Bilkent University

Department of Computer Engineering

Senior Design Project

NINO: NINO Is Not OCR

High-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

High-Level Design Report
Dec 31, 2018

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 Purpose of the system

NINO (NINO Is Not OCR) is a note taking app. The main purpose of the app is to allow the users to take photos of and then convert the handwritten and printed notes into an editable format which can be exported as LaTeX or PDF. NINO aims to recognize and digitize notes with text, mathematical equations, plots, trees, UML diagrams. In addition to that, our app proposes to assign categories to notes in order to assemble similar notes together, provide a quick summary of any given note, recognize and match professional/scientific terms to their Wikipedia link. This system will be constructed by a combination of object oriented design techniques and architecture design.

1.2 Design goals

The main purpose of NINO is to digitize the so called notes. In order to produce a working and expandable system we have decided to include the following properties into our software design:

1.2.1 Extensibility and Modularity

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, so long the module is written. Adding extra modules to the system should be straightforward as long as the module functions correctly by itself.

1.2.2 Maintainability

Since the system is highly modular, subsystems should not be highly coupled, a change or problem in one subsystem should not affect others. This is important to be able to preserve the quality of service as more modules are added.

1.2.3 Reusability

Since the system is highly modular and designed to work as a pipeline, any module of the project can be easily extracted and used in other softwares. Similarly, the pipeline structure also simplifies using existing software if needed within NINO?s system.

1.2.4 Usability

Interactivity is one of the most important features that make Nino special. Users should be able to navigate through the clients without having problems on how to use it, the process should seem natural without a high learning curve. There should be appropriate description to each module so that the user can clearly understand its function and use cases, preferably with examples or tutorials. 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. Users should be able to contact developers to make suggestions or report problems and their experience.

1.2.5 Performance

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. Results of any human interaction (other modules) such as clicks or selections should not take more than 1 second to provide good user experience.

1.2.6 Availability

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.

1.2.7 Scalability

Adding more storage and compute to the server should be easy as long as new storage compute hardware is available. Rise in demand must be addressed quickly to not cause any drop in performance or downtimes in the system. Modules should be able to work on more than one document if available to make better predictions.

1.3 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.

1.4 Overview

NINO (NINO Is Not OCR) is an Android application, targeted primarily for tablets, that can detect text, equations and figures in pictures taken of notes on paper or a whiteboard. The user can take pictures directly within the application or load preexisting pictures from

the device. After the program analyzes the picture, it may present annotations on the picture and store them so that the user can later view and search through them.

The application consists of a client Android and web interface for end users, as well as a backend server application for data storage and processing. In the client application, the user is first presented with a login screen where the user may register or log in. After logging in the user may either add a new picture to their library, whether by camera or from the device, or view previously annotated pictures. After a picture is taken it is sent to the server for processing. The server first segments the image into regions containing text, equations, graphs or other figures, parse the text or equations (in LaTeX) contained in each region, and then process the text and equations in order to provide helpful hyperlinks to the user. For example, the program may link a variable or concept to where it was defined in the notes, or to an online resource such as the Wikipedia page for the concept which the user would be able to open without leaving the application. The program can also process the text in the document to summarize its content and tag it with keywords in order to enable more comprehensive searches.

After processing, the server stores the annotated picture in the cloud storage reserved for the user and sends it back to the client application. The client application then displays the notes overlaid on the image in an interactive interface where the user may click on links, rearrange or modify the notes as desired. The user may give positive or negative feedback on the performance of the recognition of text and equations, and opt to make corrections and send them to improve the accuracy of the program. The user may also share the annotated picture either directly as an image or as a LaTeX [4] or PDF [5] file. The notes the user has uploaded are collected within the personal storage space allotted to the user, so that the user can view and modify them similarly, search through them, and categorize them e.g. with respect to course title and subject so as to assist the program in improving its tagging and summarization of notes.

2 Proposed software architecture

2.1 Overview

Software architecture gives detailed information about the structure of NINO?s system, bringing together the classes explained in the analysis report in an orderly way. In the following subsections, the interactions between subsystems, technical decisions made so far and the classes/modules planned to be deployed are discussed.

2.2 Subsystem decomposition

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.

