



VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
DEPARTMENT OF COMPUTER SCIENCE II

Problem-based Project

Todoist Extention System ClapDoist

Done by:

Rokas Alechnavičius

Tomas Banyš

Benas Urbonas

Supervisor:

Lecturer Linas Būtėnas

Vilnius
2018

Contents

Abstract	3
Santrauka	4
Introduction	5
1 Analysis	6
1.1 Functional Requirements	6
1.2 Similar Systems on the Internet	7
1.3 Analysis on the Most Popular Project Managing Systems	8
1.3.1 Microsoft Project	8
1.3.2 Slack	9
1.3.3 Todoist	11
2 Structure, Data and Todoist API	13
3 Design	18
3.1 Database Design of the System	18
3.2 Design of the Website	20
3.2.1 Dashboard Design	22
3.2.2 Plotly Framework, Charts and Graphs in ClapDoist	24
3.2.3 Calendar	28
4 Implementation	29
4.1 Developing Tools	29
4.2 Website Implementation	29
4.2.1 Synchronization, Backup implementation	30
4.2.2 Table Data Implementation Throughout the Website	31
4.2.3 Calendar Implementation	33
4.2.4 Graphs implementation algorithms	34
4.3 Configuration of Virtual Machine, Apache, Custom Django Commands	36
4.4 Heuristic Evaluation of the Website	38
Conclusions and Recommendations	39

Abstract

To make Todoist a more appealing system for project managers that have more than a few different projects on their hands, innovative technologies such as websites could be used. In this report a website for extending Todoist is presented. It is made specifically for extending Todoist using its application programming interface and it could not be easily modified and adapted for other project management systems. The website is useful for a project manager because it presents features such as data analysis from a certain amount of time, backups of previous data versions in case some unwanted changes happen, other useful information. In order to make the website stable and trusty, a web framework Django was used. To store and use data for analysis, PostgreSQL database is used. Website uses a calendar in order to make viewing and analyzing projects a simple task to do for the project managers. In the report all features and their explanations are presented in detail in order to explain website and its full functionality in addition to explaining how are the operations happening in the back-end.

Santrauka

Todoist projektų valdymo sistemos plėtinys

Ši ataskaita yra skirta pristatyti Internetinį tinklapį, skirtą praplėsti Todoist projektų valdymo sistemos funkcionalumą. Tam, kad tikslas būtų įgyvendintas, reikia realizuoti serverį su duomenų baze, kuri gautų bei laikytų duomenis, kuriuos atsiunčiame prisijungusiam vartotojui naudojantis Todoist sistemos programavimo sąsajos pagalba. Šalia to, reikia sukurti malonaus dizaino tačiau profesionalų dizainą tinklapyje, kad projektų vadovas nepasigestų funkcionalumo ir neturėtų var-tyti akių ieškodamas kur yra tai, ko jam reikia.

Internetinis tinklapis, praplėčiantis Todoist projektų valdymo sistemos funkcionalumą yra skir-tas projektų vadovams, kuriems nepakanka Todoist sistemoje esančio funkcionalumo ir yra didžiaja dalimi skirtas duomenų analizei ir jos įvairiems atvaizdavimo būdam. Tinklapis, sukurtas Django pagrindu, sugeba prisijungusiam vartotojui parsiųsti esamus jo duomenis naudojantis jo perduotu sistemos programavimo sąsajos raktu, šiuos duomenis išsaugoti bei atvaizduoti. Taip pat einant laikui saugoti naujus bei senus duomenis, atlikti jų analizę ir parodyti įvairius analizės atvaiz-davimo būdus. Tinklapis siūlo funkcijas nuo paprasčiausio projektų bei užduočių atvaizdavimo interaktyvioje lentelėje, iki kalendoriaus, analizės atvaizdavimo ir įvairių pokyčių fiksavimo ir pavaizdavimo. Sistema taip pat leidžia vartotojui pasirinkti, ar jis nori, jog jo duomenys būtų kau-piami iki tam tikros nurodytos ribos.

Visa informacija tinklapyje pateikiama anglų kalba. Nuspręsta naudoti anglų kalbą dėl to, kad Todoist sistemos vartotojai yra iš įvairių šalių, todėl paprasčiausias ir daugiausiai vartotojų priimti-nas variantas yra sistema, kuri veikia anglų kalba. Visi kintantys duomenys(Projektai, užduotys) yra saugomi duomenų bazėje. Duomenų bazė yra saugoma šalia serverio ir yra pasiekama naudo-janti Django sistemos programavimo sąsaja, kuri palengvina tiesioginius kreipimusis į duomenų bazę norint paimti ar atnaujinti duomenis.

Tinklapij palaiko Apache serveris, kuris lengvai bendrauja su Django karkasu ir padeda sukurti greitai veikiančią sistemą. Tinklapis yra pritaikytas daugumai kompiuterių bei Interneto naršyklių. Visi pagrindiniai duomenys yra saugomi mūsų duomenų bazėje bei Todoist duomenų bazėje.

