python array (NumPy array) to ease the morphology sequence procedure.

In the third step, the loaded image is transformed into a GrayScale image for Otsu Thresholding.

In the fourth step, Otsu Threshold of the gray scaled image is obtained (Python skimage library Otsu Function).

In the fifth step, a series of Morphology Sequences are applied to the thresholded image to finetune the labeled areas. This step is the main functionality of Image Processing Procedure and the hardest step to implement. We have implemented two different but mostly equivalent Morphology Sequences Procedures, one for automatic procedure for general users, and one for manual morphology procedure for more sophisticated users. The alternative morphology procedures are opening, closing, erosion, dilation, area opening, and are closing. We have realized that multi erosion followed by multi dilation and finally area closing is well-suited for most pollen images. Thus, we have automated this procedure and expect users to supply only one parameter, number of multi erosion and dilations to apply. Users can find the correct parameter of his/her pollen image with trial and error since the range is pretty small going 0 to 40 by 5. In the manual mode, users can apply all morphology procedures in disered order by specifying the whole sequence. For example, 'E10-D10-AC100000' means 10 erosion followed by 10 dilation and followed by 100000 area closing.

In the sixth step, we label the regions of the thresholded image with skimage library. It shows distinct regions in the image.

In the seventh step, each single pollen image is extracted from the original image according to the coordinates of labeling in the previous step. Thus, we obtain the single pollen images of the original image.

In the eighth step, we forward each pollen image to the ML model to classify it. Details of the ML model will be supplied in the next section.

In the ninth and the last step, procedure labels the original image with the classification information supplied by the ML model.

The Procedure

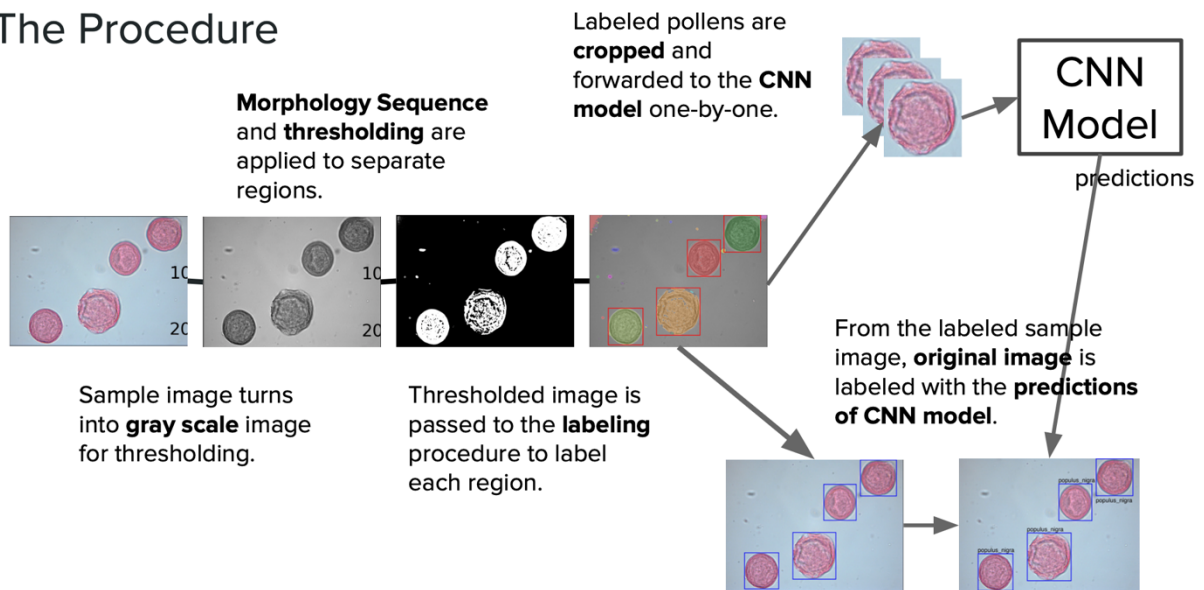


Figure 7: Image Processing Procedure Detailed

4.2 Machine Learning Implementation

After the preparation of the dataset, we created from scratch and the Image Processing Procedure implementation; we have implemented the core functionality of PolliVidis, machine learning; more specifically Convolutional Neural Network for multiclass image classification.

In the first step, we have implemented different CNN architectures with pretrained parameters to make use of transfer learning. We have implemented AlexNet, ResNet50, VGG16, and VGG19 architecture. Moreover, we have frozen the models half way through and changed their final layer to match it with our number of classes which is 23 currently. It should be noted that we are starting to write an article on the machine learning part of this project which will use different and fine-tuned architectures.

After loading the model, we have implemented the transformations we will apply to the training and validation datasets to make use of Data Augmentation. We have decided which data augmentation techniques we will use.

After that, we have implemented the dataset loading and partition procedures which loads the dataset into memory and partition it into training, validation, and test datasets.

Then, we have decided and implemented the loss function (Cross Entropy Loss), optimizer (Adam Optimizer), and Learning Rate Scheduler. We have passed these to the training function.

Before implementing the training function, we have implemented the load and save models and checkpoints. Checkpoints allow us to put literal checkpoints into the training procedure and continue it within different runs. It helps a lot since Google Colab disconnects us after 2 hours of idleness.

Then, we have implemented the training procedure which is the barebone of the entire machine learning implementation. We have used Batch Gradient Descent with Adam Optimizer and Cross Entropy Loss. It means that at each epoch, the procedure divides the training dataset into batches and evaluates the model at the end of each batch. After each batch, the loss function is recalculated and passed to the optimizer which will minimize it. Then, Adam optimizer recalculates the unfreezed weights of the model. Finally, the learning rate scheduler takes a step and recalculates the current learning rate Adam optimizer should use. Algorithm calculates the loss and accuracy at the end of each batch and shows it. After that, the validation process takes over. Validation loss and accuracy is calculated in here for us to decide on the hyperparameters of the model which can be listed as follows:

1. Learning Rate
2. Epoch
3. Batch Size
4. Step Size of Learning Rate Scheduler
5. Gamma Value of Learning Rate Scheduler
6. Training-Validation Split Ratio

After the training procedure, the model gets into the testing procedure which will decide the accuracy, F1, precision, and recall scores of the trained and fine-tuned model.

After the testing, several plots are calculated and shown such as; loss, accuracy, confusion matrix, and 20 pollen images tested.

