CS 492
Senior Design Project
2021 Spring

Low-Level Design 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

Currently, the process of identifying a pollen is considered as a tedious job since it must be done manually. Since the samples of pollen are collected manually an image requires manual labor to process. Furthermore, identifying pollen from irrelevant noise or content as well as distinguishing pollen variants from one another must be done through trained eyes, which itself is an inconvenient job. Additionally, training new students to identify pollens requires further effort, especially distinguishing pollens with similar granular cell structures.

Fortunately, along with the developments in machine learning, most systems are transforming into automated models. The advancements of image processing and image classification highly inspires an automated system that could improve the manual process of pollen classification. 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, the branch of biology which examines pollen. Hence, we aim to develop a system that can 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. Furthermore, this application would aid students trying to learn the details of palynology. PolliVidis would allow users to easily upload an image to the web application and get the results. Apart from being widely accessible, PolliVidis also aims to create a database for palynologists to share pollen data across the country.

1.1 Object Design Trade-offs

1.1.1 Security vs Efficiency

Since using hashing algorithms to store passwords needs extra time for calculations, it may cause delay in the system's response time. However, 200 milliseconds of response time, which is the maximum response time recommended by Google [1], is decided for this process. Hence, we aim to have an optimal balance between security and efficiency.

1.1.2 Functionality vs Usability

Ease of use is one of the main concerns while designing the front-end of the application. It can be said that front-end design favors usability over functionality, however, it is important to note that our main functionality is related to classification and counting of the

pollen samples, and it will not be affected by the decision of choosing usability over functionality.

1.1.3 Compatibility vs Cost

PolliVidis is a web-based application which will be working on with the most popular four web browsers which are Google Chrome, Safari, Microsoft Edge and Mozilla Firefox [2]. To increase the compatibility, it is considered developing a mobile application, however, it would increase the cost and it is decided that web application offers enough compatibility.

1.1.4 Speed vs Functionality

The functionality of the ML model will be favored over the response speed of the system since one of the main aims of the system is to help academics by classifying the pollen species and count pollen with accurate results.

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

Low Level Design Report diagrams follow the UML guidelines [3] as the previous reports of PolliVidis.

Until this report; PolliVidis has used UML Use Case Diagram to describe user interactions with the web application [4], UML Sequence Diagram to describe the lifeline of an object [5], UML Activity Diagram to describe the control flow of the system [6], and UML Deployment Diagram to define the hardware communication of the system [7].

In the Low-Level Design Report, PolliVidis uses UML Class Diagram to describe statically the structure of the system [8]. 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 [9].

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 pollen.

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.

PyTorch: Python package for Machine Learning and Deep Learning.

2. Packages

2.0 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.

2.1 Client

2.1.1 Presentation Tier

2.1.1.1 UI Subsystem

UI Subsystem consists of user interface front-end views.