Tinklapijo privalumas yra jos naudingumas. Kadangi sistema nori pratęsti Todoist sistemą, ji neturi didžiosios dalies funkcionalumo kurį turi pati Todoist sistema ir gali vietoje to susitelkti į naujus bei inovatyvius sprendimus kaip duomenų analizė. Kitas sistemos privalumas yra tas, jog šią sistemą galima visuomenės plėsti, nes yra be galo daug įvairių būdų ir papildomų tyrimų, kuriuos galima atlikti su išanalizuotais duomenimis.

Introduction

Project management systems have always been the go-to for project managers because of the functionality and simplicity that they offer in regards to controlling projects and their work flow. One of the most popular systems nowadays is the Todoist project management system, which appeals to more than a few million users each day . Well structured, nicely administrated and easy to use, this system is surely one of the best choices when it comes to project management. However, it is missing some important features that feel like they could be added in order to make Todoist feel more whole and complete for the project manager. This, however, is not yet implemented, therefore it should be modernized enough to keep people interest in using the system.

Nowadays most people work on their computer rather than other platforms. The same goes for managing projects, which they do on their laptops or regular computers, tablets etc and this is why a Todoist extension which is website based can be a popular choice to use in order to have extended Todoist functionality. First of all, it would be on the same platform as Todoist itself, therefore it would be comfortable to use. In addition to that, more people own computers at work/home than other software so making a website based system is a simple and very convenient choice. All these features can attract more users to use the extension in order to receive additional functionality for their project management system needs.

Therefore, the main goal of the project is to create a website which should offer various functionality which Todoist itself does not have such as data analysis, data backups, various additional information and calculations based on the projects and tasks which the user is responsible for. This is done by using the users Todoist API key and syncing their Todoist data with our website. Then, the data is analyzed with its previous versions if possible and different results are displayed for the user. Every project manager using the system should get their fill of the information about the projects that they are managing. The same goes for the tasks in the project and people that are collaborating in the project and are responsible for the tasks. Also, to make our system more attractive, slick and stylish interfaces should be created that allow simple yet effective work with the functionality of the website. Also, managers should be able to choose different analysis and ways of display for their data, according to their own personal choices and preferences. Every page and different ways of display will be accessed either from the main window, or from the navigation bars that are always displayed on the top of each web page. In addition to that, a universal display for projects and tasks will be in a responsive table or a calendar which allows more interactive approach to seeing ones work flow.

To achieve this goal and create a fully functioning and flexible website some additional parts are necessary. First of all, a stable connection and data transfer between our system and Todoist API must be created. In order to save the various and continuos data, a strong and safe database must be used in addition to making sure that data is constantly updated and saved without any errors. In addition to that, the data needs to be backed up in case of a rollback request or a system fault. Next, a framework should be used that allows administrator to see the flow of system itself and gives him the rights to change the data remotely if needed. lastly, the website should be created with strong stability, error and action logging, pleasing design.

1 Analysis

1.1 Functional Requirements

Before analyzing other project management systems, it is important to define minimal and expected, extra functional requirements in order to understand why is our analysis based on those specific systems rather than other ones. First off, minimal requirements:

- A sturdy website that can act as a Todoist extension.
- As a Todoist extension, it needs to be able to take and use information about projects and tasks.
- One of the usages is a table of all your projects and tasks, that could be filtered by various parameters such as overdue tasks or tasks assigned for a specific person.
- Another use are graphs that provide information about projects and tasks which Todoist does not provide itself, meaning that graphs are drawn after some sort of data analysis is applied(for e.g differences in your project(s) among syncs).
- Third use is a calendar, where tasks with deadlines should be displayed. the user then should be able to acquire additional information about the tasks in some way, the calendar should also be filterable.

These minimal requirements mean that the project becomes presentable, meaning that it has an actual use, rather than being a block of code that does not do anything Todoist itself would not provide. Moving on to next type of requirements:

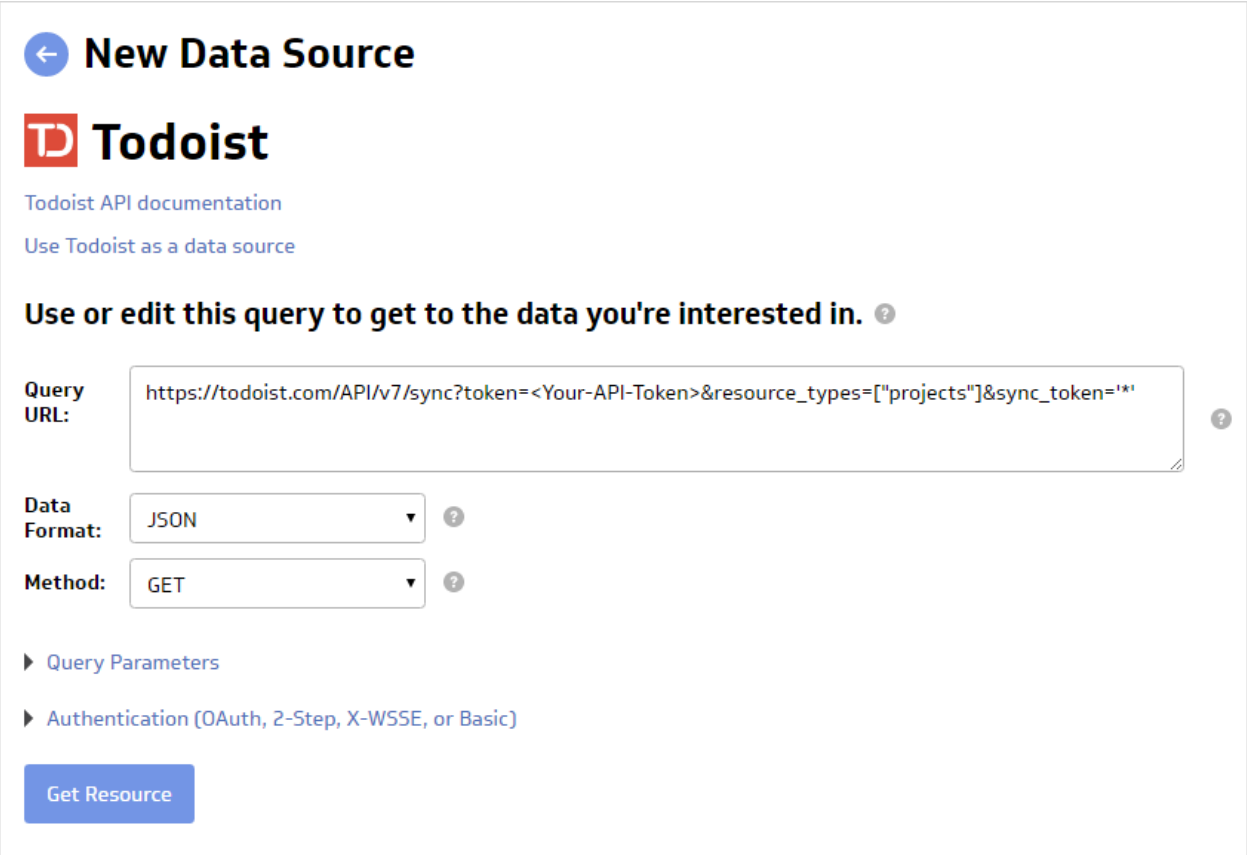
- All design should be simple yet elegant, meaning that after all minimal functionality is implemented, it is expected that the website should feel and be nice to use in order to make user experience more enjoyable.
- The user should be able to customize the way he uses our system. Meaning that he can define whether he wants to use data analysis and keep up old data, etc.

Expected requirements, as can be seen, mostly focus on the HCI part of the course and is mostly about pleasant user experience. Lastly, the extra functionality is as follows:

- A backup system should be implemented, meaning that user could be able to choose one of his old syncs, and that data would then be sent to Todoist and synced with it, turning the project back into its old state.
- Most of the important data in the database should be encrypted in order to prevent site admins from doing anything troublesome.

1.2 Similar Systems on the Internet

As far as the data analysis and display of it goes, only one similar system was found that uses Todoist data using its API and displays various things according to the data that they receive. This system is called Klipfolio [1] and it allows to build a Todoist dashboard to track your individual and team projects. It allows for management and measurement of your efficiency as the dashboard is real-time. As far as the functionality goes, the Klipfolio system is not too much special or amazing. However, there is one thing that makes the system more appealing than our system and that is the custom queries that the user can write to Todoist API in order to receive not all of its data, but a specific type and amount of data. the Figure 1 shows the query creation page for Todoist API using Klipfolio system. As seen, the user needs to add a URL with their API token, choose which data format he wishes to receive and choose various other parameters in order to define the exact data that he wants to receive from the application programming interface.



New Data Source

TD Todoist

[Todoist API documentation](#)

[Use Todoist as a data source](#)

Use or edit this query to get to the data you're interested in. ?

Query URL:

Data Format: ?

Method: ?

► [Query Parameters](#)

► [Authentication \(OAuth, 2-Step, X-WSSE, or Basic\)](#)

[Get Resource](#)

Figure 1. Klipfolio [1] query creation page

The second system, however, does not have anything related to Todoist itself. Instead of that, Sisense system focuses on allowing users to unify their data into visual dashboards, turn the data into insights and share it among their colleagues, business partners. Even though this system also focuses on grouping up data and visualizing it, it looks and works very differently to ClapDoist. The strong parts of Sisense are their stunning visuals which combined create a wonderful dashboard that would amaze almost anyone in addition to analysis functionality which it provides. A nicely created dashboard example of Sisense can be seen in figure 2 .