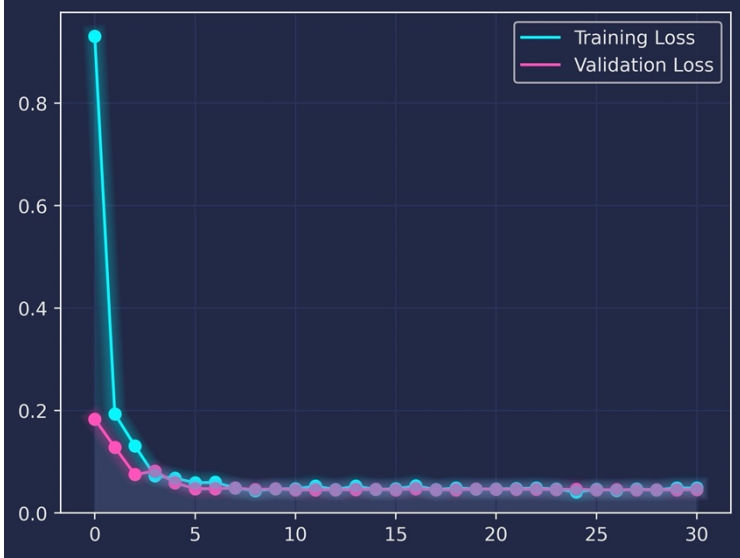


Figure 10: CNN Loss Graph

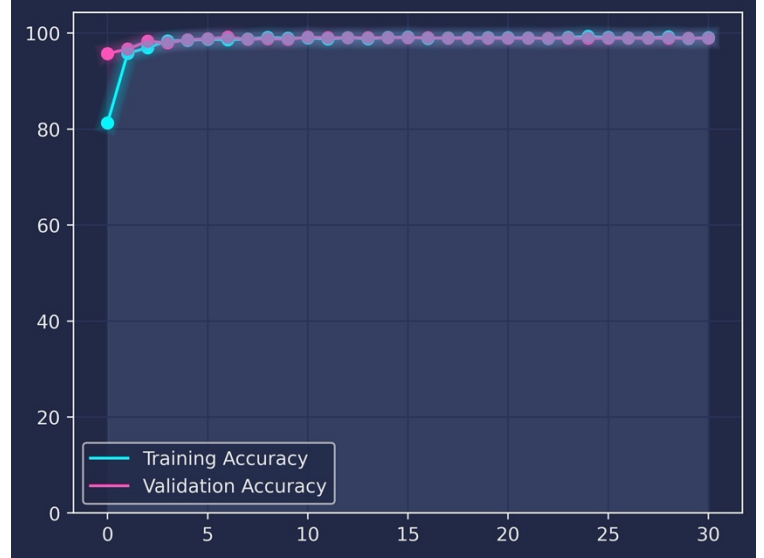


Figure 9: CNN Accuracy graph

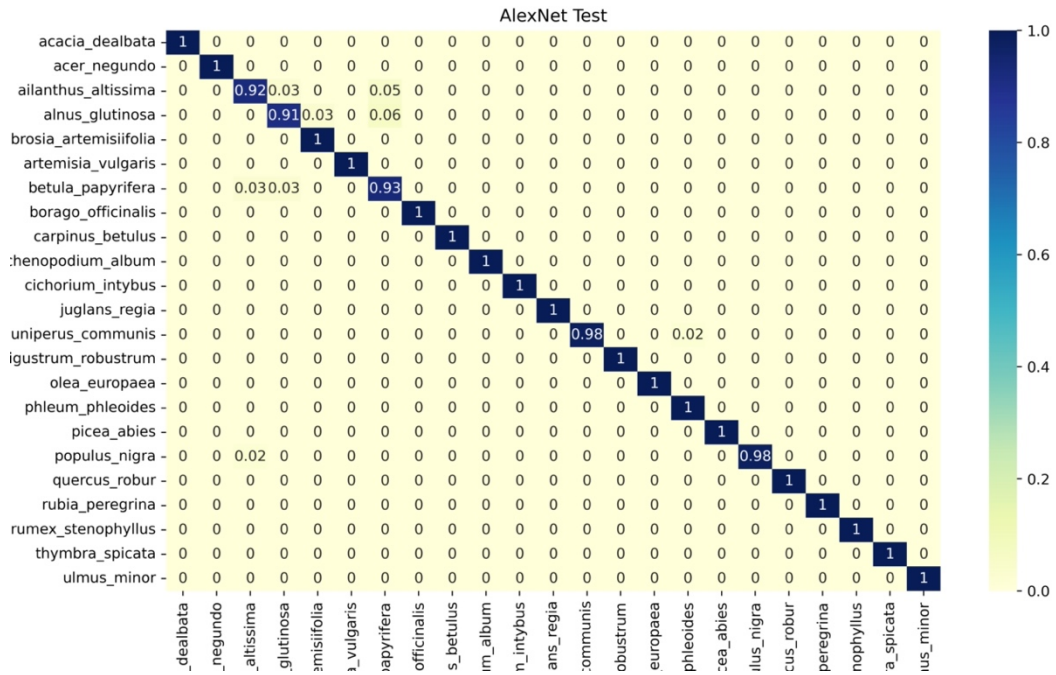


Figure 8: CNN Confusion Matrix

Accuracy	98.72%
Precision	98.81%
Recall	98.8%
F1 [Macro]	98.8%

Figure 12: CNN Performance Metrics

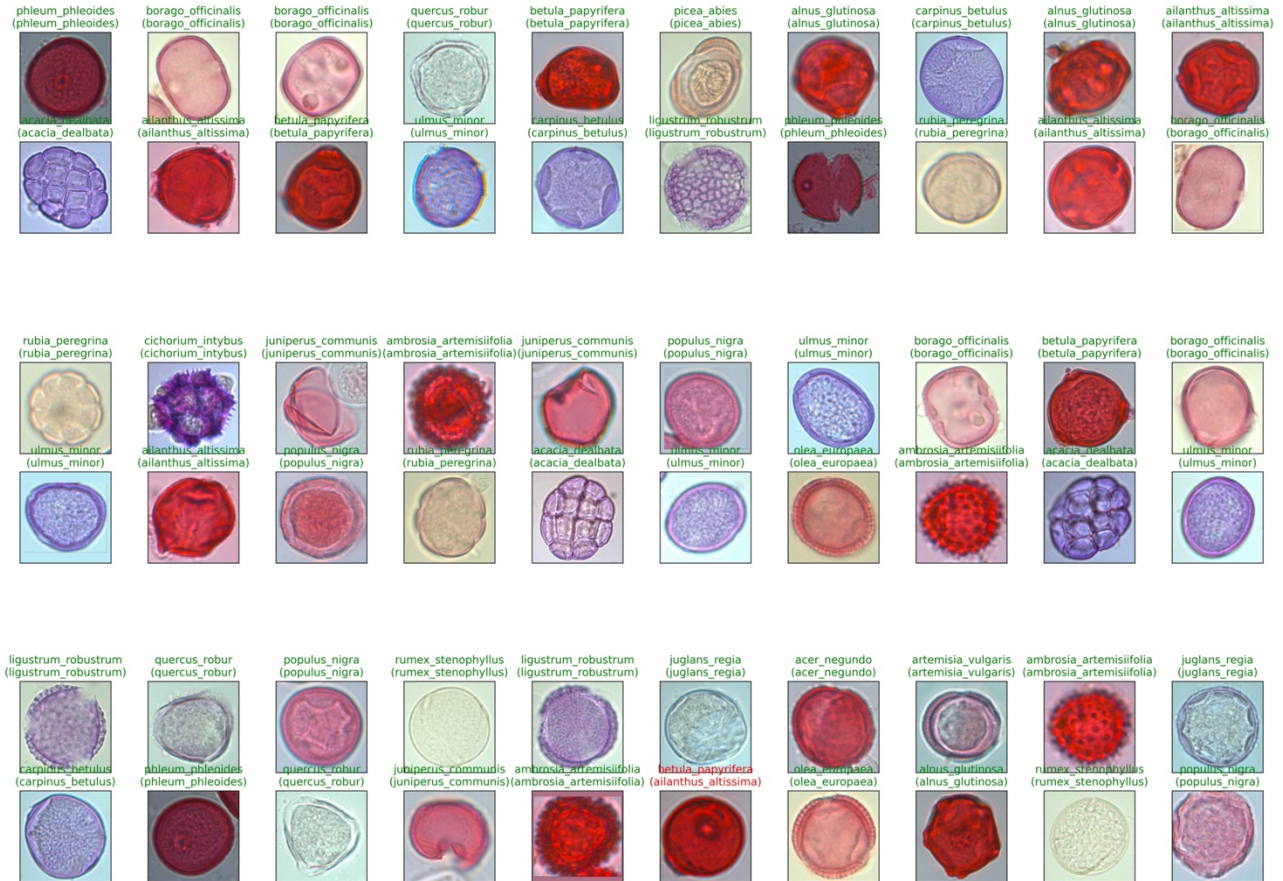


Figure 11: CNN Test Examples (red ones are wrongly predicted)

4.3 Backend and Database Implementation

The backend portion of PolliVidis can be divided into 2 parts: the request handler and the database. PolliVidis utilizes Django REST API Framework in its backend and MySQL as its database implementation.

