Bilkent University

Department of Computer Engineering

Senior Design Project

Nino: Nino Is Not OCR

Low-Level Design Report

Ata Deniz Aydın, Ecem İlgün, Ergün Batuhan Kaynak, Selim Fırat Yılmaz

Supervisor: Hamdi Dibeklioğlu

Jury Members: A. Ercüment Çiçek and Özcan Öztürk

Low-Level Design Report
Feb 18, 2019

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.

Contents

1 Introduction

Note taking amounts to an important part of studying materials for many students, be they grade school or graduate students. In many cases, there are no lecture notes or slides readily available, or the ones that are available do not cover all of the lecture content. This requires students to take notes during the lecture.

However, in some cases, the student has to note down the words fast without truly processing their content, or write down equations without having the time to analyze and thus understand them. In the case of writing down sentences, one can achieve faster note taking with tablets/computers [1] but when the lecture notes consist of plots, graphs or mathematical equations, our market research shows that there is no hardware/software help to achieve faster note taking than writing it down [2].

In some cases, scribing all the things written on the board (even without making sense of it) proves to be quite hard, therefore many note taking strategies have arisen [3]. Many of these strategies revolve around the rule that one first takes notes of keywords, and then rewrites the notes after class [3]. However, there are two possible problems with this strategy: the student might forget the information conveyed via that particular keyword, or might miss important information and not write a keyword about it.

Currently, EverNote™, OneNote™ and Google Docs™ offer OCR (optical character recognition) features in which an image is converted to a text. However, these conversions do not cover mathematical equations or figures. As it is discussed above, many courses include some kind of figures or mathematics, therefore capturing only text with OCR might lead to loss of important information. Hence, current software are not capable of providing a complete note taking system.

We thus propose an app that will convert handwritten and printed notes into an editable format which can be exported as LaTeX or PDF. These notes will preserve the alignment of the original picture, in either landscape or portrait format. The student will be able to add his/her own notes on top of the ones extracted from the image. In this way, we hope to engineer a note taking system that would make the lecture not only more interactive for the student, but will also let the student listen to more of the lecture.

1.1 Design tradeoffs

1.1.1 Usability vs Functionality

We aim to provide an easy and smooth experience in order to attract and keep users. Users should be able to navigate through the client apps without having problems on how to use it, the process should seem natural without a high learning curve. User interfaces for web and Android should not differ too much, so that they do not seem like altogether different products for users switching devices. Therefore, usability of the system is an important design goal. However, we wish to add several functionalities in order to make Nino a novel product on the marketplace. Therefore, we cannot diminish functionality. Nino aims to balance between functionality and usability.

1.1.2 Extensibility & Modularity vs Performance

The system will be highly modular. Any additional modules will be integrated without causing any problems to other modules. The system will be implemented in a way that addition of extra modules will be easy. Adding extra modules to the system should be straightforward as long as the module functions correctly by itself. Hence, we decided to favor use of patterns -e.g. visitor- and concepts -e.g. inheritance- over performance.

1.1.3 Performance vs Cost

There will be quite a bit of processing when notes are being created. The user should not hang waiting for a long time to keep both interactivity and usability at acceptable levels. To this aim, each function module must not take more than 5 seconds to complete its process. However, we do not have the budget to buy hardware and distribute the processing, therefore the performance of the product is limited by our budget.

1.1.4 Availability vs Cost

Since all the note processing is done in a remote server and not locally, availability is at utmost importance. System uptime should be high to not cause any inconvenience to the user. Scheduled maintenances to the core functionality should be announced beforehand

and should not take very long. Since the system is highly modular, the whole system should not be down due to an update to a particular submodule. However, we aim to keep cost of this minimal.

1.1.5 Cost vs Accuracy

Since Nino relies on machine learning algorithms to provide its services, the system will have an accuracy rate in which wrong output is accepted to some degree. However, the accuracy of the output should be high enough so that the users will continue to use our services. Due to budget constraints, we have chosen to use free off the shelf software when needed rather than buying commercial ones. Free software might provide less accurate results than commercial ones, therefore, we choose to favor cost over accuracy.

1.2 Interface documentation guidelines

Class	ClassName
Class description	
Attributes	
`attribute`	Attribute description
Methods	
`method(arg1, arg2)`	Method description

1.3 Engineering standards

In our reports, we have relied on the Unified Modeling Language (UML) standard [10] to model class interfaces and interactions, the former through object, class, package and deployment diagrams and the latter through activity and sequence diagrams. We have also followed the Institute of Electrical and Electronics Engineers (IEEE) style [11] in citing references.

1.4 Definitions, acronyms, and abbreviations

Note: a handwritten picture containing regions of text, equations, plots and figures.

Sketch: any digitizable type of data apart from text and equations.

Segment: a maximal rectangular region in a note consisting of a single type of data (text, equations or sketches), enclosed by a bounding box.

Category: a directory containing notes of the same context (e.g. course topic).

Client: the mobile or web application interacting with the user.

Server: the system communicating with each client and processing images sent from them.

Segmentation: partitioning a given image into different segments.

Detection: identifying segments of a particular type (e.g. text segments) without necessarily recognizing the data stored in the segment.

Recognition: identifying the content in a particular segment (e.g. recognizing the text written in a given text region).

Annotation: improving and adding to the content stored in each segment, e.g. identifying keywords in text and adding hyperlinks to keywords.

2 Packages

The large-scale architecture of the application will follow the client-server model. The client side will be in the form of a mobile or web application that interacts with a single user, allowing the user to capture images and send them to the server to be converted to a note, or view or edit already created notes. The server side will handle most of the data processing necessary to convert images into notes, such as region detection and recognition, as well as maintaining the data stored for each user including notes stored in the cloud and authentication information.

2.1 Client

The client is the user interface layer of Nino, where the user interacts with notes to create and edit them. The client contains the presentation and controller subsystems. The presentation subsystem contains user interface elements for the user to interact with. The controller subsystem is responsible of creating appropriate requests depending on the events initiated by the views and then communicating with the server to send the requests.

Along with custom created layouts and Android activities, Nino uses an open source note taking project, Scarlet-Notes, to enhance UI presentation and logic. Nino's activities and logic are connected to Scarlet-Notes (denoted as "base" in the diagrams), and most of the functionality is changed to fit Nino's use case.