The client side is separated into two main subsystems: the presentation subsystem, handling the graphical user interface that the user interacts, and the controller subsystem which controls interactions with external systems, such as the camera, local storage and the server. The presentation is itself broken into different modules for authentication, viewing and editing notes and viewing the user profile. When the user makes a request to authenticate or to create or open a note, this call is sent to the controller which itself communicates with the appropriate external system. For example, if the user attempts to log in, the presentation communicates this with the controller, which sends a request to the server, waits for a response (either success or failure), and communicates the response back to the presentation. Likewise if the user chooses to create a new note, the controller first invokes the device camera and calls the presentation to change to the appropriate interface, then after receiving an image from the camera, the controller sends a request to the server to create a note from the image, and after getting a response invokes the presentation to display the note.

The server side is also separated into four main subsystems: request handling, data

management, user management and note processing. Request handling includes communications with each client, maintaining and scheduling active requests made by clients, sending the request to the respective module and returning the response to the request to the appropriate client. For example, if a client sends a request to convert an image into a note, the request handler first invokes user management to verify credentials, then data management to save the image under the storage reserved for the user, then note processing to process the image to create a note, and finally sends the note back to the client in response. The subsystem for data management holds for each user the storage space for notes reserved for them, as well as authentication data and shared data used in note processing, such as model parameters, training datasets and other configuration data. 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 subsystem for note processing is itself broken down into several submodules, such as 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. Each module may also depend on the outputs of other modules, e.g. text and equation recognition must come after segmentation and text analysis after text recognition, and the execution order of the modules in the pipeline must respect these dependencies.

2.3 Hardware/software mapping

Our server side modules will be implemented in Python. For database management Django [6] is used.

The client and the server communicate through a REST (REpresentational State Transfer) API. HTTP requests will be made by the client depending on user actions and the server will return with an appropriate response to constitute a communication. The client can be run on smartphones or tablets with the Android operating system, as well as through a web client. Both clients can configure already created notes or create notes using existing pictures, but only Android applications can take pictures via smartphone and tablet camera to create new notes. Simply, the client and server nodes communicate via the HTTP protocol as seen in the figure below. The details of nodes can be found in subsystems.

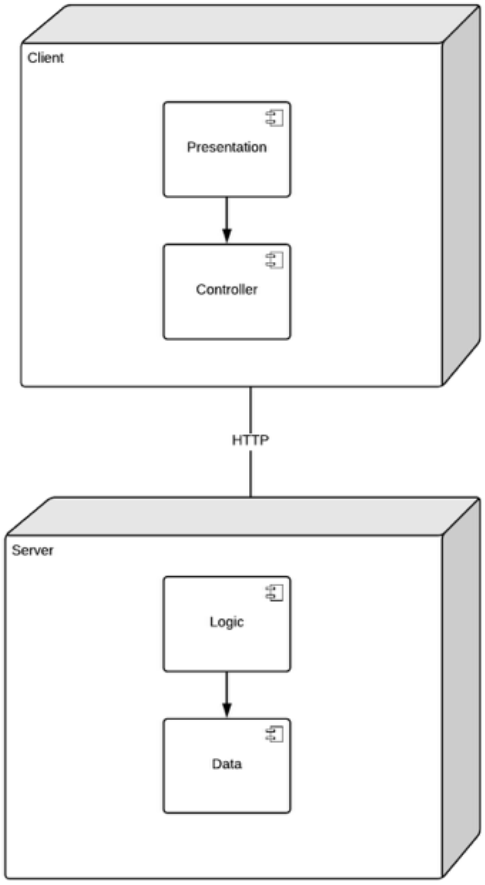

Figure 1: Component diagram of the HTTP communications between subsystems.

2.4 Persistent data management

Persistent data includes both the notes taken by the user and the information kept within the server. When a user is done making changes and saves a note, the note is saved remotely to the server as well as locally if the note is created from an android device. The notes are saved as jpeg images. Notes stay on the server as long as their total size do not exceed the limit provided by the server.

The server also keeps information about the users and notes in a Django database. These information are used for services such as authentication. Readily trained models are also kept within server side and used when a module that uses them are invoked.