Firstly, in order to create the request handler, the required requests are discussed, planned and defined. In these discussions, content, purpose and the utilization of these requests are established. Afterwards, all these requests are divided into two as admin calls or API calls. The API calls were established as the requests that would be used in the end product by the user meanwhile the admin calls were created for testing, debugging, verification, and management purposes.

Furthermore, Django model classes were created which all corresponded to a table in the database in order to properly acquire the results from the calls that would be made to database. Additionally, the serializers of each model is implemented. These serializers allowed the managed, created or acquired objects to be sent as a JSON type data as a HTTP response to frontend. Hence, especially in get requests, the backend could provide an HTTP response containing complex objects rather than data of primitive type to frontend.

Simultaneously, the SQL statements for the database are created. These sql statements are implemented as callable functions of the DatabaseManager. Additionally, these functions are created as general and context free as possible. In other words, they are created as general get, update, delete and insert methods for particular tables. Therefore, this allowed these functions to be reusable and callable by different requests and requirements of the frontend as well as the backend.

After the functions that execute the SQL statements for tables are finished, and along with the progress of the frontend, logical request handling functions for the planned request were started to be implemented. These functions would correspond to a request from the frontend, and then would execute a function from DatabaseManager. Furthermore, it would return a proper HTTP response for the frontend such as an error response if there is a bad request, the request data is unsuitable, or a problem occurred in the database. Conversely, especially if the request was a get request, the response data would be prepared in the form of Django models and serializers if the request is handled successfully. Additional operations could be done for the data such as filtering uploaded samples for the pollen map with properly given specifications.

For the most part, the backend implementation was run along with the progress of the frontend. This was caused by the fact that a backend function was useless without a proper frontend counterpart and furthermore, content, details and requirements of the requests could change during the development and implementation stage, and backend would need to react as well as adapt to the changes by aligning its progress with the frontend. This also allowed the connection testing and request handling verification to be complete simultaneously.

A major issue of the Django was the problem of sending an image from backend to frontend. Although process of storing the image in the MySQL database was successful, by storing it as binary data, and the process of retrieving the said image from database to the request handler, the image could not be returned as an HTTP response since it is converted to a JSON data, which could not store an image or its binary data.

After further investigation and research, this image issue was found to be a common problem in Django projects. Nevertheless, the transfer of image from frontend to backend was necessary in PolliVidis in order to direct the pollen image to the machine learning model, which itself would identify pollens and recognize them. During this process, a processed image would be created which needed to be transferred from backend and frontend; this part of the process was not achievable by Django alone.

Hence, it was decided to use Google Firebase systems file management service. Instead of transferring the image from frontend to backend and vice versa, the image would be uploaded to the Firebase file storage space. Next, the backend would use this download this image and use it in its process and afterwards, the newly generated image would be uploaded to the same storage space which then could be accessed by the frontend. Since each analysis could be identified by a sample id, the images could be uploaded and accessed by the same sample id automatically. Although this meant that the image would be take a longer route between frontend and backend, that consequently increase the analysis time of the whole system, using Firebase solved the major problem of image transfer.

Finally, the model created in the machine learning implementation would be connected to the backend via the MLManager class, the backend was able to handle each frontend request including taking an image, passing it to the machine learning model, and then returning the result of the identification process; or simply, allowed the analysis of the pollen images.

In the end, several bug fixes and performance improvements were applied and the backend implementation was successfully completed.

4.4 UI Implementation

PolliVidis is a web application and it is created using React.js as the main framework. First the design drafts of the pages are prepared. After the first designs, the pages are implemented one by one. At the beginning of the project, we created a priority list for the web application functionalities and the pages are implemented regarding that order. The list can be seen below:

Priority List

- 1) Analyze Sample
- 2) Analysis Report
- 3) Pollen Map
- 4) Previous Analyses
- 5) Sign Up, Login, Logout
- 6) Profile
- 7) Feedback
- 8) About Us, User Manual

The main user scenario of PolliVidis is as follows. A user opens the web application and goes into the Analyze Sample page to upload an image. Image is sent to the backend and also stored in Firebase. The machine learning analysis is done in the backend and the Analysis Report page displays the response. In addition, a user can view pollen analyses done by Academic users in the Pollen Map page. Furthermore, if an Academic login is done, the user's analyses are added to the Pollen Map. Also, Academic users have profile pages and can see their previous analyses. Moreover, every user can view the About Us page and User Manual, also can send feedback.

In order to send and get information to and from backend Axios is used. "Axios, which is a popular library, is mainly used to send asynchronous HTTP requests to REST endpoints" [15].

In addition, every page is a React function component and is composed of other smaller components as we followed a bottom-up approach. Thanks to this approach, we could use the same components for different pages. We manage the communication between parent and child components with React's callback function. For the built in components such as buttons we used Material UI, a famous React component library. Material UI components have built in CSS but they can be customized and we customized them the way they suit our purposes.

Other than Material UI library, for the Map component, we used Google Maps API built for JavaScript. Google Maps API lets users choose their location from the displayed map. Furthermore, with the Geolocation API and “Get My Location” button we give users freedom to choose their current location without marking their location from the map. For filtering purposes, to filter the species of pollen that is going to be displayed, we decided to use Material UI’s Switch component. With the npm library Date-Picker, we let users decide the time interval of the pollen analyzes to be displayed on the map. For the session-based user registration system, we used the sessionStorage functionality of React to keep the user information stored from the moment the user is logged in until the browser tab is closed. To manage the navigation between pages, we used React Router DOM package. For external packages that described above, we used the npm packaging system to import packages into our application.

5. Testing Details

5.1 Testing of Image Processing Procedure

Testing of Image Processing Procedure is straightforward as the final product of the procedure is available as soon as the code runs. Thus, testing of how well the Image Processing Procedure performs on the dataset is visualized in the development process. Image Processing Procedure is said to be well performed when it manages to extract more than %95 of the entire raw dataset properly. Its performance on the user supplied image relies on the morphology sequence user supplies.

5.2 Testing of Machine Learning Procedure

Testing of Machine Learning Procedure is maintained by log files. The procedure created a plain text file on Google Drive as a log file at the beginning of its design. After that, the procedure logged everything it did on the dataset and model with proper header including all hyperparameters and the timestamp.

Hyperparameter testing of the machine learning model is maintained by the log files as well as the output graphs of the procedure such as loss, accuracy, confusion matrix, and last but not least, validation set.

After the testing of the final ML model, we obtain more than %98 accuracy. Moreover, all other scores such as F1 are higher than %98.

5.3 Testing of Backend and Frontend

Testing of the website is mainly manual. Expected behaviors of database, user interface and api calls were mostly checked by using frontend screen and console and also Django terminal.

6. Maintenance Plan and Details

Image Processing and Machine Learning Procedures are backboxes that do not require maintenance in time except the library versions these procedures use. However, they are highly open to improvements since the success rate of the Image Processing Procedure can always be improved as well as the accuracy and the number of the pollen types in Machine Learning Procedure. Such improvements will be considered when we write the article of this project.