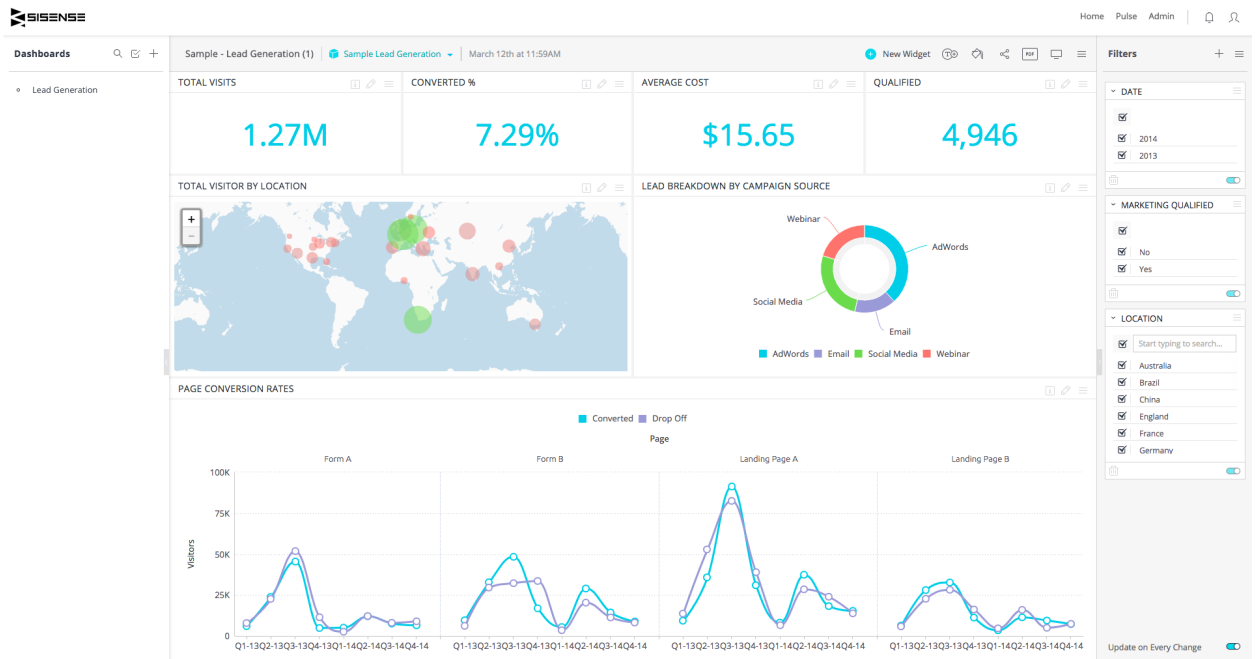


Figure 2. Exemplary Sisense dashboard

1.3 Analysis on the Most Popular Project Managing Systems

1.3.1 Microsoft Project

Starting off the project management systems we have the Microsoft Project [2]. It is a project management software, meaning it has the capacity to help plan, organize and manage resource tools and develop resource estimates. This software is developed by Microsoft and it is designed to assist a project manager in developing a plan, assigning resources to tasks, tracking project and task progress, managing the budget and analyzing workloads. Talking about the functionality, Microsoft Project creates budgets based on assignment work and resource rates. When the user assigns resources to tasks and the assignments, the program helps him by calculating the cost, equal to the work times the rate, which is rolled up to the task level and then to any tasks above it and to the project in the end. Resources(meaning both materials, money and people as work power) can be shared between all projects using a resource pool that is given. Each resource has its own calendar(for example a person resource calendar defines when is the person free to be assigned to a task and when he is already on a task and cannot be assigned another one). In order to reduce the amount of work and calculations for the user, resource rates are used to calculate total assignment costs which are rolled up and summarized. However, all resources can be defined in label without a limit, making it unable to determine how many finished products can be produced with a given

amount of 'materials'. The application creates "critical path" schedules, which constructs a model of the project that includes:

1. A list of all activities required to complete the project.
2. Duration that each activity will take to complete.
3. The dependancies between the activities.
4. Logical end points such as milestones.