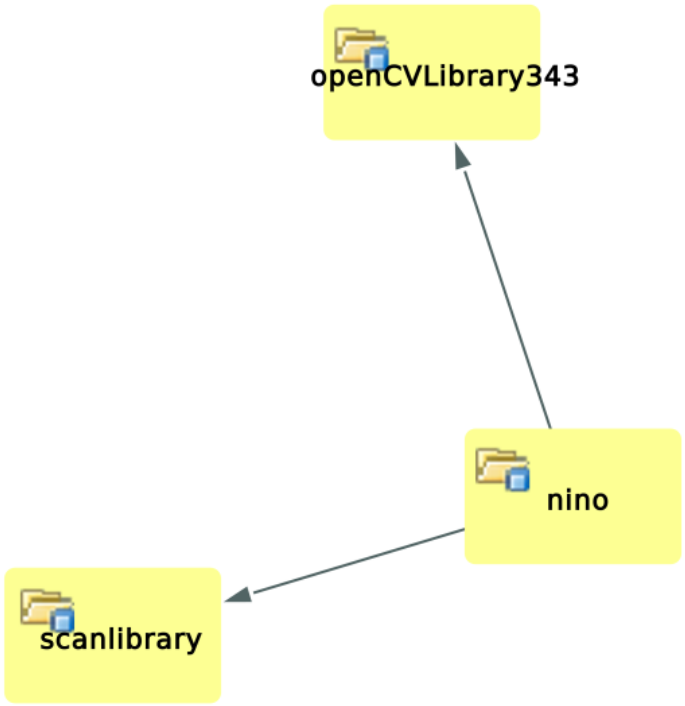

Figure 1: High level view of NinoClient. Package Nino contains all the presentation and logic subsystems and uses off the shelf libraries.

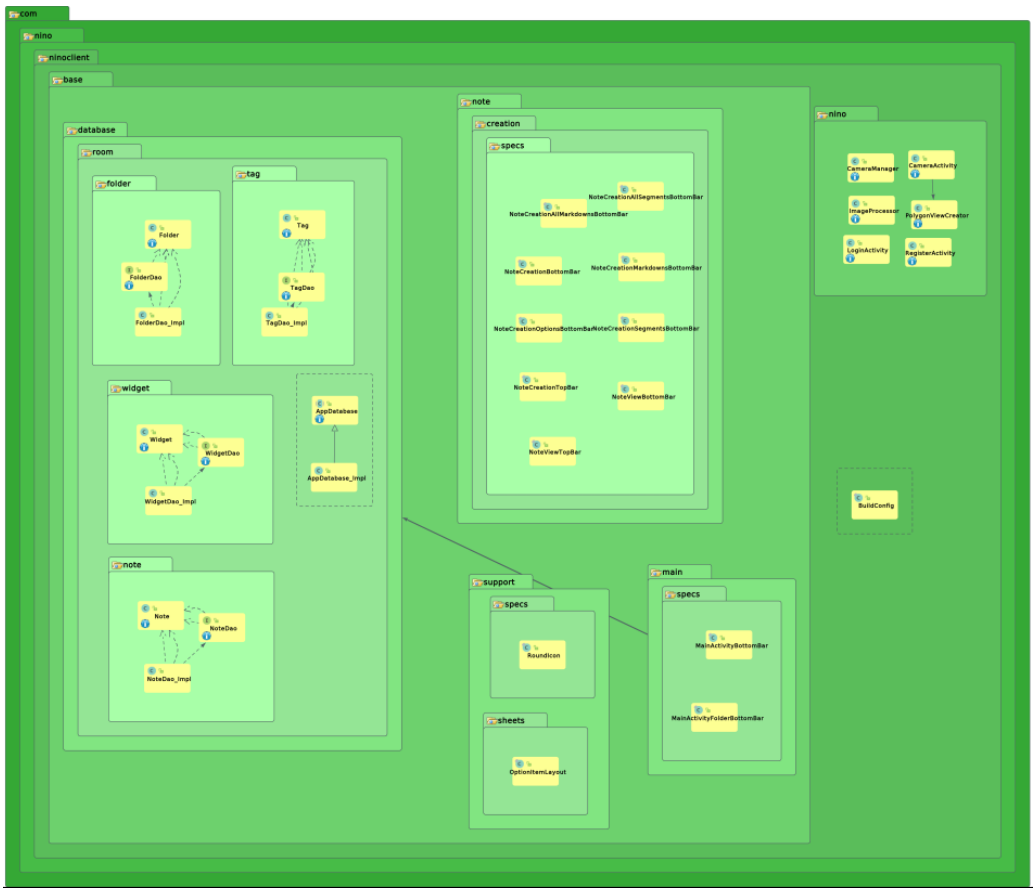

Figure 2: The partial* hierarchy of modules inside the client system. *: No automation to generate UML diagrams of Kotlin classes exists at this point, only java classes are shown.