The server program will be built for Ubuntu 18.10 [7] so that it can use the local storage to store images related to notes, model data, and model parameters. To store notes, authentication data and other data related to users, a database system is used. As our database system, we have chosen SQLite [8] which is a small, fast, self-contained, highly reliable SQL database engine.

2.5 Access control and security

Users must create an account to use Nino. Each user is required to select a username and password for authentication services, while providing an email is optional. A user account can only see or edit notes that were created by it, i.e. each user has its own note space and cannot access the note space of others. The passwords are kept as hashes in the database and the authentication with the hash created on the client side. The server never acquires the password as plaintext.

Our authentication system uses Oauth (Open Authorization) technology [9]. It is standard for token-based authentication and authorization on the Internet. It allows our REST API to be used without exposing the end user?s password to the phone or network. Relatedly, many REST API endpoints require authentication. The token required for authentication is obtained from the earlier request to an endpoint of server via username and hashed password. The client accesses related API endpoints by a token.

2.6 Global software control

Nino uses an event-driven software control. When creating notes, user selections cause requests to made to the server. The server handles the requests and returns the processed result to the client. This event driven control is also made on other aspects, such as authentication where a login request is created with the user typed username and password and an authentication token is returned if the credentials are correct. After authentication is made, the server is asked to list the notes the user has, yet again by a request. Another example would be the information update request handled by the server whenever the user updates a preference.

2.7 Boundary conditions

2.7.1 Initialization

Users need to access the Nino client to use it. Client can be used through smartphones or tablets with Android operating system; as well as through the web with a supported web browser. Following procedures are the same for both web and Android clients. The user first needs to be authenticated using account username and password. If the user does not have an account, it must be created (registered) using the client. Account creation can be initiated by the related button on the login screen. The user is reminded of the required format for username and password both in login and registration (e.g. the minimum password length). If the login credentials do not meet what was expected and authentication fails, the user is returned back to the login screen. Since the authentication takes place on Nino?s server, internet connection is required at this stage. When the user logs in to Nino, he/she is presented with the note screen. The authentication token is saved for 30 minutes before deletion, so the user can access the account again within that interval without authenticating again.

2.7.2 Termination

The user can log out of Nino at any desired time, except when the client is waiting for the result of an operation on a note from the server. Unless the authentication token becomes

invalid (i.e. 30 minutes have passed since last authentication), the user is not logged out of the client when the application is terminated on Android system.

2.7.3 Failure

In case of a crash in the client, related client data is tried to be saved before termination. Since internet connection is required to use most of the features of Nino, lack of internet connection can cause failures.

3 Subsystem services

Nino is formed by the interaction of two systems: the client and the server.

3.1 Client

The client can be used through smartphones or tablets with Android operating system; as well as through the web with a supported web browser. 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.