Web application maintenance includes database maintenance, frontend framework library's version updates, Google Maps Api and Google Firebase account and storage checks, browser compatibility checks.

7. Other Project Elements

7.1 Consideration of Various Factors in Engineering Design

- **Public Health**

PolliVidis aims to increase the life quality of people that have pollen allergies by showing the allergenic pollen information at his/her area so that users can take precautions. Since it is directly related to public health, we have to have some standards which ensures to protect public health, at least not put it in danger. That is the main reason PolliVidis allows only academics to upload their analyses to its database and uses these analyses to give predictions. The system ensures the reliability of each academic. In any case, the predictions of PolliVidis should be taken as advice.

- **Public Safety**

Only concern with public safety can be the privacy of academics since PolliVidis stores the communication information of each academic. The system gives privacy options to the academic about who can view their information. If an academic wants to keep his/her communication information private, s/he can do so.

- **Public Welfare**

PolliVidis wants to increase the user's quality of life by showing the allergenic pollen information at his/her area so that the user can learn the pollen levels and take precautions. Moreover, academics can share sample analyses quickly which may increase collaboration in academic research.

- **Global Factors**

PolliVidis will be designed in a way that it can be opened to the world easily with some adaptations and regulations since different countries have different allergenic pollen types and a single model cannot be trained to classify all of them. Moreover, analysis sharing between academics would help any academic worldwide.

- **Cultural Factors**

Since PolliVidis is related mostly to academic research and scientific information sharing, effects of cultural factors are minimal.

- **Social Factors**

Sample and analysis upload to the pollen map will be allowed for academic staff only. Thus, uploading random images to the map will not be possible to prevent potential abuse. The feedback page allows users to send feedback about system problems or specific samples. Finally, PolliVidis does not support direct messaging which reduces the concerns about online abuse.

- **Environmental Factors**

It is not expected that the usage of PolliVidis website will cause any environmental problems apart from the energy the servers and database will use and this can be seen as the bare minimum damage. Any type of printed or physical material will not be needed. Moreover, the generated dataset will be published and eliminate the need of collecting the same samples again and again for future research, which saves human and electrical energy.

- **Economic Factors**

All the services PolliVidis uses are free and PolliVidis does not charge its users for the map usage or sample analysis.

	Effect Level (out of 10)	Effect
Public Health	8	Showing allergenic pollen density on a map so that people can take precautions is one of the main goals of the system
Public Safety	3	System will have privacy options so academics can keep their information private if they would like to.
Public Welfare	8	System aims to raise the public welfare with the Pollen Map feature so that users can take precautions but also as letting people share their pollen samples quickly in order to increase collaboration in academic research.
Global Factors	3	PolliVidis will be designed in such a way that it can be a world-wide application easily, also sharing analyses with the world would help any academics worldwide.
Cultural Factors	2	The effect of cultural factors are negligibly low.
Social Factors	4	PolliVidis does not have a direct messaging service and the system will not allow people to upload random images so that online abuse will be prevented.

Environmental Factors	3	Publishing the dataset will eliminate the need of collecting the same samples, hence consumption of electric energy will be minimized.
Economic Factors	2	The effect of economic factors is negligibly low.

7.2 Ethics and Professional Responsibilities

PolliVidis aims to provide allergenic pollen classification with the best accuracy. Furthermore, it aims to provide information about the allergenic pollen density to its users therefore the data it provides to its users should not be biased. For registered users, it should ensure the privacy of their personal information and fit the KVKK in Turkey. It should not share or process any personal data without the user's consent [16]. For further versions, privacy policies might be extended to fit into global regulations. For unregistered users, it should not save or require any usage of personal information.

7.3 Judgements and Impacts to Various Contexts

- **Developing PolliVidis as a website:**

As developers, we decided to produce PolliVidis as a website; we could also choose a mobile application or desktop application. However, our goal is to help not only the palynologists in Ankara University but to help others in different parts of the world in future improvements, therefore; we decided that a website would fit the best. Furthermore, mobile applications would not be useful because microscopes are usually connected with computers and it is easier for scientists to upload samples from computers.

Judgment	Level of Impact	Description
Global Context	9	Publishing it as a website enables scientists from other parts of the world to use PolliVids.
Economic Context	4	PolliVidis is a free website and does not require any installation fee.
Environmental Context	6	Pollen map becomes available for everyone in Turkey.
Social Context	8	PolliVidis website enables scientists to see analyses made by others.

- **Choosing the English as the language of the website:**

PolliVidis is an English website since nearly every university in Turkey requires a level of English and it is the most commonly used language in the world.

Judgment	Level of Impact	Description
Global Context	9	English is the most commonly used language.
Economic Context	4	Making it English might increase the number of users.
Environmental Context	0	None
Social Context	0	Making it English might increase the number of users.

- **Disabling non-academic users to upload analyses on pollen map:**

Information published in pollen map is naive information about pollen densities and any misinformation might be dangerous for allergic people. Therefore; in order to upload an analysis in pollen map, users should be registered academics.

Judgment	Level of Impact	Description
Global Context	4	It is important for public health.
Economic Context	0	None
Environmental Context	0	None
Social Context	8	It is important for public health.

- **Publishing a Pollen Map that includes only Turkey:**

For the first version, PolliVidis only has a pollen map that includes Turkey. However, pollen map is crucial for the allergenic people and making it global in the first version might bring safety issues on correctness of analyses.

Judgment	Level of Impact	Description
Global Context	0	None.
Economic Context	0	None.
Environmental Context	6	Pollen map becomes available for everyone in Turkey.
Social Context	3	People cannot see pollen analyses in different parts of the world.

7.4 Teamwork Details

7.4.1 Ömer Ünlüsoy

- Helped creating the dataset in Ankara University (photographed >2000 pollens).
- Designed and implemented the Image Processing Procedure for the dataset and user supplied images.
- Designed and implemented the Machine Learning Procedure.

7.4.2 Gamze Güliter

- Helped creating the dataset in Ankara University (photographed >2000 pollens).
- Helped Ömer on ML implementation

7.4.3 İrem Tekin

- Found the project topic and provided connections with Ankara University and Zonguldak Bülent Ecevit University Palynology departments.
- Helped creating the dataset in Ankara University (photographed >2000 pollens).
- Designed and implemented UI of the web application using React.
- Helped with backend-frontend communication implementation.
- Helped to solve some backend problems.

7.4.4 Ece Ünal

- Helped creating the dataset in Ankara University (photographed >2000 pollens).
- Worked with İrem on the UI implementation
- Worked with Ada and İrem to manage the communication between backend and the frontend

7.4.5 Umut Ada Yürüten

- Planned and implemented the backend
- Worked on backend - frontend communication
- Worked on backend - database communication
- Connected backend with the machine learning model
- Improved the database and contributed by creating SQL statements
- Helped solving several UI problems
- Suggested and implemented Firebase for image transfer

7.5 New Knowledge Acquired and Applied

In order to understand the nature of the pollen classification problem, we needed to learn its motivation. Therefore, we read a lot of academic papers about pollen and pollen classification applications.

There is only one member in our team who is experienced with Deep Learning, who is Ömer. Therefore, he leads us in the learning process. We watched YouTube tutorials and UdeMy Courses on Convolutional Neural Networks. For building the website, we used YouTube tutorials and hands-on experience. For backend and database application, we again, combined our previous knowledge and followed tutorials.

8. Conclusion and Future Work

In conclusion, as the PolliVidis team, we managed to implement a machine learning model with more than %98 accuracy which is higher than previously published pollen classification papers. Therefore, we plan to publish an article about the computer vision aspect of our project. Our goal is to write our paper in the summer and send it to the journals as soon as possible.