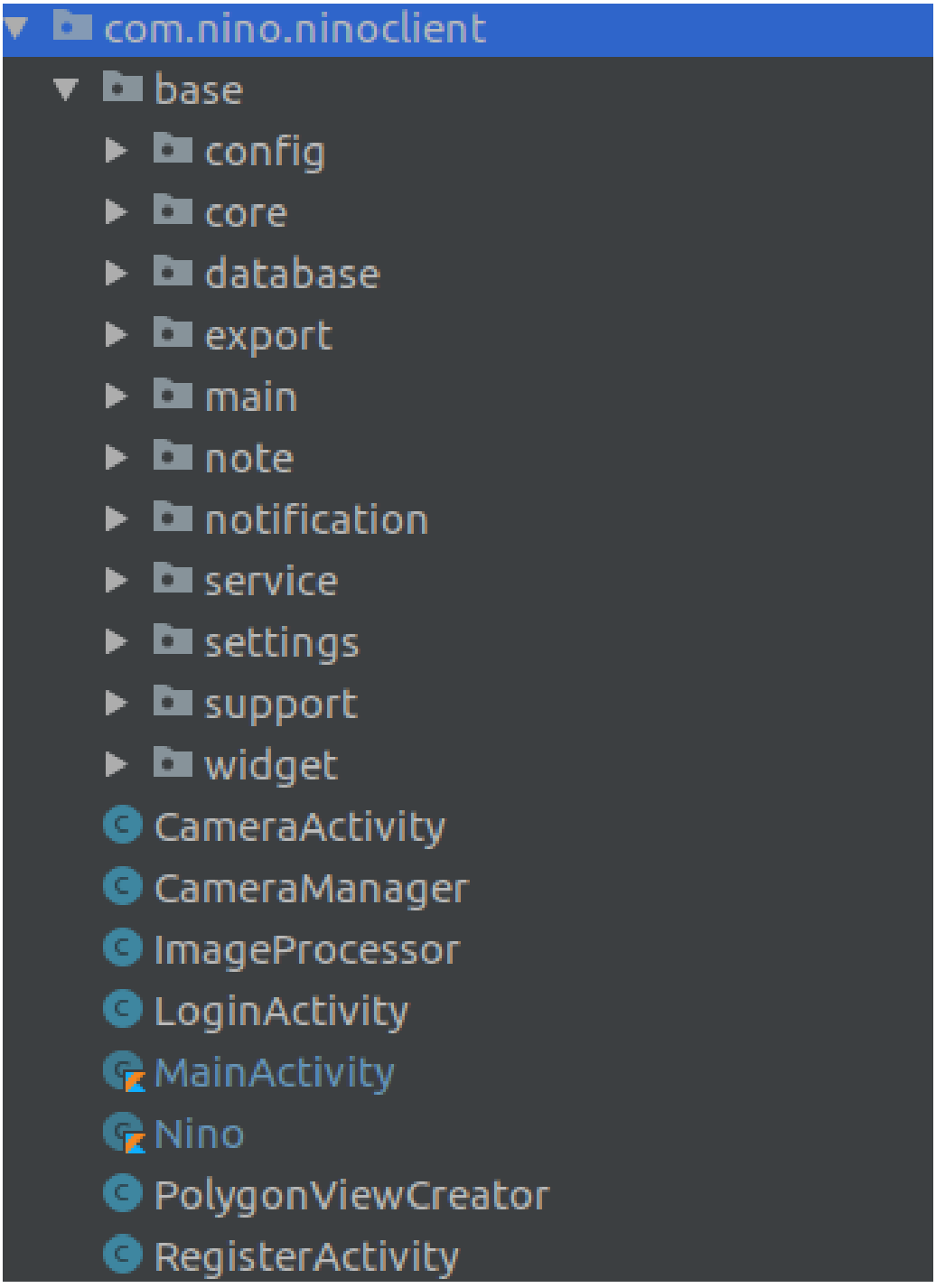

Figure 3: List of all non-library packages and classes.

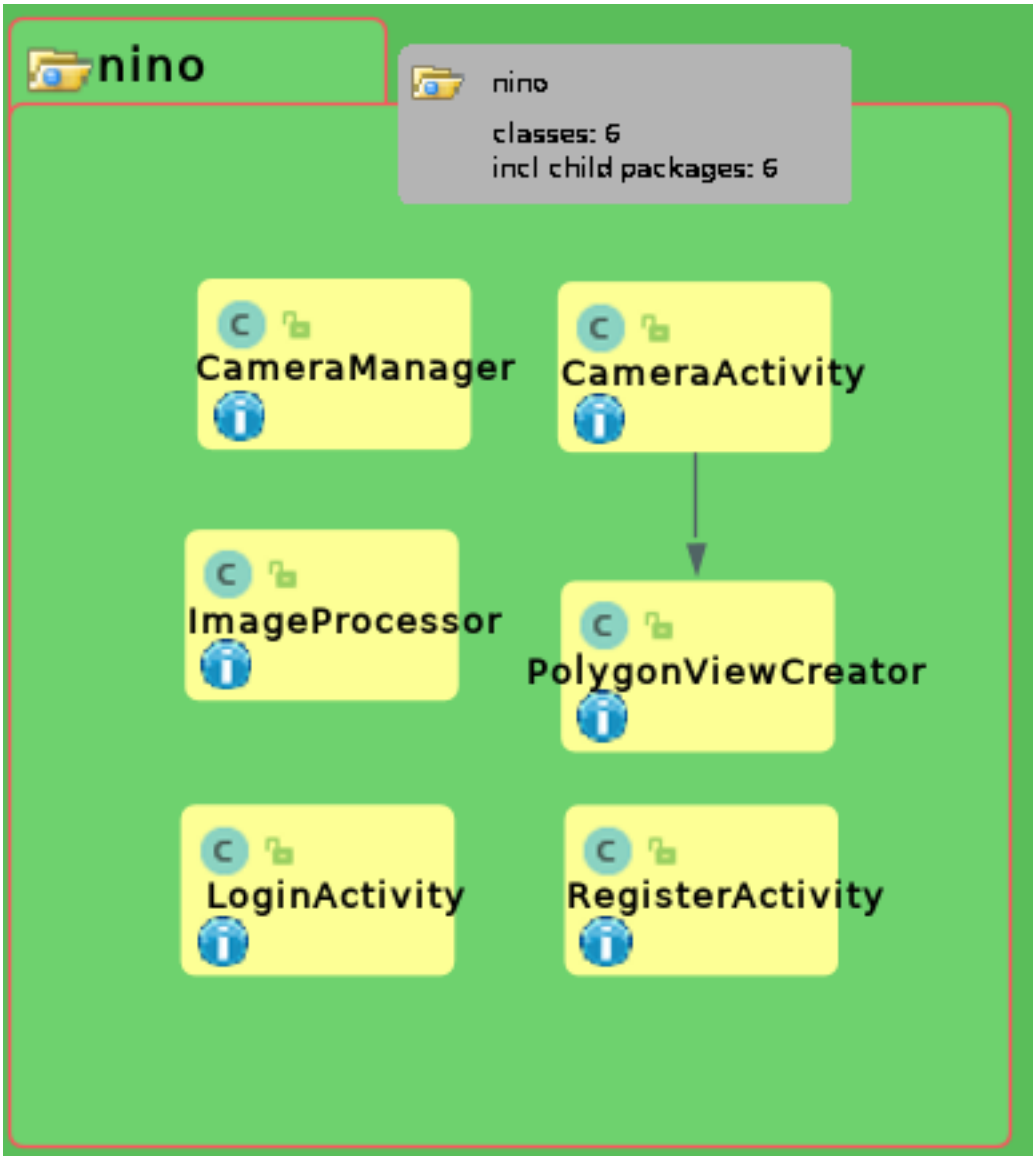

Figure 4: nino package.

2.1.1 Presentation

The presentation layer contains views and their event firing mechanisms.

LoginActivity: The view class for the login screen. Checks credential formats before asking for request creation. Launches MainActivity (located in package "base") if the user credentials are valid.

RegisterActivity: The view class for the register screen. Checks credential formats before asking for request creation. Launches LoginActivity if the user is created successfully.

CameraActivity: The view class for the camera screen. The user takes the picture to be processed by Nino here. Uses CameraManager class and opencv library to detect a region where actual writings, figures, or mathematical expressions are located in the image. Uses a PolygonView to let the user to select/edit the region if automatic region detection is not enough.

MainActivity: The view class for the notes (main) screen. Lists the notes of the user and allows interactions with them, such as editing and deleting. Can launch the CameraActivity to take pictures and initiate note creation.

2.1.2 Controller

The logic of the client executes operations like image processing and server communication, depending on the events created by the views on presentation layer.

CameraManager: Responsible of configuring the camera in Android apps and taking pictures.

ImageProcessor: Uses Opencv library to detect a region where actual writings, figures, or mathematical expressions are located in the image. Handles morphological operations, edge detection, image enhancement and warping to the images. Configures a PolygonView created by PolygonViewCreator to present the resulting region to the user.

PolygonViewCreator: Creates a PolygonView to let the user to select/edit the region if automatic region detection is not enough. PolygonView is an editable polygon region

that denotes which part of the image will be warped and sent to the Nino server to be processed.

2.2 Server

An image and the required operations on that image arrive as a request to the server. The server then processes the image with the given modules and creates an output. The output is then sent back to the client as a response. In addition, the server also handles authentication and updates to user information database. The server is separated into the Logic and Data layers. The request handling is also inside the server side of the project, and is maintained by Django.