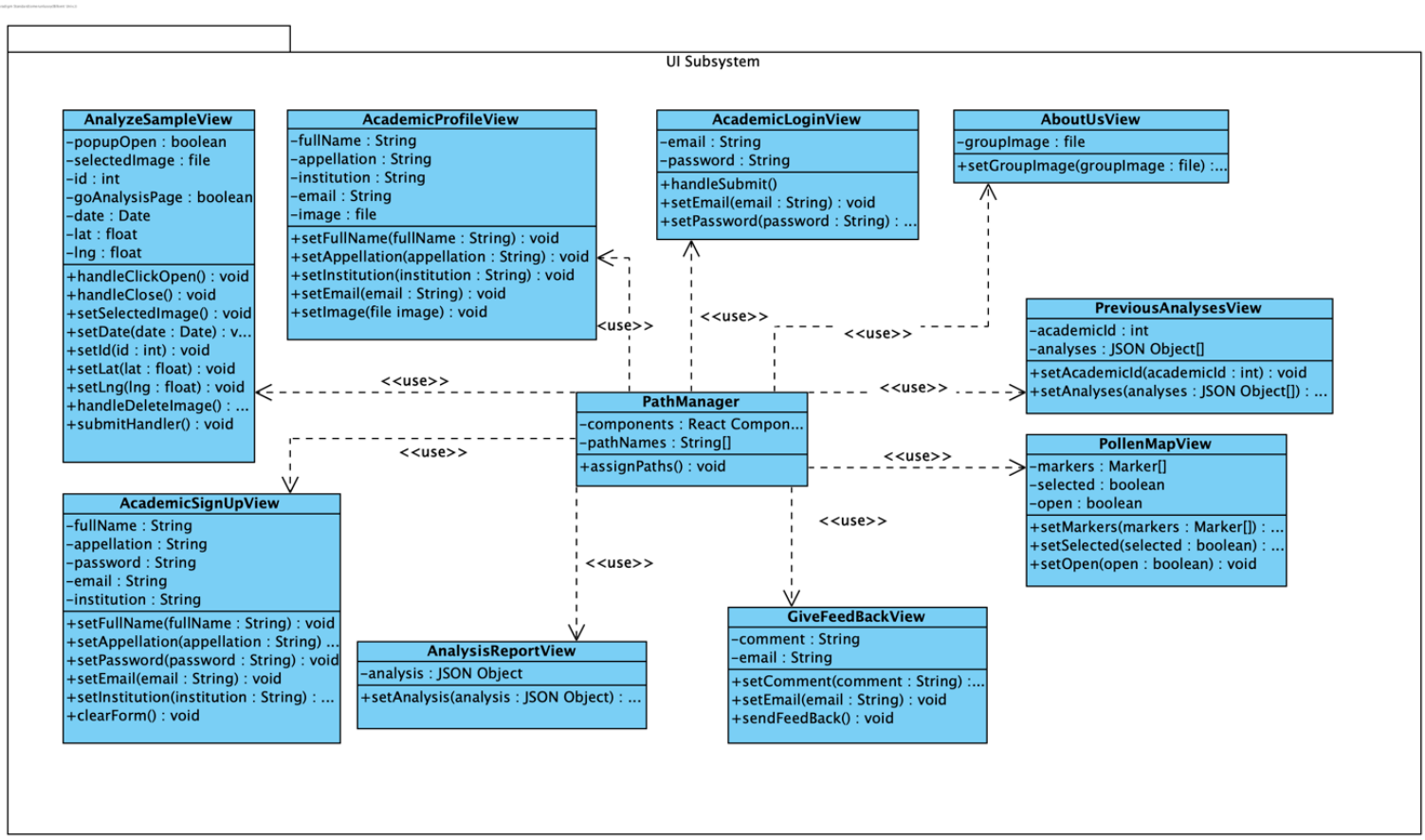

Figure 1: UI Subsystem

2.1.1.1.1 Path Manager

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

2.1.1.1.2 AnalyzeSampleView

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

2.1.1.1.3 AnalysisReportView

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

2.1.1.1.4 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.

2.1.1.1.5 PreviousAnalysisView

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

2.1.1.1.6 AcademicLoginView

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

2.1.1.1.7 AcademicSignUpView

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

2.1.1.1.8 AcademicProfileView

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

2.1.1.1.9 AboutUsView

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

2.1.1.1.10 GiveFeedBackView

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

2.2 Server

2.2.1 Logic Tier

2.2.1.1 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.