We also spent a lot of time collecting our own data. Furthermore, we took more than 5000+ pollen pictures using a microscope, each of them containing multiple pollen samples. Using our image processing algorithm, our data size increases up to 6000 samples. Currently, there are pollen datasets such as PalDat, however, the size of these datasets are either small or they do not contain samples with the same format (taken with different microscopes, in different labs). Therefore, we plan to publish our data set both in GitHub and as a paper if possible. Our data is collected by using clean samples, and the same format. We believe that it would be really useful for the future biological and palynological research.

We also created a pollen map on our website which shows the pollen densities in Turkey and believe that it will be really useful for the people with seasonal pollen allergies.

There are some companies such as Arçelik, who currently work on pollen classification applications on their air-cleaning products. One of our future goals is to collaborate with one of these companies on their future pollen classification applications.

9. User Manual and Installation

9.1 About Installation

Pollividis does not require any installation since it is a web application. Anyone with an internet connection and website address can use Pollividis.

9.2 Pollen Map

This is the page when users enter the Pollividis webpage. It shows the pollen map which contains pollen analysis of specific locations. There are mini red microscope icons on the pollen map. These icons work as a button and if the user pushes one of them, he/she can see the pollen analysis made by academics on that location.

There is a red button “Filter” on the left side of the main page, by using this button, any user can see the location of a special pollen. For example, if they filter the name “Populus nigra”, they will only see the locations of Populus nigra pollens in the map.

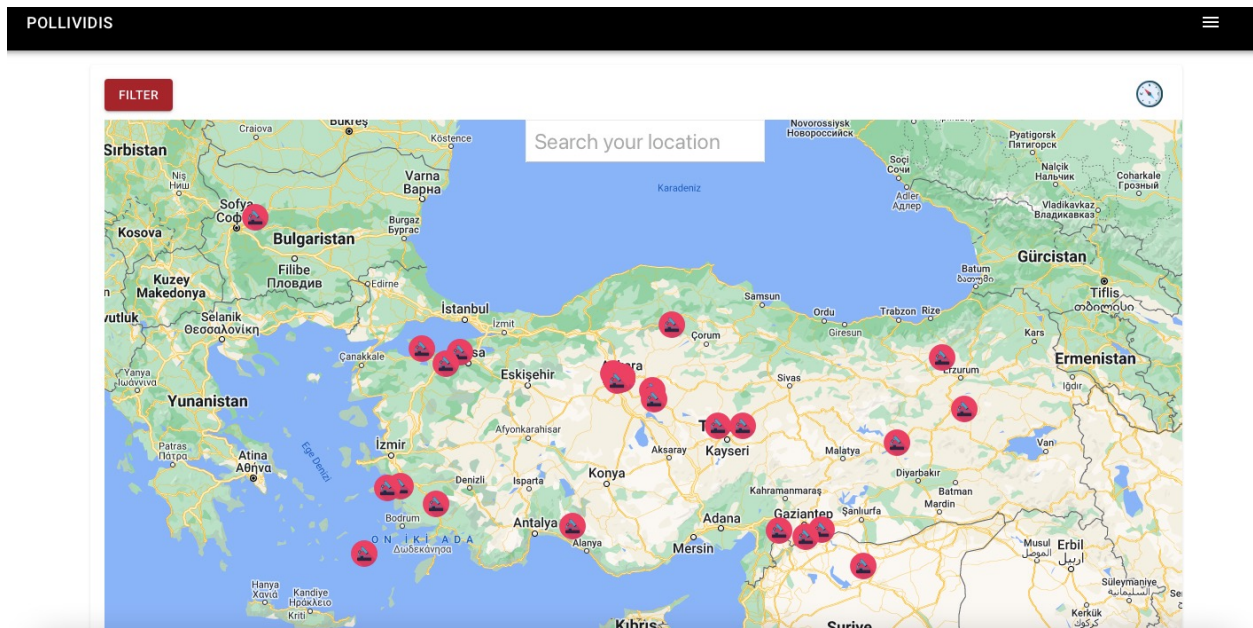


Figure 13: Pollen Map

9.3 Registration (Sign Up)

If a user presses the academic login button in the navigation menu, the login page will be open. There is a “Sign Up” button in that page which will direct the unregistered user to the Registration Page. In the Registration Page, academic users will be signed up to the PolliVidis by their name, appellation, institution, email and a safe password.

Users need to register by using a unique email address, previously used email addresses are not allowed. For the safety issues, all registered users need to be academics, therefore, academicians have to give the name of the institution that they work in order to validate their registration request.

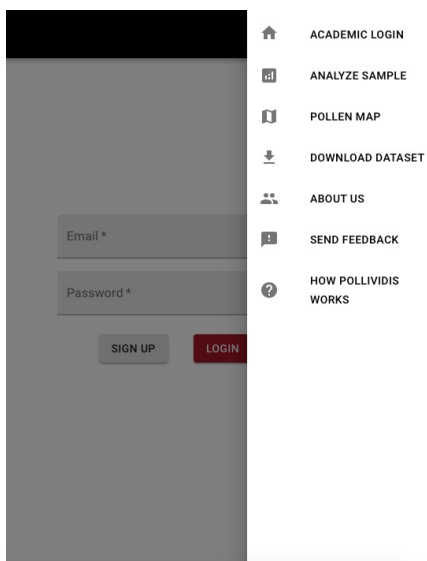


Figure 14: Menu Bar

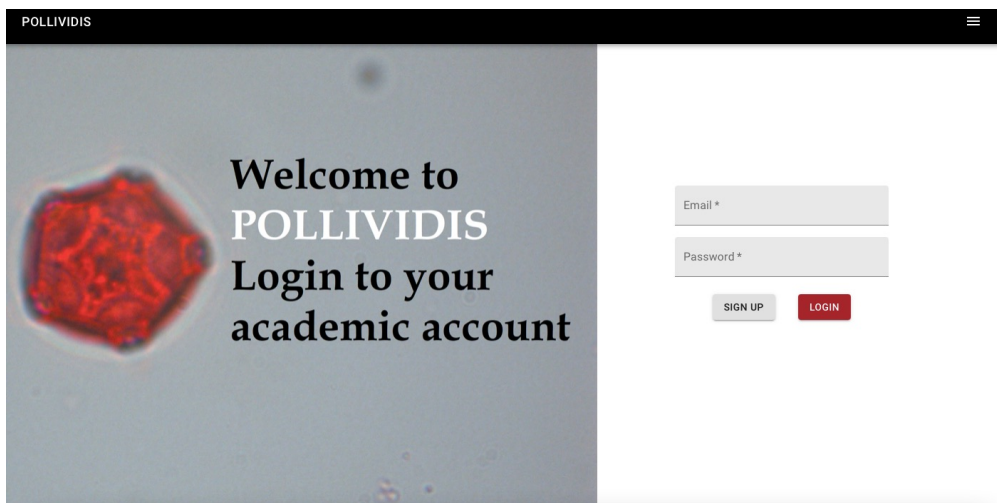


Figure 15: Login

POLLIVIDIS ☰

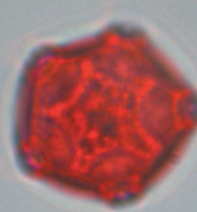
Academic Sign Up

Figure 16: Academic Sign Up

9.4 Academic Login

In order to add analysis to the pollen map, or look back to their previous analysis, academics have to login from the academic login page. Below is the academic login page where they can login by their academic email and password.