The logic layer governs the database operations and the note processing pipeline. User management also maintains a list of active users, handles invocations to log in, log out or create a new account, and authorizes accesses or modifications to the storage of each user. Lastly, note processing receives an image, processes it and converts it into a note, or receives and processes existing notes either for the purpose of training the models used or annotating notes again after editing.

The data layer is where the models and other persistent information are kept. The subsystem for data management holds for each user the storage space of notes reserved for them, as well as authentication data and shared data used in note processing and other configuration data.

The subsystem for note processing is itself broken down into several submodules: preprocessing, segmentation, text recognition, mathematical expression recognition and text analysis, which are executed in a pipeline. The image received is first sent to the preprocessor module, which denoises and straightens the image, and afterwards to the segmentation module, where regions for text, equations, plots and figures are identified and organized in a tree data structure. Each subsequent module then operates on and annotates this data structure, functioning as an intermediate representation shared across modules. For example, the module for text recognition visits each text region in the tree, processes the image lying within the region, and writes the text it has read back to the node corresponding to that region.

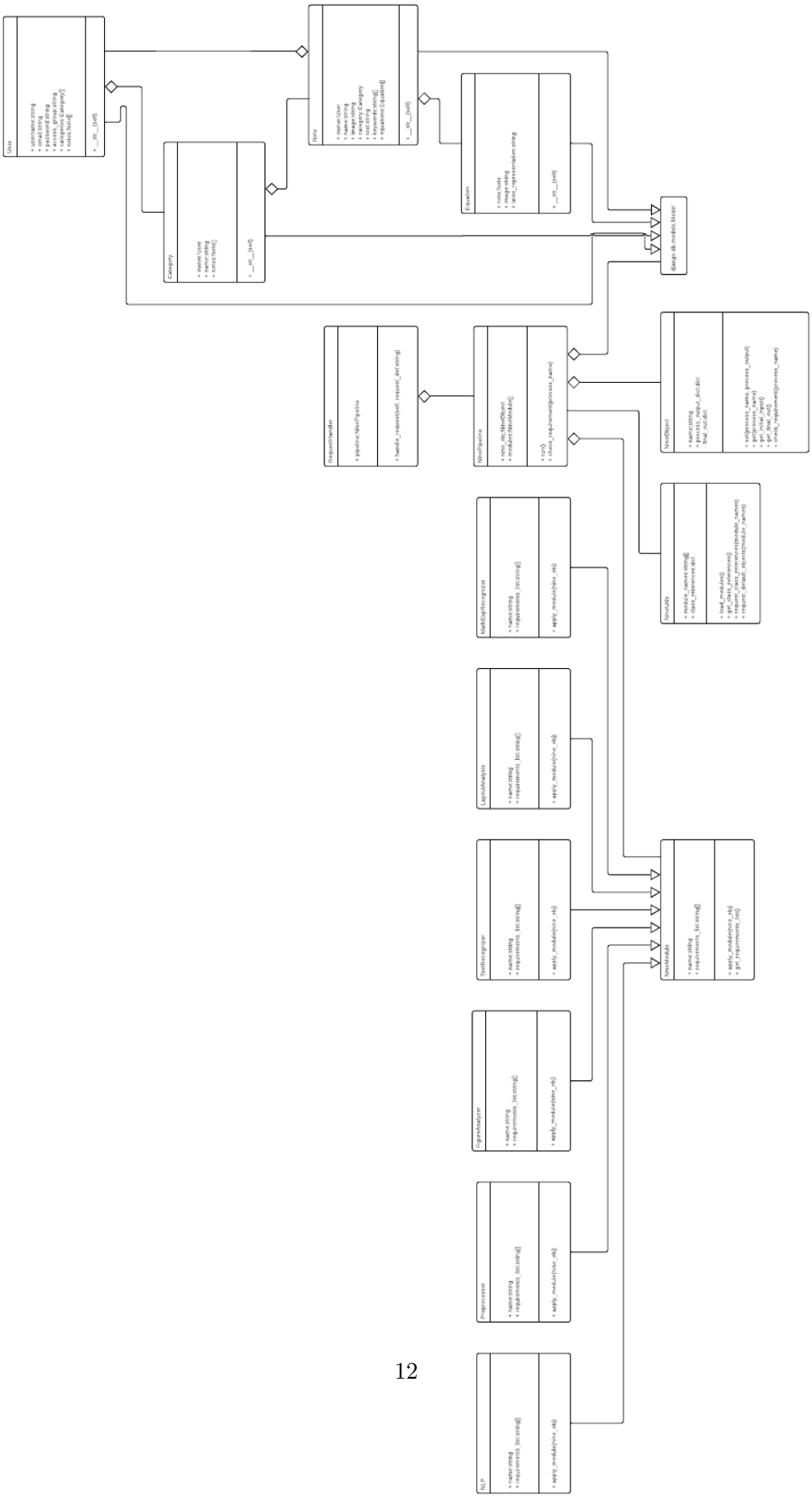

Figure 5: Server UML Diagram.

2.2.1 Logic

This is the subsystem responsible of Nino's logic and processing. Pipeline and Modules are two packages that constitute these operations.

RequestHandler: Turns client requests into tasks and runs them in the pipeline. Creates responses and sends them back to the client.

2.2.2 Pipeline

Nino adopts a modular design. The pipeline package is the key component in realizing this design choice.

NinoPipeline: After an incoming request, all the required modules are fetched from the Modules package and turned into sequential operations. The pipeline runs these modules one after the other to produce an output.

NinoObject: This is the object that is created using a request. The pipeline then processes this object; modifying it to create the final output note.

Utils: Some utility functions for NinoPipeline and NinoObject.

2.2.3 Modules

Modules package contains the actual modules used to create notes.

LayoutAnalysis: Segments images into regions.

FigureAnalyzer: Processes figures in images, by e.g. denoising, straightening etc.

TextRecognizer: Recognizes text in text regions.

MathExpRecognizer: Recognizes mathematical expressions in math regions.

NLP: Contains operations related to natural language processing.

2.2.4 Data Access Objects

This layer is where the data access objects used by the application are kept.

User: User data access object.

Note: The notes created by the users.

Category: The category object that keeps notes organized.

Equation: The equation object that keeps equation information.

3 Class interfaces

3.1 Client

Automatically generated Java interface from Nino's current prototype can be found in Appendix.