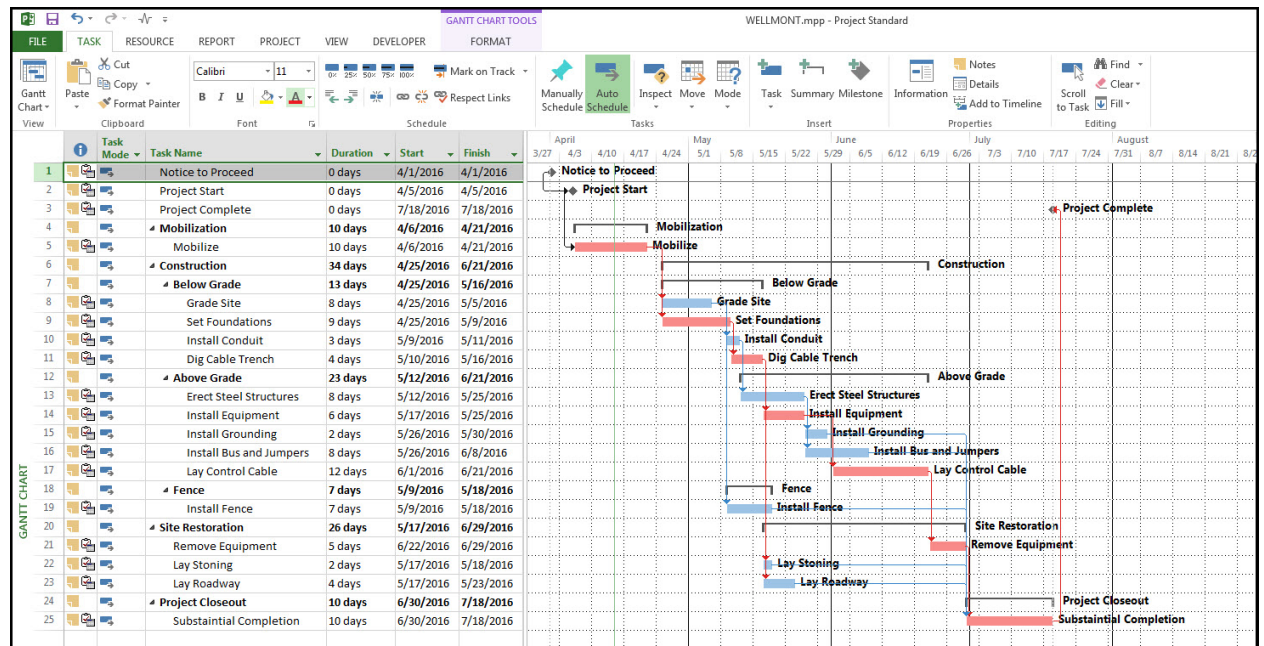


Figure 3. A List of project activities on the left side and Gantt chart representation of the same activities on the right

These schedules are visualized in a Gantt chart. A Gantt chart is a type of bar chart that illustrates a project schedule as well as show dependency relationships between activities and current schedule status. The Figure 3 Shows all the different tasks of a project on the left side of the figure, which is represented by a list. Meanwhile on the right side of the figure it is seen that a Gantt chart representation is shown. Dependency lines can be seen between different tasks, showing that the one must be completed in order for the other one to be able to be started.

While looking at Microsoft Project, the team has decided to take the idea of representing various data by charts and therefore a lot of information that is shown will be seen in different pie and bar charts to make it more intuitive and understandable for the project manager. In addition to that, an effort will be made to implement a more simple version of the Microsoft Project Gantt chart in our Todoist extension system in order for the project manager to visually see the tree of all his projects, the tasks that they contain, dependencies etc.

1.3.2 Slack

Slack [3] software is a much, much different story than Microsoft Project. At a first glance, Slack offers various IRC-like features such as persistent chat rooms, private groups and direct

messaging. All the content within Slack is searchable, including files, people, messages. Slack integrated with a lot of third-party services and supports community-built integrations. To talk more broadly about Slack, it is important to list what makes it so popular and good and list the reasons why we chose to analyze it.

- First of all, Slack brings the communication together, meaning that everyone has a shared workspace where conversations are organized and accessible. This means that Slack channels give everyone a way to organize all those conversations and make sure that people involved in a task are there when needed. Additionally, making a channel for a task or a project means that everyone has a knowledge and a view of what is going on in the project all the time.
- Next, Slack makes it so a person does not have to attend every meeting in order to stay informed. As Slack has a searchable archive of your team's conversations, the knowledge about the decisions are known for everyone involved. In addition to that, files that are shared are also indexed, so the user can easily search within documents, Google Drive files etc to find the relevant information as fast as possible.
- Also, Slack integrates with a lot of enterprise and productivity product available, and allows you to build your own with their open Application programming interface which helps keep your team coordinated and working faster.

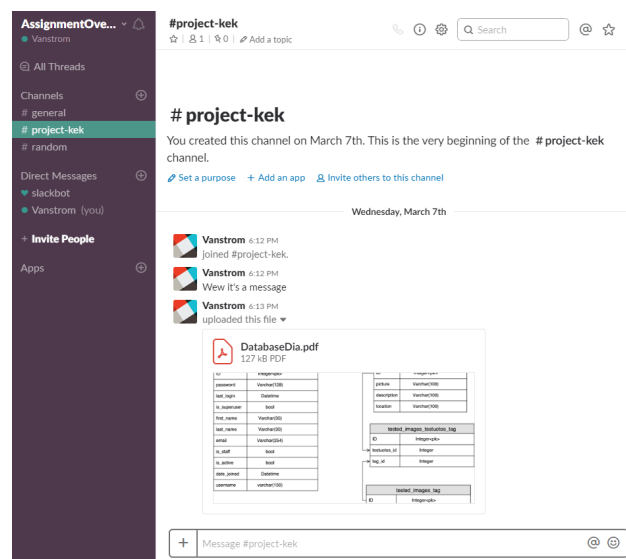


Figure 4. A Slack [3] project chat page, where various channels can be seen

Slack has a slick and simple design that makes understanding things very simple and basic. As it can be seen in the figure 4 There are various channels that can be created for a project or a task. People can be added to specific channels and they can easily discuss or share information, files related to their topic. Important files and messages can be pinned, third party apps that simplify or increase functionality can also be added.