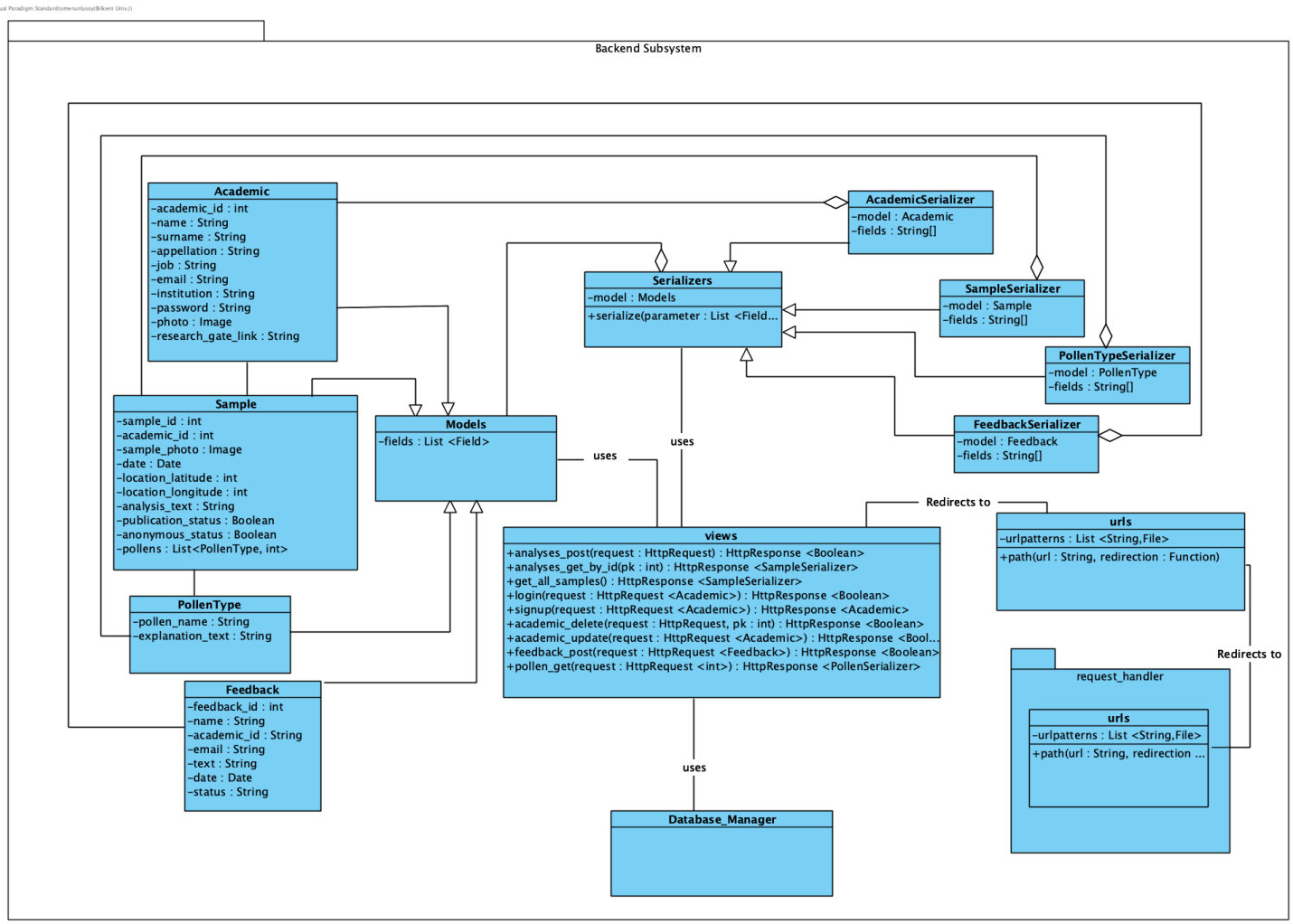

Figure 2: Backend Subsystem

2.2.1.1.1 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.

2.2.1.1.2 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.

2.2.1.1.3 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.

2.2.1.1.4 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.

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

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

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

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

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

2.2.1.1.10 Serializer

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

2.2.1.1.11 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.

2.2.1.1.12 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.

2.2.1.1.13 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.