3.1.1 Presentation

Class	LoginActivity
The view class for the login screen.	
Attributes	
`REQUEST_READ_CONTACTS`	int id for permission request of reading contacts
Methods	
`onCreate(savedInstanceState)`	Performs initialization of all fragments.
`attemptLogin()`	Checks the user's username and password and logs in if they are in the database.
`isUsernameValid(username)`	Checks if `username` is valid, i.e. holds the length etc. constraints.
`isPasswordValid(password)`	Checks if `password` is valid, i.e. holds the length etc. constraints.
`showprogress(show)`	Shows the progress bar of the loading page

Class	RegisterActivity
The view class for the registering screen.	
Attributes	
`REQUEST_READ_CONTACTS`	int id for permission request of reading contacts
`registerTask`	keeps a register request from the server
`usernameView`	An AutoCompleteTextView* for the user to enter their username.
`emailView`	An AutoCompleteTextView* for the user to enter their email.
`passwordView`	Text field for the password.
`passwordRepeatView`	Text field for repeating the password.
`progressView`	The view called by showProgress method. It shows the progress bar during communication with the server for the register request.
`loginFormView`	The view for login form (username & password entering screen).
Methods	
`onCreate(savedInstanceState)`	Performs initialization of all fragments.
`populateAutoComplete()`	Fills an AutoCompleteTextView* with autocompletion advises.
`mayRequestContacts()`	Returns if the contacts can be read, based on permissions.
`onRequestPermissionsResult( requestCode, permissions, grantResults)`	Calls populateAutoComplete() if permission to read contacts is granted.
`isUsernameValid(username)`	Checks if `username` is valid, holds the length etc. constraints.
`isEmailValid(email)`	Checks if `email` is valid, holds the length etc. constraints.
`isPasswordValid(password)`	Checks if `password` is valid, holds the length etc. constraints.
`isRepeatPasswordValid(password, passwordRepeat)`	Checks if `password` and `repeatPassword` match and are valid.
`showprogress(show)`	Shows the progress bar of the loading page
`onCreateLoader(i, bundle)`	Instantiate and return a new Loader, that loads data, for the given id `i`.
`onLoadFinished(cursorLoader, cursor)`	Called when a previously created Loader has finished its load. [13]. Adds emails to its AutoCompleteTextView.
`onLoaderReset(cursorLoader)`	Called when a previously created loader is being reset, and thus making its data unavailable[13].
`addEmailsToAutoComplete(emailAddressCollection)`	Add list of emails to AutoCompleteTextView.

*: AutocompleteTextView is an editable text view that shows completion suggestions automatically while the user is typing [12]

Class	CameraActivity
The view class for the 'taking photo of notes' screen.	
Attributes	
`REQUEST_TAKE_PHOTO`	int id for request of using android camera and taking a photograph.
`mCurrentPhotoPath`	Current directory path that the photos are saved in.
`rgba`	Matrix for RGB color values with an alpha channel.
`mainlv`	Main image view.
`pvc`	PolygonViewCreator for the camera activity page.
`progressView`	The view called by `showProgress()` method. It shows the progress bar during communication with the server for saving the photo.
`cameraView`	The view of android camera for taking photos inside the app.
`token`	Token to check against the server token. If matched the user is authorized
`edgeDetectedTakenImage`	Bitmap of edges of the photograph
`finalImage`	The image bits shown to user.
`warpButton`	The warp button.
Methods	
`onCreate(savedInstanceState)`	Performs initialization of all fragments.
`dispatchTakePictureIntent()`	Creates & dispatches an intent to take a photo with the android camera and share it with the app.
`onActivityResult(requestCode, resultCode, data)`	If picture is taken: calls `rotateBitmapOrientation()` and `detectNote()` on the picture.
`persistImage(bitmap)`	Creates a image file and saves a bitmap image, e.g. `finalImage`, there.
`saveBitmapToFile(file)`	Saves the bitmap image to a file.
`detectNote(bitmap)`	Detects edges of the note in the photograph and returns bitmap of the note.
`matToBit()`	Converts Mat to Bitmap type.
`warp(inputMat, startM)`	Warps the image to represent the note better as a scanned image.
`rotateBitmapOrientation(photoFilePath)`	Normalizes the image's orientation to portait, as in regular A4 paper print-outs.
`createImageFile()`	Creates and saves an image file.
`showprogress(show)`	Shows the progress bar of the loading page.
`changeViewVisibility(show)`	Alters visibility of the `cameraView`
`onResume()`	s invoked when CameraActivity is on the foreground. Initializes OpenCV library.

3.1.2 Controller

Class	CameraManager
Manager class for configuring camera and taking pictures	
Methods	
`rotateBitmapOrientation( photoFilePath)`	Normalizes the image's orientation to portait, as in regular A4 paper print-outs.

Class	PolygonViewCreator
Creates and manages a PolygonView as explained in section 2.1.2.	
Attributes	
`polygonView`	An instance of PolygonView.
`rWidth`	Width ratio of the image bitmap
`rHeight`	Height ratio of the image bitmap
`bitmapPos`	Keeps the positions `getBitmapPositionInsideImageView()` returns.
Methods	
`createPolygonWithCurve(approxCurve, rgba, iv)`	Sets the empty PolygonView instance with the data from ImageView, RGBA values and the approximate curve obtained from contours.
`createPolygonWithRect(rect, rgba, iv)`	Sets the empty PolygonView instance with the data from ImageView, RGBA values and according the points defined by rect.
`getBitmapPositionInsideImageView (imageView)`	Returns width and height of image & its left and top positions.
`getOutlinePoints(tempBitmap)`	Returns top-left, top-right, bottom-left and bottom-right points of the image bitmap.
`getPoints()`	Returns points that define the polygon.
`getBitmapWidth()`	Returns width of bitmap.
`getBitmapHeight()`	Returns height of bitmap.

Class	ImageProcessor
Detects where the notes are in the picture, preprocesses it for further use.	
Methods	
`detectEdges(bitmap, threshold)`	Detects edges of the note.
`findContours(edges)`	Returns contours with the largest area.
`findApproxCurve(maxContour)`	Finds and returns the best curve approximation of the image from the given countor area.

3.2 Server

3.2.1 Logic

Class	RequestHandler
Handles requests to the server side API of Nino.	
Attributes	
`pipeline`	NinoPipeline object.
Methods	
`handle_request(self, request_def)`	Handles the request given by request_def.

3.2.2 Pipeline

Class	NinoModule
A module processing note objects.	
Attributes	
`name`	Name of the module.
`requirements_list`	List of modules that need to be applied to the note before this module.
Methods	
`apply_module(nino_obj)`	Process the given note object and record the output in the object.
`get_requirements_list()`	Return the list of requirements.

Class	NinoObject
A note object, which maintains a dictionary of outputs after each stage of processing.	
Attributes	
`name`	Name of the object.
`process_output_dict`	Dictionary mapping each process name to the output of the process after processing the object.
`final_out`	The output of the last process applied to the object.
Methods	
`set(process_name, process_output)`	Assign `process_output` to be the output returned by the given process.
`get(process_name)`	Return the output returned by the given process, or None if such a process has not been executed yet.
`get_initial_input()`	Return the initial input.
`get_final_out()`	Return the output of the last process applied to the object.
`check_requirement(process_name)`	Return whether the given process has been applied to the object.