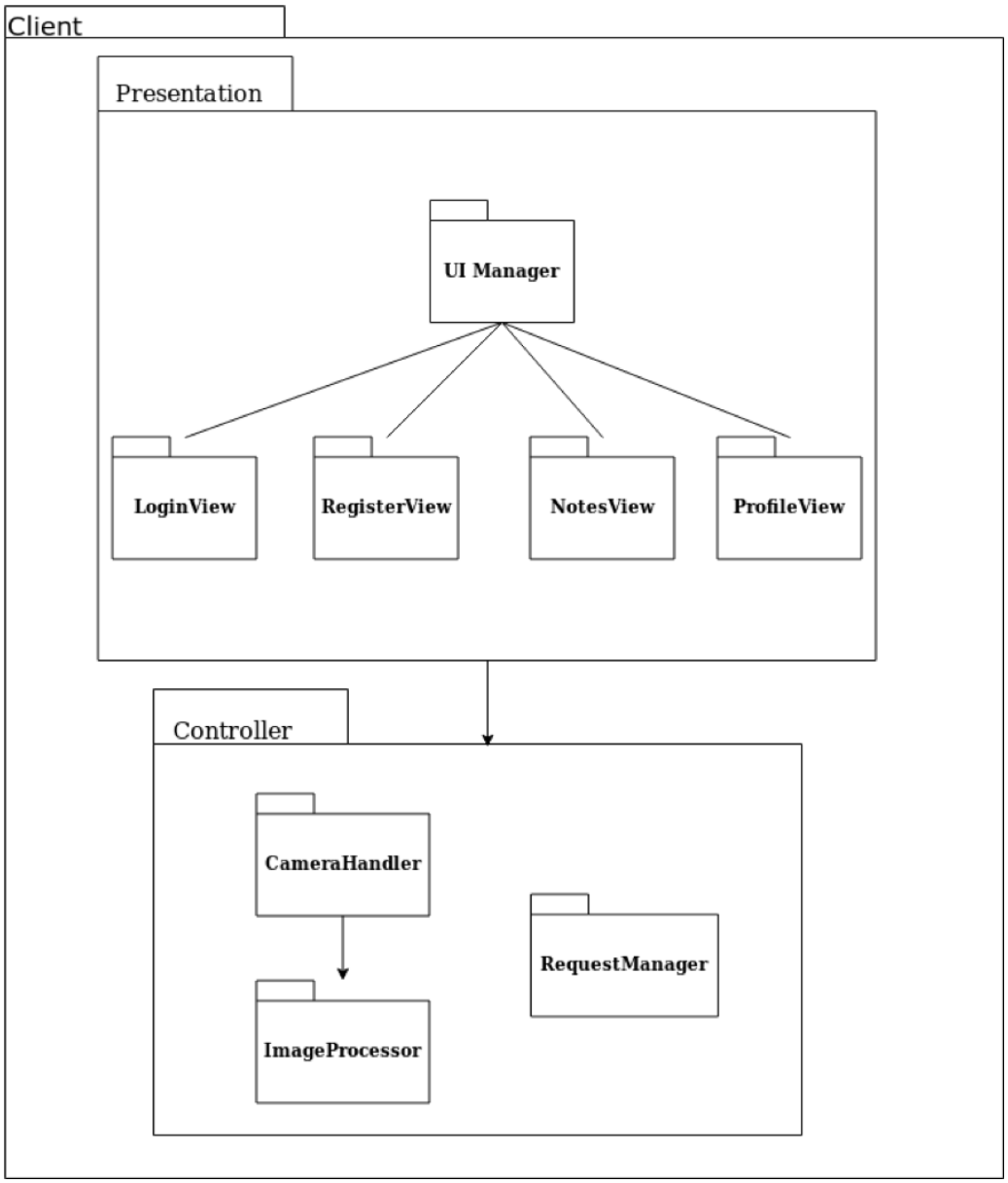

Figure 2: The hierarchy of modules inside the client system.

3.1.1 Presentation

The presentation layer contains views and their event firing mechanisms.

UIManager: A global manager to mediate other view classes.

LoginView: The view class for the login screen. Checks credential formats before asking for request creation. Uses RequestManager to make login requests.

RegisterView: The view class for the register screen. Checks credential formats before asking for request creation. Uses RequestManager to make register requests.

NotesView: 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 camera to take pictures and initiate note creation. Uses RequestManager to list available notes and CameraHandler to take pictures.

ProfileView: The view class for the profile screen. Uses RequestManager to get user info.

3.1.2 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.

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

ImageProcessor: Handles morphological operations, edge detection, image enhancement and warping to the images.

RequestManager: Creates all the requests depending on the events fired by the presentation layer and communicates with the server. Waits for the server response and notifies views of the result.

3.2 Server

The server is where all the processing happens. 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 Logic layer accepts the requests and updates/creates/removes data. The logic layer governs the database operations and the note processing pipeline. The data layer is where the models and other persistent information are kept.