2.2.1.2 ML Subsystem

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

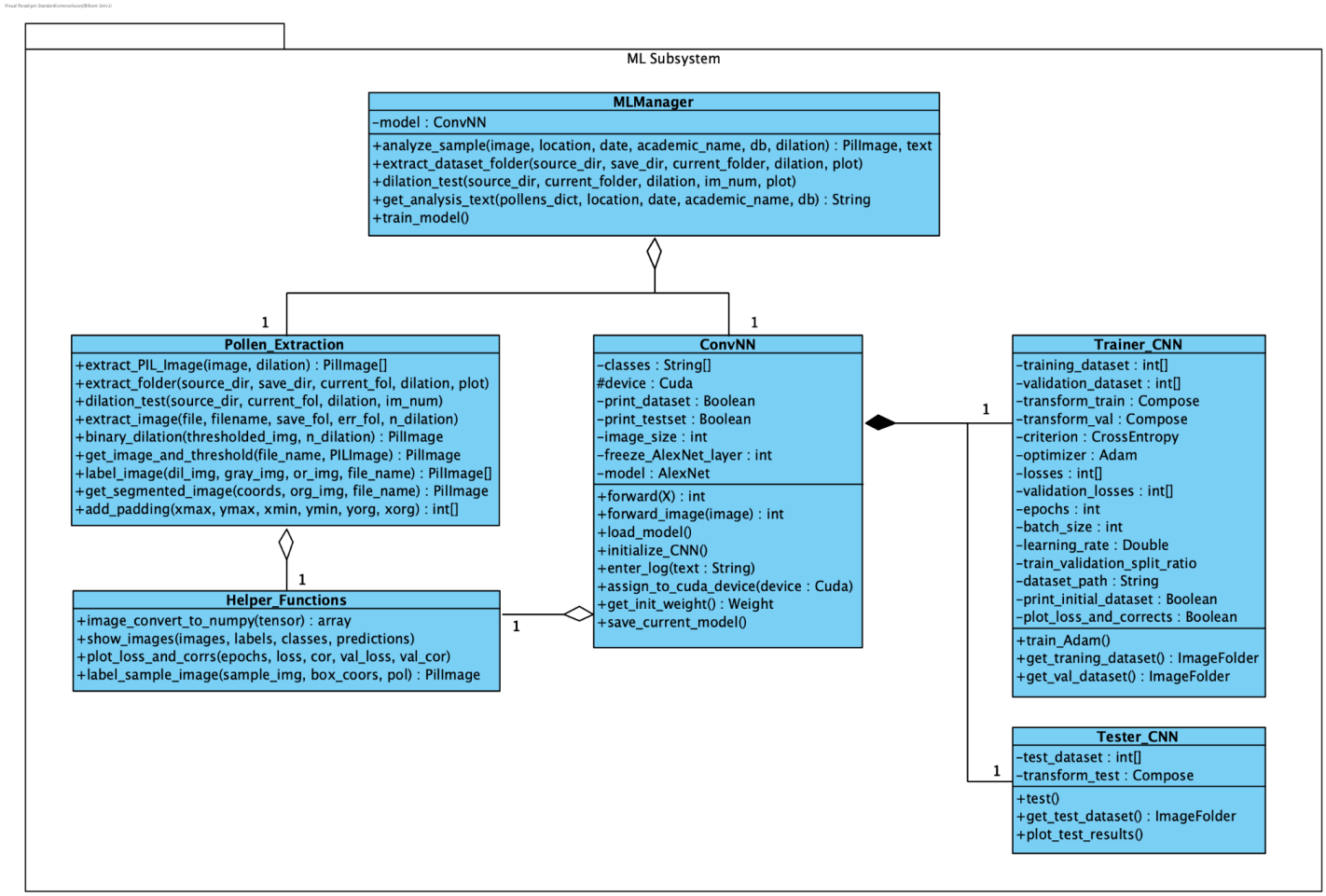

Figure 3: ML Subsystem

2.2.1.2.1 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.

2.2.1.2.2 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.

2.2.1.2.3 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.

2.2.1.2.4 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.

2.2.1.2.5 Tester_CNN

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

2.2.1.2.6 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.