Class	NinoPipeline
Applies a sequence of modules to an object.	
Attributes	
`nino_obj`: NinoObject	The object to be processed in a pipeline.
`modules` : NinoModule[]	The modules to be applied to the object in order.
Methods	
`run()`	Apply each module in the sequence to the object.
`check_requirements(process_name)`	Return whether the given process has been applied to the object.

Class	NinoUtils
Utility functions for NinoObject and NinoPipeline.	
Attributes	
`module_names`	List of module names in the order they are to be applied.
`class_references`	Dictionary mapping each module name to an reference to the corresponding module.
Methods	
`load_modules()`	Initialize `class_references`.
`get_class_references()`	Return `class_references`.
`request_class_references(module_names)`	Return the class references for the given list of module names.
`request_default_objects(module_names)`	Return a list of default instantiations for the given list of module names.

3.2.3 Modules

Class	Preprocessor : NinoModule
A module processing note objects.	
Attributes	
`name`	Has constant value `"Preprocessor"`.
`requirements_list`	Has constant value `[]`.
Methods	
`apply_module(nino_obj)`	Process the given note object and record the output in the object.

Class	LayoutAnalysis : NinoModule
Segments images into regions.	
Attributes	
`name`	Has constant value `"LayoutAnalysis"`.
`requirements_list`	Has constant value `["Preprocessor"]`.
Methods	
`apply_module(nino_obj)`	Receive the initial image of the note object, segment the image into regions, and record as output a tree of bounding boxes for each region.

Class	TextRecognizer : NinoModule
Recognizes text in text regions.	
Attributes	
`name`	Has constant value `"TextRecognizer"`.
`requirements_list`	Has constant value `["LayoutAnalysis"]`.
`model`	Neural network model to process images and convert them into text.
Methods	
`apply_module(nino_obj)`	Receive and traverse the segmented bounding box tree of the object, visit each text region and annotate it with the text recognized in it, and record as output the annotated tree.

Class	MathExpRecognizer : NinoModule
Recognizes mathematical expressions in math regions.	
Attributes	
`name`	Has constant value `"MathExpRecognizer"`.
`requirements_list`	Has constant value `["LayoutAnalysis"]`.
`model`	Neural network model to process images and convert them into mathematical expression trees.
Methods	
`apply_module(nino_obj)`	Receive and traverse the segmented bounding box tree of the object, visit each math region and annotate it with the mathematical expression recognized in it, and record as output the annotated tree.

Class	FigureAnalyzer : NinoModule
Processes figures in images.	
Attributes	
`name`	Has constant value `"FigureAnalyzer"`.
`requirements_list`	Has constant value `["LayoutAnalysis"]`.
Methods	
`apply_module(nino_obj)`	Receive and traverse the segmented bounding box tree of the object, visit each figure box and process it, and record as output the annotated tree.

Class	NLP : NinoModule
Contains operations related to natural language processing.	
Attributes	
`name`	Has constant value `"NLP"`.
`requirements_list`	Has constant value `["TextRecognizer"]`.
`model`	Neural network model(s) for natural language processing.
Methods	
`apply_module(nino_obj)`	Receive and traverse the segmented bounding box tree of the object, visit each text box and process it, and record as output information such as keywords.

3.2.4 Data Access Objects

Class	User : django.db.models.Model
Contains User data access object(DAO) class.	
Attributes	
`username`	The username of the user.
`email`	The email of the user.
`password`	The hashed password of the user.
`access_group`	The access group of the user that defines the rights to read/write/delete/modify notes/categories/users etc.
`categories`	List of categories belonging to the user.
`notes`	List of notes of the user.
Methods	
`__str__()`	String representation of user object.

Class	Category : django.db.models.Model
Contains category data access object(DAO) class.	
Attributes	
`owner`	The Category object's owner user.
`name`	Name of the Category object.
`notes`	List of notes of the category object.
Methods	
`__str__()`	String representation of category object.

Class	Note : django.db.models.Model
Contains main note data access object(DAO) class.	
Attributes	
`owner`	The Note object's owner user.
`name`	Name of the Note object.
`image`	Raw image path of the note object.
`category`	Category of the note object.
`text`	Recognized text of the note object.
`keywords`	Important keyword list of the note object.
`equations`	List of equations of the note object.
Methods	
`__str__()`	String representation of note object.

Class	Equation : django.db.models.Model
Contains equation data access object(DAO) class.	
Attributes	
`note`	The Note object's owner note.
`image`	Cropped image path of the equation object.
`latex_representation`	Latex representation of equation object.
Methods	
`__str__()`	String representation of equation object.

4 Appendix

Client subsystem's auto-generated Java interfaces:

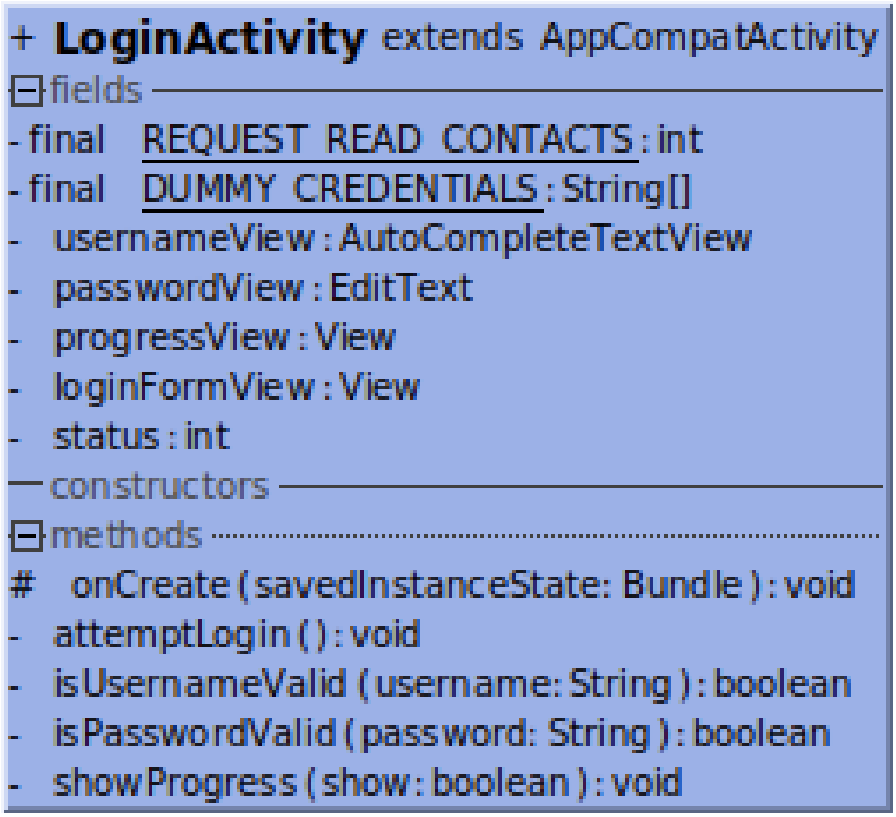

Figure 6: LoginActivity

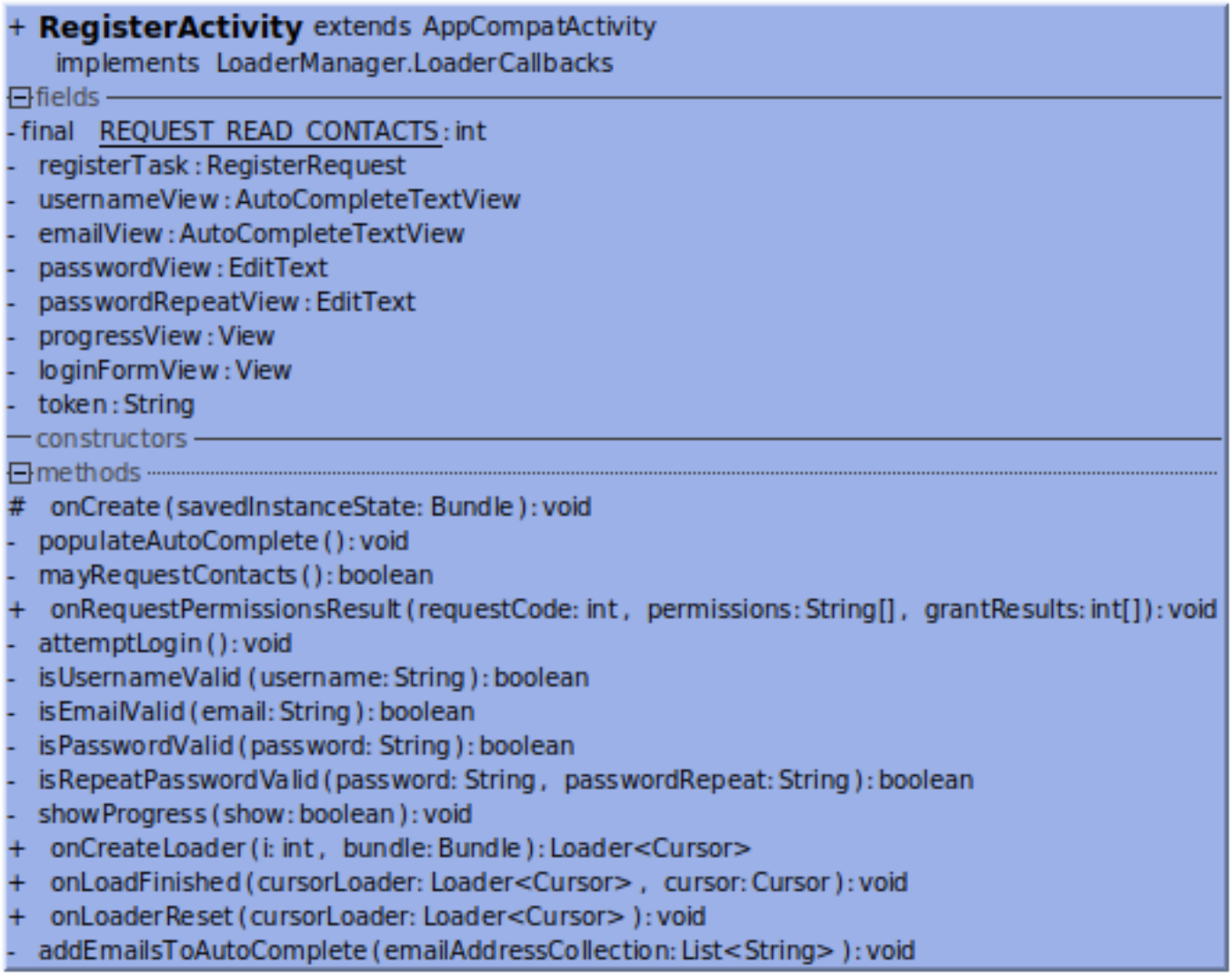

Figure 7: RegisterActivity

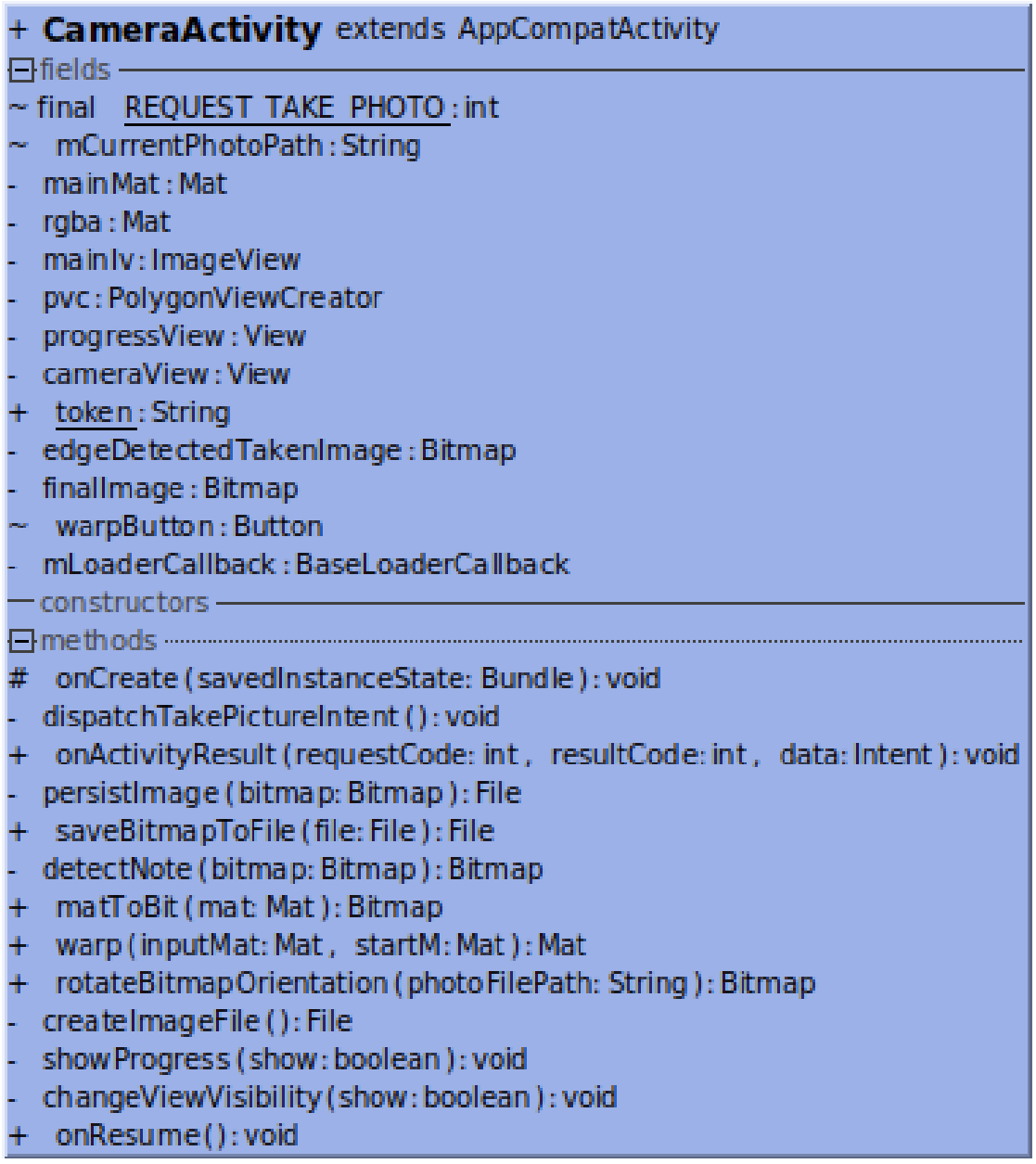

Figure 8: CameraActivity

Figure 9: CameraManager

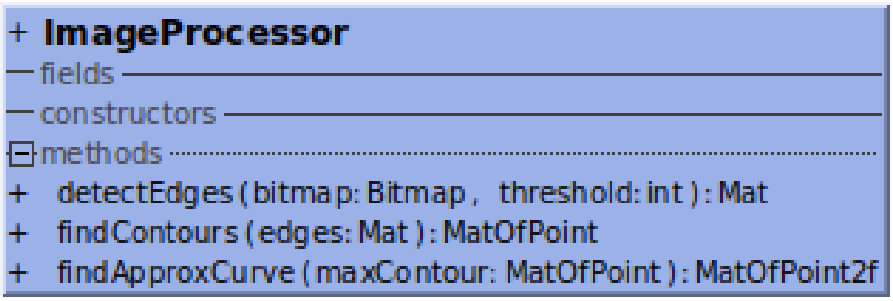

Figure 10: ImageProcessor

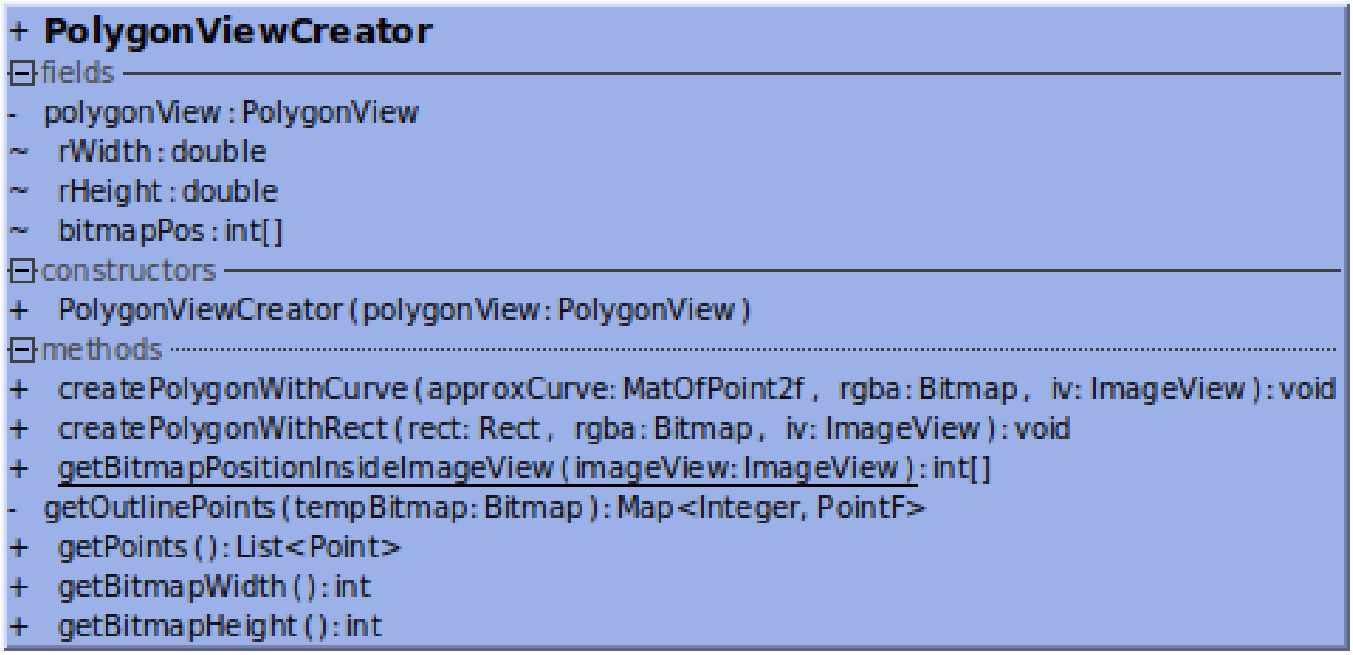

Figure 11: PolygonViewCreator

References

[1] "3 Easy Steps to Digitize Your Study Notes". [Online]. https://www.developgoodhabits.com/digitize-study-notes/. [Accessed Feb 18, 2018].

[2] "How to digitize your handwritten notes". [Online]. https://www.popsci.com/digitize-handwritten-notes. [Accessed Feb 18, 2018].

[3] "How to Take Study Notes: 5 Effective Note Taking Methods". [Online]. https://www.oxfordlearning.com/5-effective-note-taking-methods/. [Accessed Feb 18, 2018].

[4] "The LaTeX Project". [Online]. https://www.latex-project.org/. [Accessed Feb 18, 2018].

[5] "PDF 2.0". [Online]. https://www.iso.org/standard/63534.html. [Accessed Feb 18, 2018].

[6] "The Web framework for perfectionists with deadlines — Django". [Online]. https://www.djangoproject.com/. [Accessed Feb 18, 2018].

[7] "Ubuntu 18.10 (Cosmic Cuttlefish)". [Online]. http://releases.ubuntu.com/18.10/. [Accessed Feb 18, 2018].

[8] "SQLite Home Page". [Online]. https://www.sqlite.org/. [Accessed Feb 18, 2018].

[9] "OAuth 2". [Online]. https://oauth.net/2/. [Accessed Feb 18, 2018].

[10] "Unified Modeling Language". [Online]. http://www.uml.org/. [Accessed Feb 18, 2018]

[11] "IEEE Citation Guidelines". [Online]. https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf. [Accessed Feb 18, 2018].

[12] "AutoCompleteTextView". [Online]. https://developer.android.com/reference/android/widget/AutoCompleteTextView. [Accessed Feb 18, 2018]

[13] "LoaderManager.LoaderCallbacks". [Online].
`https://developer.android.com/reference/android/app/LoaderManager.LoaderCallbacks`.
[Accessed Feb 18, 2018]