After analyzing Slack the team decided to hold off implementing something similar to this software as well as using its API in order to make a similar subprogram within our system. The idea of making something similar to Slack in order to help project managers communicate with

simple users responsible for tasks are still on our minds, however, it has been pushed as the last priority if the system is finished earlier than expected, because the team feels like this is not a must to-do at the current stage.

1.3.3 Todoist

Todoist [4] is a cloud-based task management system for personal and professional productivity. Users can manage their tasks and projects across many different platforms such as Windows, Android, iOS etc. There are three different Todoist versions: Todoist Free, Todoist Premium and Todoist Business. The paid versions have enhanced collaborative aspects as well as additional productivity features. Users can have many projects and tasks, which are discussed more in depth in the Todoist API part of the report later on. The main idea of this analysis part is to list the functionality which Todoist already implements and mark the points that the team feels should be added in order for the Todoist management system to feel more complete.

Talking about the strong parts of Todoist, there are a few things that must be mentioned. First of all, the speed. One of the Todoist's selling points is its speed. It feels like using a native app. It is done by making the interface slightly responsive while all of the data transfer asynchronously. You do not even have to wait for data to be passed from your web client to the Todoist server and vice versa. The same goes for updating, deleting and creating tasks. Even the searching and sorting is done client side, making filtering nice and fast.

Next is the usage of HTML5's local storage API. This allows the app to go offline if the Todoist server is not available and use local storage in order to create and modify your data. When the server comes back, all of your data is pushed back to the cloud and synced. This is a nice feature for a project management application because people tend to use their application day-in and day-out, therefore an application which is available whether the server is up or not is a very amazing thing to have.

Lastly, Todoist also has its mobile support. The idea is to have one app that supports a web browser in addition to a mobile browser in order to ensure that your 'to-dos' are available anywhere. However, as it is a web app rather than a native app, it is a bit less responsive and slower than a real native app would be.

Talking about the functionality itself, Todoist offers some nice basic things such as creating projects, sub-projects, indenting them. Next come the tasks and sub-tasks which are related to the projects. In addition to that, tasks can be assigned to users, making them collaborators in a certain project. Moving on, projects and tasks can have deadlines and Todoist gives a simple and nice overview of everything that is happening in the system. The figure 5 shows the main page of your Todoist. The user can see his overdue tasks, the tasks that have deadlines today or the other chosen day. On the left, all projects that he is involved with are seen and they can be clicked to see the sub-projects, tasks. It is a simple yet functional interface where user can control all the work flow.