POLLIVIDIS ☰



**Welcome to
POLLIVIDIS
Login to your
academic account**

Figure 17: Academic Login

9.5 Navigation Menu without Academic Login

Actions of a non-academic user is limited in PolliVidis. The user can analyze a pollen sample, look at the pollen map but cannot add anything to the pollen map. Below is the left navigation menu in PolliVidis without any login.

Users without an academic login can also send feedback to developers, see informative pages such as “How PolliVidis Works” and “About Us”.

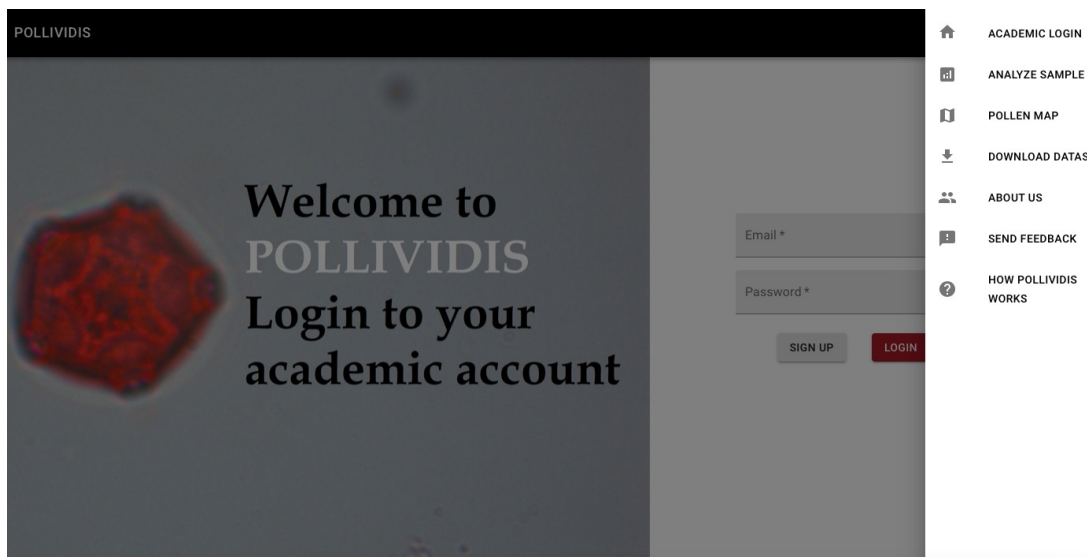


Figure 18: Navigation Menu Bar without Login

9.6 Navigation Menu with Academic Login

Below is the navigation menu after academic login. In that menu, academic users can see their profile from the “Profile” button, see their previous analyses by the “Previous Analyses” button and log out by the “Logout” button.

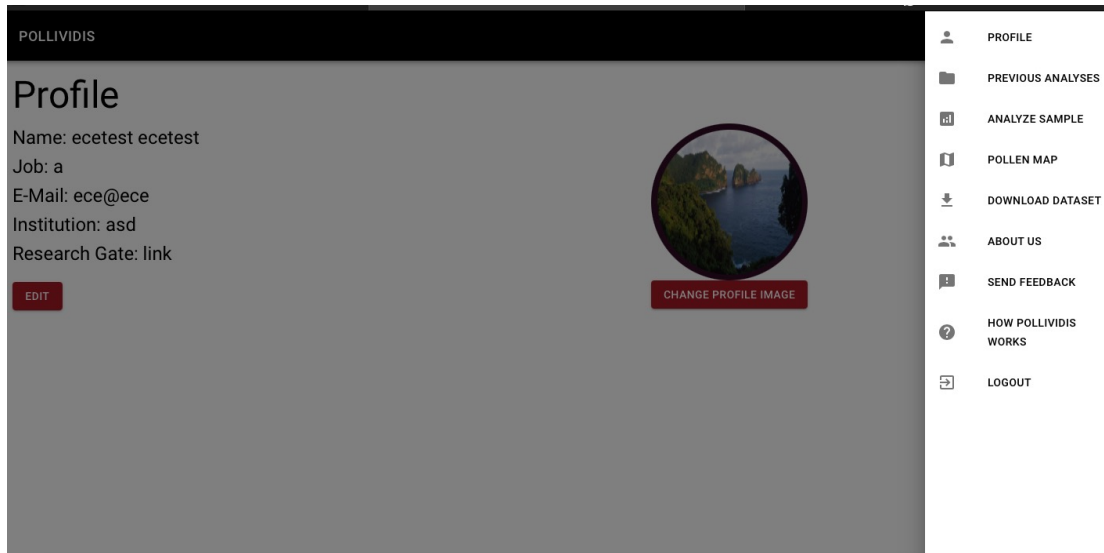


Figure 19: Navigation Menu Bar with Login

9.7 Analyze Sample Page

In order to analyze a sample, users should press the “Analyze Sample” button in the navigation menu. After pressing the button, the Analyze Sample page will be open. In that page, there are option buttons in the upper left corner. By using the “Upload Sample Image” button, users can upload any pollen image from their local device.

There is a Morphology Sequence box with the default value 10. This input is for erosion and dilation settings in pollen image processing, on average using the value ten gives good results, however; users can change it.

There is a “Get My Location” button next to the Morphology Sequence box. This button is used to get the user's location automatically in order to save the analysis in a pollen map. However, users can also give any location by pressing on that location in the pollen map.

After all the required inputs are given, the user needs to press “Analyze” button and wait for machine learning algorithm to analyze the sample.