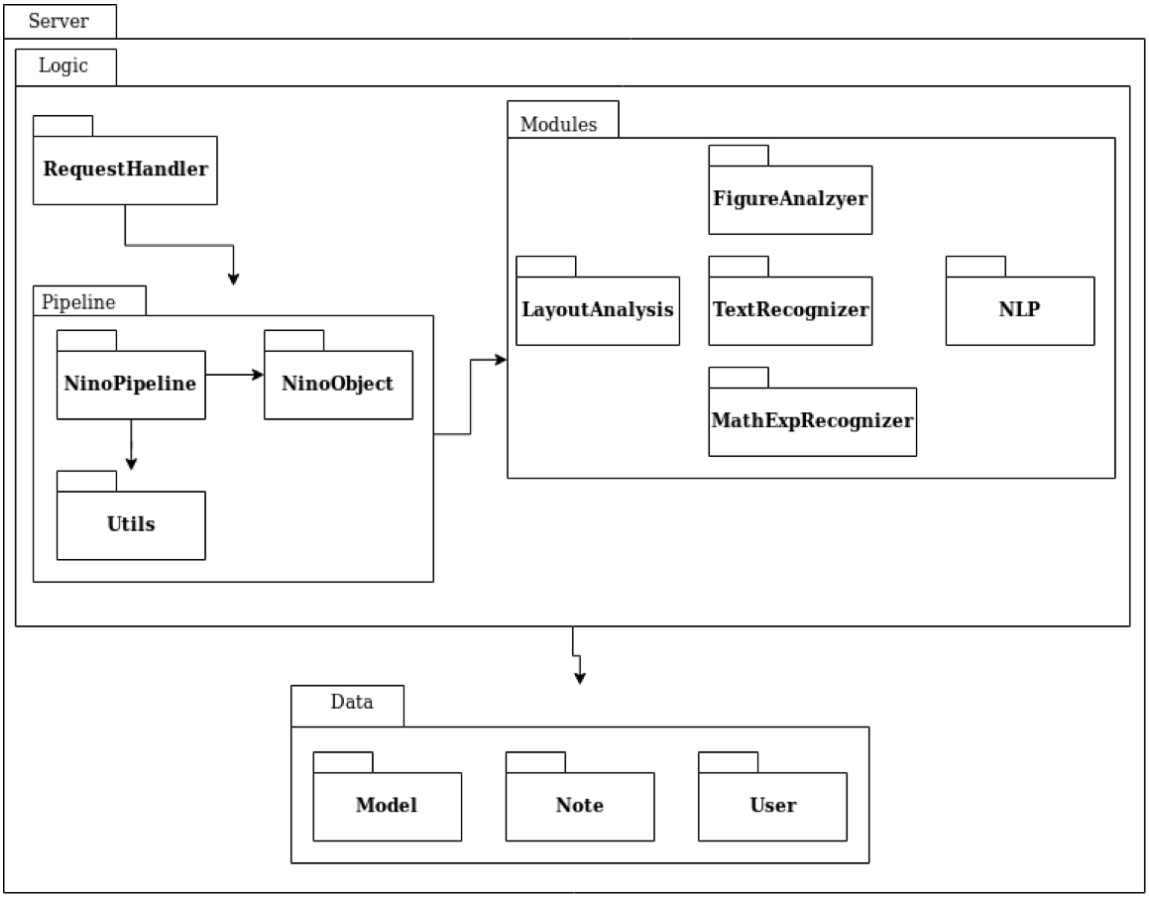

Figure 3: The hierarchy of layers and submodules within the server.

3.2.1 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.

3.2.2 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.

3.2.3 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.

3.2.4 Data

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

Model: Parameters for models used in note processing modules.

User: User information, such as username, that is kept in a django database.

Note: The notes created by the users.

References

[1] "3 Easy Steps to Digitize Your Study Notes". https://www.developgoodhabits.com/digitize-study-notes/. [Accessed Dec 31, 2018].

[2] "How to digitize your handwritten notes". https://www.popsci.com/digitize-handwritten-notes. [Accessed Dec 31, 2018].

[3] "How to Take Study Notes: 5 Effective Note Taking Methods". https://www.oxfordlearning.com/5-effective-note-taking-methods/. [Accessed Dec 31, 2018].

[4] "The LaTeX Project". https://www.latex-project.org/. [Accessed Dec 31, 2018].

[5] "PDF 2.0". https://www.iso.org/standard/63534.html. [Accessed Dec 31, 2018].

[6] "The Web framework for perfectionists with deadlines — Django". https://www.djangoproject.com/. [Accessed Dec 31, 2018].

[7] "Ubuntu 18.10 (Cosmic Cuttlefish)". http://releases.ubuntu.com/18.10/. [Accessed Dec 31, 2018].

[8] "SQLite Home Page". https://www.sqlite.org/. [Accessed Dec 31, 2018].

[9] "OAuth 2". https://oauth.net/2/. [Accessed Dec 31, 2018].