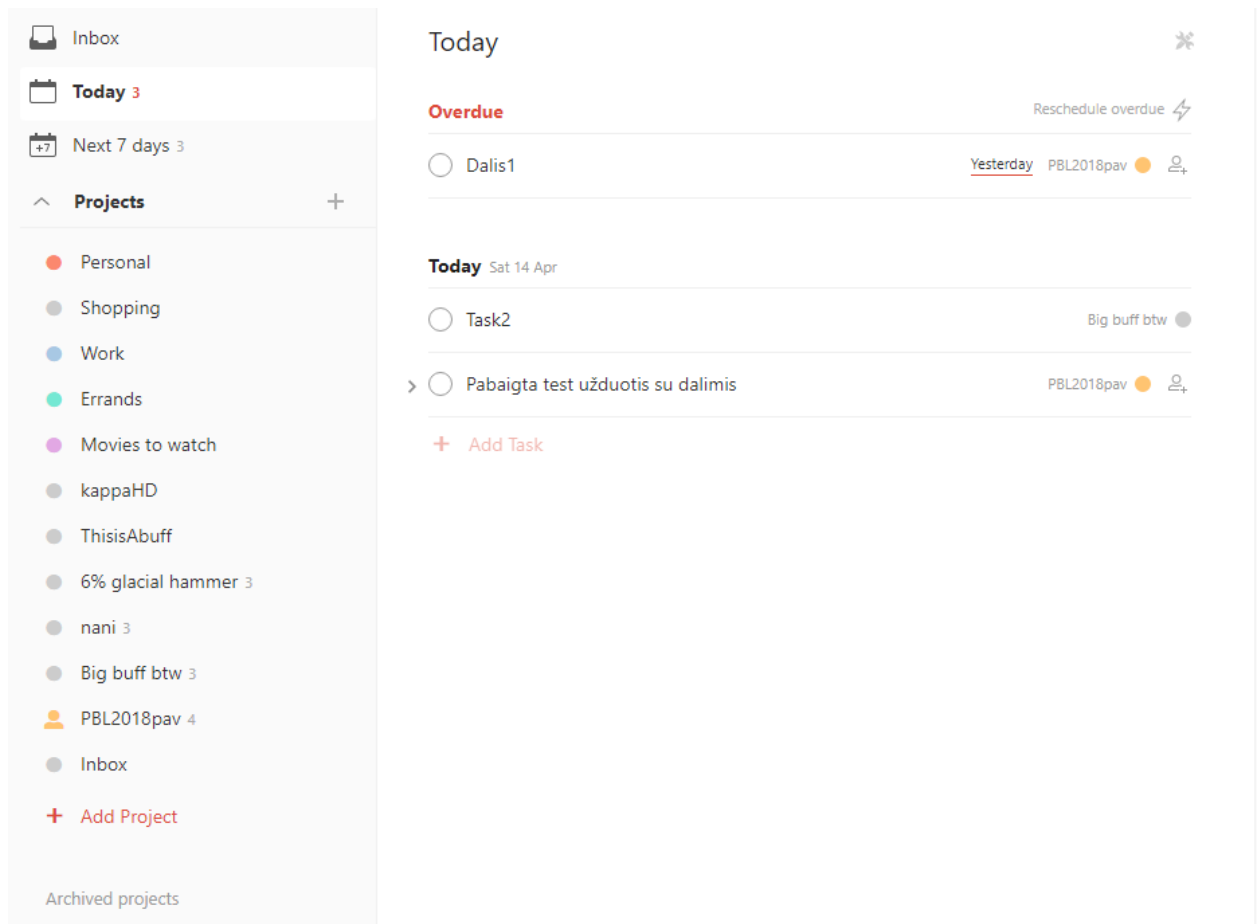


Figure 5. The main window of Todoist [4], where information can be seen

As seen in the analysis of Todoist, the system provides a great and simple way of controlling your projects and tasks. However, it does not provide any real means of data analysis nor data visualization. Therefore, our task is to obtain various data from Todoist and do various analysis of it, displaying the results in various charts and graphs. Also, some sort of data modification will be allowed in the system, but the team decided to not copy all the functionality that Todoist already possesses because it would consume a large amount of already precious time as well as achieve nothing of value because these functions are already present in Todoist itself. Moving to the next chapter, an detailed explanation on Todoist projects, users, tasks and collaborators will be provided when Todoist API is discussed. In addition to that, explaining on what the system does with Todoists data will be provided.

2 Structure, Data and Todoist API

To understand the importance of data in our system, we must first design and show the structure to provide a general view of the system.

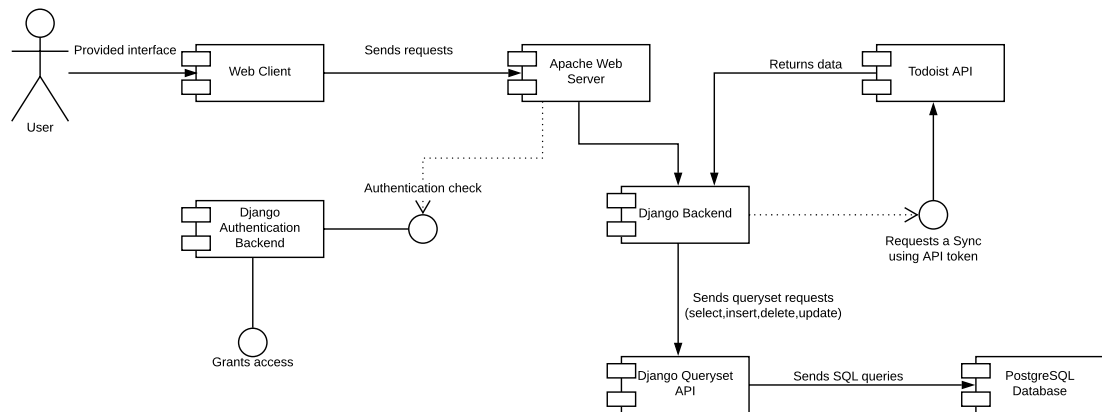


Figure 6. Structure of the System

As can be seen in figure 6 the system is split into few parts. First of all we have the web client itself and the security backend that comes with it. Through the web client, a user can send a sync request to Todoist's Application programming interface. When it happens, a sync occurs and various data, such as information on projects, tasks and collaborators is collected. the data is then written into lists of dictionaries and returned back to our web client. here it is parsed in order to help the Postgresql database understand it and then sent to the database. Database creates table rows for data received and everything goes back to the beginning.