POLLIVIDIS ☰

Analyze Sample

UPLOAD SAMPLE IMAGE
UPLOAD IMAGE COLLECTION

GET MY LOCATION

ANALYZE

Sample Image Preview

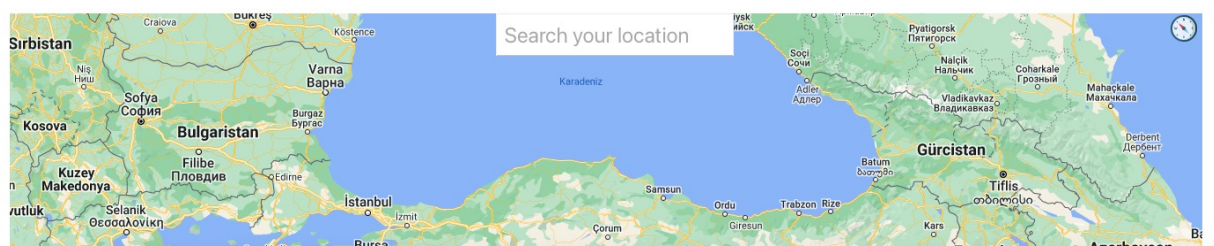


Figure 20: Analyze Sample

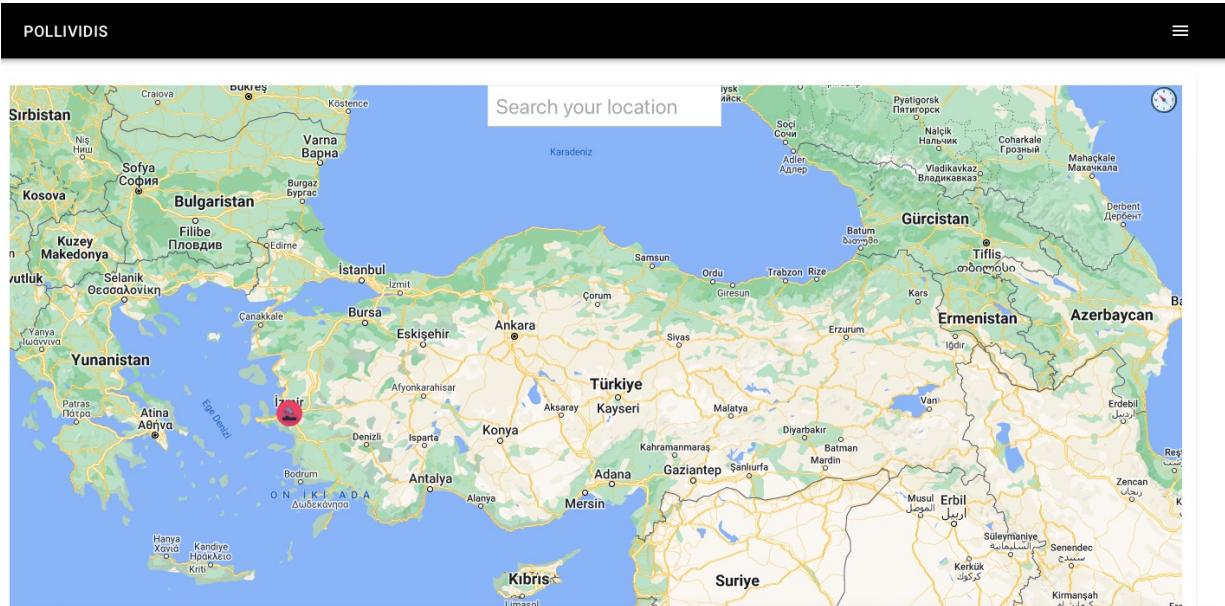


Figure 21: Analyze Sample Location

9.7.1 Upload Image Page

When users press the “Upload Sample Image” button in the Analyze Sample page, the Upload Image Page opens. From this page, users can browse through their local device and upload a pollen image from there in order to analyze it.

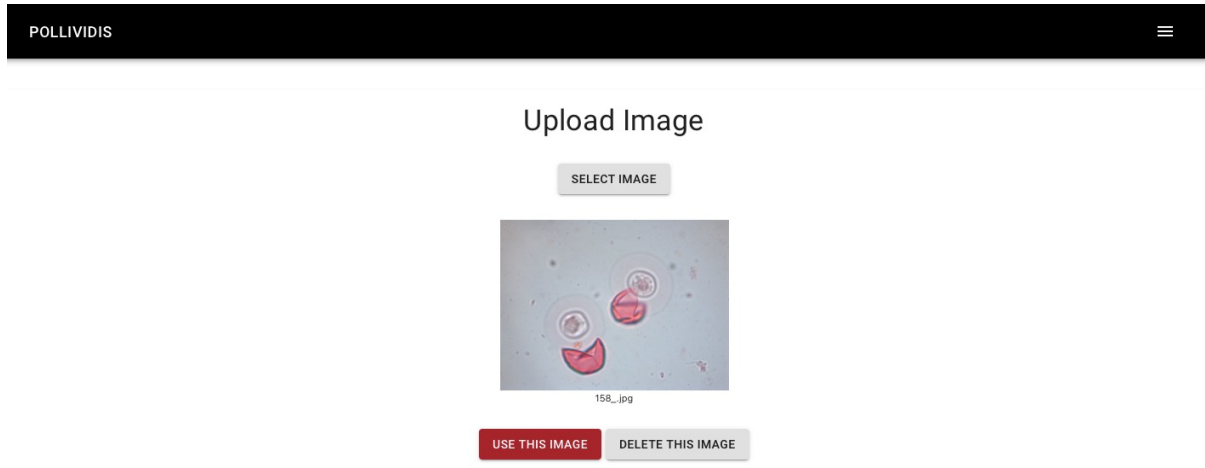


Figure 22: Analyze Sample Upload Image

9.7.2 Analysis Report Page

After the user gives the required information and presses the “Analyze” button, the related Analysis Report page opens. From this page, users can see the classified pollen that the sample contains and the other details such as date and location of analysis.

In the example below, the algorithm classified the pollen samples as *juniperus_communis* pollens, date is May 2 2022, Monday.

Analysis Report

Location: 38.311136772137075-27.163832813039182

Date: Mon May 02 2022 12:11:43 GMT+0300 (+03)

this is example analysis text

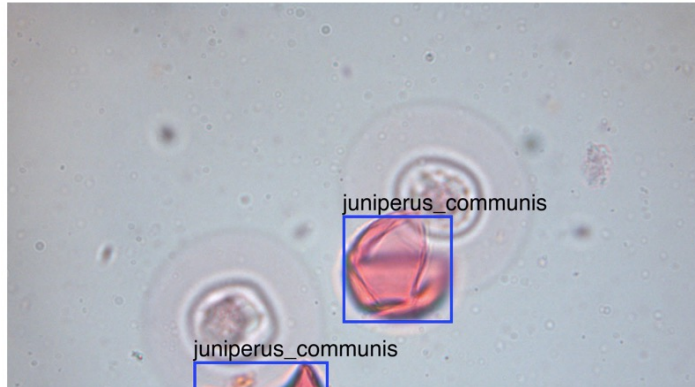


Figure 23: Analysis Report

9.8 Previous Analyses Page

Academic users can also see their previous analyses from the “Previous Analyses” button in the navigation menu. After pressing the button, the Previous Analyses page opens. In this page all the analyses made by the user are listed by their date and location. Users can see the details of them by pressing the “View” buttons on the right side of each analysis. After pressing these buttons small windows which contain information of that analysis will be open.

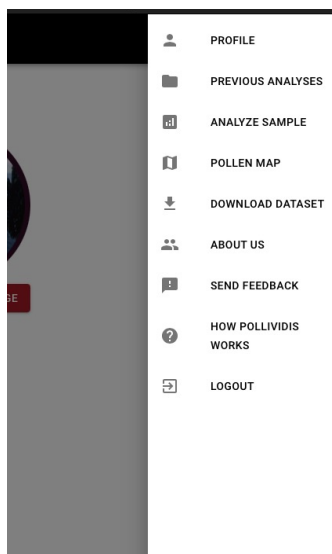


Figure 24: Navigation Menu Bar

POLLIVIDIS

Previous Analyses

- Sun Apr 24 2022 14:50:27 GMT+0300 (GMT+03:00) [VIEW](#)
- Sun Apr 24 2022 19:38:59 GMT+0300 (GMT+03:00) [VIEW](#)
- Sun Apr 24 2022 19:40:58 GMT+0300 (GMT+03:00) [VIEW](#)
- Tue Apr 26 2022 13:16:28 GMT+0300 (GMT+03:00) [VIEW](#)

Location: Address unknown
 Date: Tue Apr 26 2022 13:16:28 GMT+0300 (GMT+03:00)
 Analysis:
 this is example analysis text

Figure 25: Previous Analysis Table

9.9 Profile Page

Academic users can see and edit their profile information from the Profile button in the navigation menu. After pressing the button, the profile page of the user will be open. From the profile page, users can edit their personal information and change their profile picture. In order to upload a new profile picture, users need to press the “Change Profile Picture” button and choose a picture from their local device. In order to edit their personal information, users need to click the edit button on the left side.

- PROFILE
- PREVIOUS ANALYSES
- ANALYZE SAMPLE
- POLLEN MAP
- DOWNLOAD DATASET
- ABOUT US
- SEND FEEDBACK
- HOW POLLIVIDIS WORKS
- LOGOUT

Figure 26: Navigation Menu Bar

Profile

Name: ecetest ecetest

Job: a

E-Mail: ece@ece

Institution: asd

Research Gate: link

EDIT



CHANGE PROFILE IMAGE

Figure 27: Profile

9.10 About Us Page

In order to see credentials of developers, users need to click on the “About Us” button in the navigation menu.