2.2.2 Data Tier

2.2.2.1 Database Subsystem

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

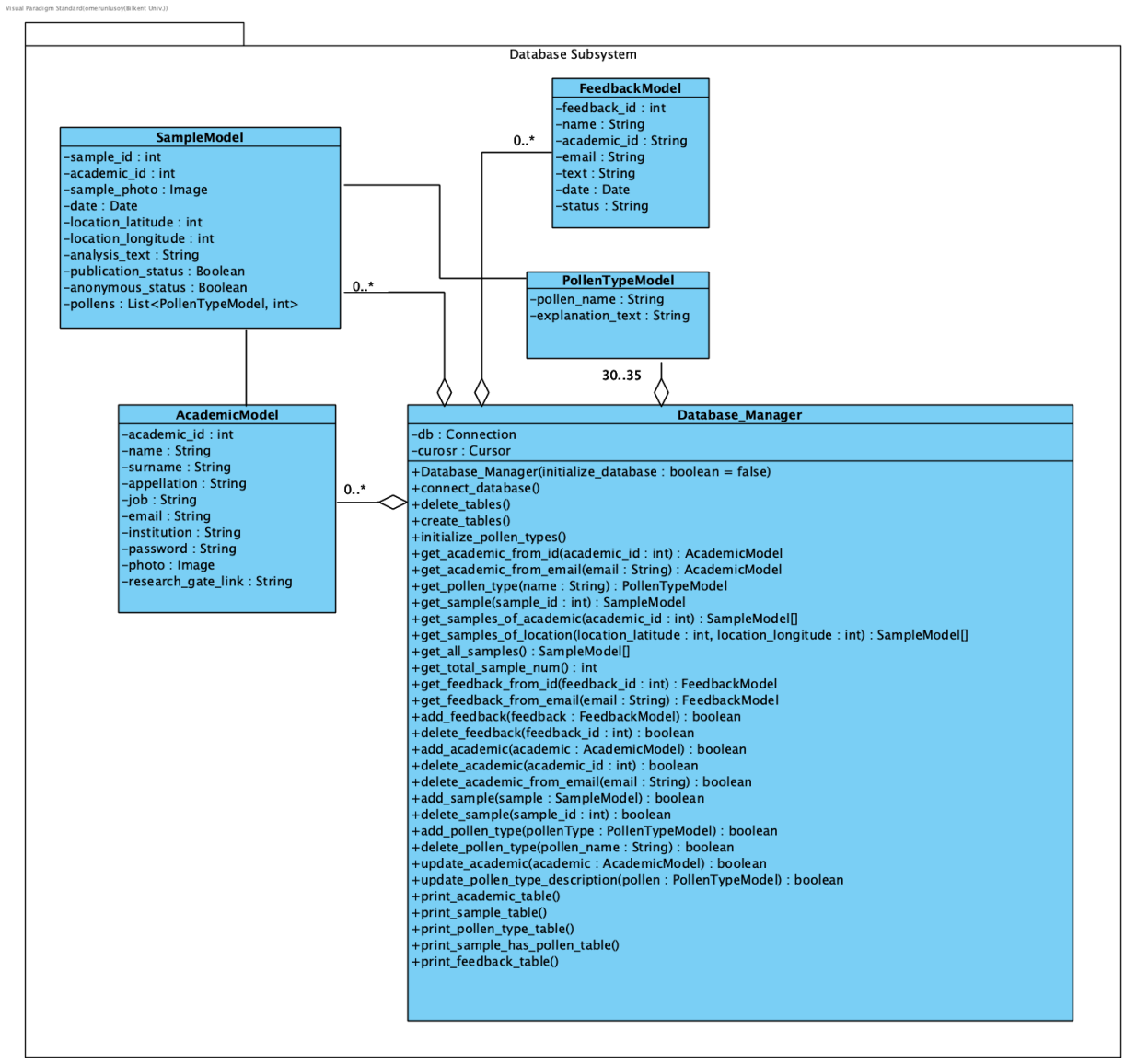

Figure 4: Database Subsystem

2.2.2.1.1 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.

2.2.2.1.2 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.

2.2.2.1.3 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.

2.2.2.1.4 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.

2.2.2.1.5 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. Class Interfaces

3.0 Introduction

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

3.1 Client

3.1.1 Presentation Tier

3.1.1.1 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 Server

3.2.1 Logic Tier

3.2.1.1 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 in 2.2.1.1.13.	
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.

3.2.1.2 ML Subsystem

Class 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.	
Attributes	
private ConvNN model	
Methods	
analyze_sample(image, location, date, academic_name, db, dilation) : PilImage, 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	
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.	
Methods	
extract_PIL_Image(image, dilation) : PilImage []	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) : PilImage	This function applies binary dilation to given image.
get_image_and_threshold(file_name, PILImage) : PilImage	This function loads the image from a filepath and applies thresholding to the image.
label_image(dil_img, gray_img, or_img, file_name) : PilImage []	This function labels the binary thresholded image to separate regions for pollen extraction.
get_segmented_image(coords, org_img, file_name) : PilImage	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_traning_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 Data Tier

3.2.2.1 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	
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.	
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) : boolean	Updates pollen type tuple with the given parameters. Returns true if successful, 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

4. Extraction Process

The pollen extraction from a given sample image is summarized in the following image with the gray scaling, dilation, thresholding, labeling, cropping, classification, and analyzed imaged construction.