About Us

Pollividis is developed by five Computer Science Students for their Senior Project.

Ömer Ünlüsoy

İrem Tekin

Elif Gamze Güliter

Umut Ada Yürüten

Ece Ünal

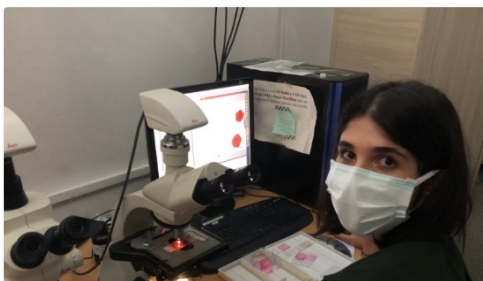


Figure 28: About Us

9.11 Download Dataset Page

In order to download the dataset collected by us, users can press the Download Dataset button in the navigation menu. In that page, there is a link for users to press in order to download the dataset.

Note for the users: Currently, dataset is not available as PolliVidis team plans to write the article of this project and the dataset will be published afterwards. For more information, contact any team member.

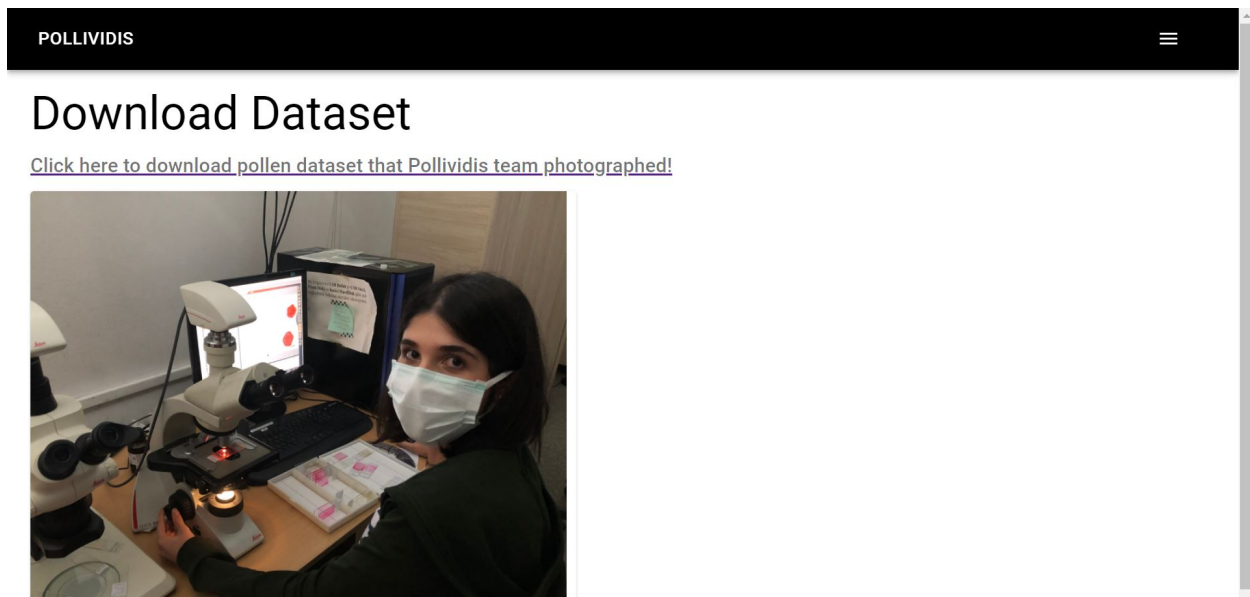


Figure 29: Download Dataset

9.12 Feedback Page

Users can give feedback to developers from the Send Feedback button in the navigation menu. In that page, they can share their opinions about PolliVidis with their names and email addresses.

We value your feedback

Your name *

Your email *

Your Feedback *

SEND FEEDBACK

Figure 30: Feedback

9.13 How PolliVidis Works Page

In order to see how PolliVidis works, users can press the How PolliVidis Works button in the navigation menu.

POLLIVIDIS
☰

How Pollividis Works

[Click here to see user manual!](#)

The Procedure

Sample image turns into **gray scale** image for thresholding.

Dilation and thresholding are applied to separate regions.

Thresholded image is passed to the **labeling** procedure to label each region.

Labeled pollens are **cropped** and forwarded to the **CNN model** one-by-one.

CNN Model

prediction

From the labeled sample image, **original image** is labeled with the **predictions of CNN model**.

Pollividis Pollen Sample Analysis can be divided into subroutines as

Figure 31: How PolliVidis Works?

10. References

- [1] K. Fakhroudinov, "The Unified Modeling Language," UML Diagrams - overview, reference, and examples. [Online]. Available: <https://www.uml-diagrams.org/>. [Accessed: 26-Feb-2022].
- [2] "Use case diagram," Wikipedia, 30-Oct-2021. [Online]. Available: https://en.wikipedia.org/wiki/Use_case_diagram. [Accessed: 26-Feb-2022].
- [3] "UML sequence diagram tutorial," Lucidchart. [Online]. Available: <https://www.lucidchart.com/pages/uml-sequence-diagram>. [Accessed: 26-Feb-2022].
- [4] "Unified modeling language (UML): Activity Diagrams," GeeksforGeeks, 13-Feb-2018. [Online]. Available: <https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams/>. [Accessed: 26-Feb-2022].
- [5] A. Athuraliya, "What is a deployment diagram: Deployment diagram tutorial," Creately Blog, 27-Sep-2021. [Online]. Available: <https://creately.com/blog/diagrams/deployment-diagram-tutorial/>. [Accessed: 26-Feb-2022].
- [6] "Class diagram," Wikipedia, 08-Dec-2021. [Online]. Available: https://en.wikipedia.org/wiki/Class_diagram. [Accessed: 26-Feb-2022].
- [7] "IEEE Reference Guide." IEEE Periodicals, New Jersey, 12-Nov-2018.
- [8] "About the customer matching process," Google Ads Help. [Online]. Available: <https://support.google.com/google-ads/answer/7474263?hl=en>. [Accessed: 08-Nov-2021].
- [9] "How fast should my website load?," Blue Corona, 20-Aug-2021. [Online]. Available: <https://www.bluecorona.com/blog/how-fast-should-website-be/>. [Accessed: 08-Oct-2021]
- [10] "Pytorch," *PyTorch*. [Online]. Available: <https://pytorch.org/>. [Accessed: 26-Feb-2022].
- [11] "scikit-image," *scikit*. [Online]. Available: <https://scikit-image.org/>. [Accessed: 26-Feb-2022].
- [12] "React – a JavaScript library for building user interfaces," – *A JavaScript library for building user interfaces*. [Online]. Available: <https://reactjs.org/>. [Accessed: 26-Feb-2022].
- [13] *Django*. [Online]. Available: <https://www.djangoproject.com/>. [Accessed: 26-Feb-2022].
- [14] *Google maps platform*. [Online]. Available: <https://developers.google.com/maps>. [Accessed: 26-Feb-2022].

[15] "Axios in react: A guide for beginners," *GeeksforGeeks*, 29-Apr-2022. [Online]. Available:<https://www.geeksforgeeks.org/axios-in-react-a-guide-for-beginners/>. [Accessed: 02-May-2022].

[16] Kişisel Verileri Koruma Kurumu: KVKK: Kişisel Verileri Koruma Kurumu Başkanlığı," Kvkk. [Online]. Available: <https://www.kvkk.gov.tr/>. [Accessed: 10-Nov-2021].