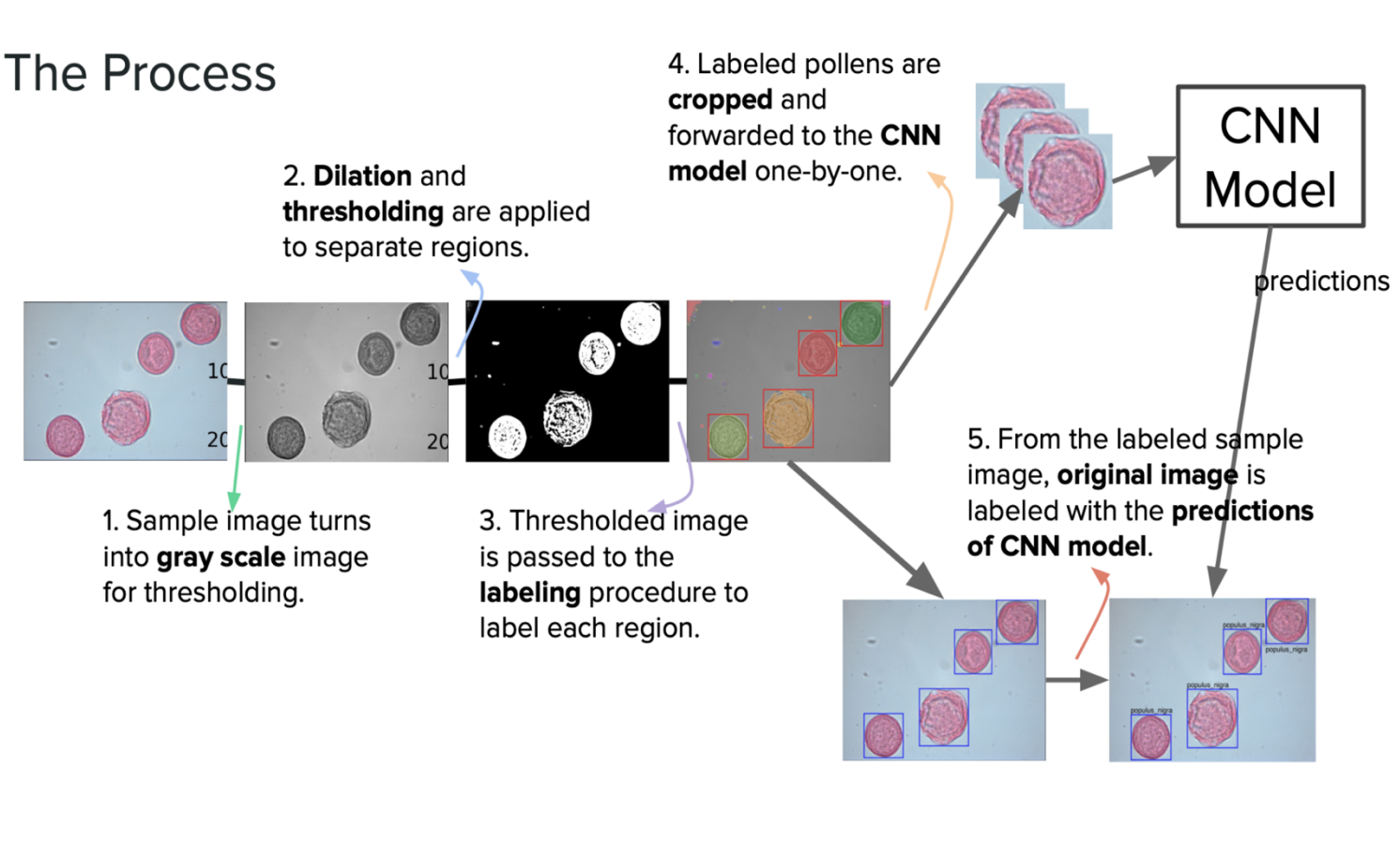

Figure 5: The Process

5. References

[1] “7 ways to reduce server response time,” *Knowledge Base by phoenixNAP*, 24-Jun-2021. [Online]. Available: https://phoenixnap.com/kb/reduce-server-response-time. [Accessed: 26-Feb-2022].

[2] “Global Desktop Browser Market Share for 2022,” *Kinsta®*, 07-Oct-2021. [Online]. Available: https://kinsta.com/browser-market-share/. [Accessed: 26-Feb-2022].

[3] K. Fakhroutdinov, “The Unified Modeling Language,” *UML Diagrams - overview, reference, and examples.* [Online]. Available: https://www.uml-diagrams.org/. [Accessed: 26-Feb-2022].

[4] “Use case diagram,” *Wikipedia*, 30-Oct-2021. [Online]. Available: https://en.wikipedia.org/wiki/Use_case_diagram. [Accessed: 26-Feb-2022].

[5] “UML sequence diagram tutorial,” *Lucidchart*. [Online]. Available: https://www.lucidchart.com/pages/uml-sequence-diagram. [Accessed: 26-Feb-2022].

[6] “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].

[7] 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].

[8] “Class diagram,” *Wikipedia*, 08-Dec-2021. [Online]. Available: https://en.wikipedia.org/wiki/Class_diagram. [Accessed: 26-Feb-2022].

[9] “IEEE Reference Guide.” IEEE Periodicals, New Jersey, 12-Nov-2018.

[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].