In order to allow users to actually see the data and its analysis, the data must be first obtained from their data in Todoist. This is done by requesting the data sync using Todoist Sync Application programming interface. when a syncing function is called, all the information about a user and his projects, tasks etc. is received. Of course, not all of the data is needed and the format received is quite different from what our system needs. First of all, in order to receive data, the user needs to provide our system with the API key that he possesses. After the API key is received, a sync is made possible. The data that we decide to acquire is as following: Projects, Tasks and Collaborators. These results in three different database tables that we fill accordingly. First off we Receive and fill the Collaborator table as they have no relationships with other tables and can be created easily. When changed into a readable format for our system, the data in JSON format looks like the one provided in figure 7 . As seen, the primary key of this table is the User ID, which is later referenced to by other tables. The Collaborator itself has a full name of the user as well as email address. Additionally, JSON also holds the database table name, which later on helps with automatically updating and loading the database.

```
{
  "pk": 16257809,
  "fields": {
    "email": "rokasalech@gmail.com",
    "full_name": "Rokas Alechnavicius"
  },
  "model": "accounts.collaborator"
},
```

Figure 7. Collaborator JSON file that Django [5] understands

Moving on we have the project object. In Todoist API, the project object looks like figure 8 , meanwhile in our system, a project JSON looks like figure 9 . As can be seen, the id in Todoist API represents the project uniquely, therefore in our model it is set as a primary key. In addition to that, some fields are also taken and used in our system. The color defines color of the bubble of a project. Indent variable defines whether the project is a sub-project and its level. Parent ID variable points to another project object, which is a higher indent and a parent of a sub-project. is deleted variable is used instead of real deletion to know whether the project is considered to be deleted or not. Project name simply refers to the name of the project and is archived variable tells us whether the project is archived or not. The project token is a new variable that we use to identify which user has this project. It is used when filtering all available projects so that user would not see projects that are not his own. And as with collaborators, the database table is defined for easier updating and creation.

An example project object

```
{
  "id": 128501470,
  "name": "Project1",
  "color": 1,
  "indent": 1,
  "item_order": 36,
  "collapsed": 0,
  "shared": false,
  "is_deleted": 0,
  "is_archived": 0,
  "is_favorite": 0
}
```

Figure 8. Project object as shown in Todoist API Documentation [6]

```

{
  "pk": 2178578704,
  "fields": {
    "Color": 7,
    "Indent": 1,
    "Parent_id": null,
    "is_deleted": 0,
    "Project_name": "Inbox",
    "is_archived": 0,
    "Project_token": "70f854c94cc857ebcf160684a11342fd9b95bd0c"
  },
  "model": "projektoi.projektas"
}

```

Figure 9. Project object in our system

Lastly, we have the task object that is taken from Todoist API. In figure 10 we can see the Task object in Todoist API model. As can be seen, our database model is quite different from the basic one, as some fields have been removed. The main fields, however, stay the same:

- id field becomes the primary key of our database table instance.
- user id field becomes task uid field and points to the user who created the task.
- project id field becomes task project id field and points to the project object instance which the task belongs to.
- content becomes task content and simply shows the name of the task.
- due date field becomes task due date and points to the deadline of this task. the date format is changed a little in order for our system to understand it.
- indent becomes task indent and it shows whether this task is a sub-task or not.
- priority is named task priority and shows the importance of the task.
- responsible uid becomes task responsible uid and points to users(collaborators) that are responsible for completing this task.
- checked keeps the name and tells us whether the task is marked as complete.
- in history keeps the name and shows whether the task is complete and is marked to be moved to history.
- is deleted keeps the name and defined whether the task is considered to be deleted or not as deleted items are not removed from the database as a good practice.
- lastly, date added shows the date when user created the task.

An example item object

```
{
  "id": 33511505,
  "user_id": 1855589,
  "project_id": 128501470,
  "content": "Task1",
  "date_string": "",
  "date_lang": "en",
  "due_date_utc": null,
  "indent": 1,
  "priority": 1,
  "item_order": 1,
  "day_order": -1,
  "collapsed": 0,
  "children": null,
  "labels": [12839231, 18391839],
  "assigned_by_uid": 1855589,
  "responsible_uid": null,
  "checked": 0,
  "in_history": 0,
  "is_deleted": 0,
  "is_archived": 0,
  "sync_id": null,
  "date_added": "Fri 26 Sep 2014 08:25:05 +0000"
}
```

Figure 10. A task object as shown in Todoist API Documentation [6]

As can be seen in the figure 11 the structures are quite different. These three database tables allow us to fully sync important data from Todoist API into our project and then apply various analysis algorithms to it in order to produce desired results. A more full documentation on Todoist API objects and their fields can be found in Appendices. Moving on, the database will be shown and explained more and the whole structure will be shown in various diagrams.


```

{
  "pk": 2559659817,
  "fields": {
    "task_responsible_uid": null,
    "task_indent": 1,
    "task_parent_id": null,
    "task_date_added": "2018-03-18T12:26:34Z",
    "task_Content": "Buff earthquake",
    "task_project_id": 2180537652,
    "in_history": 0,
    "task_priority": 1,
    "task_uid": 16257809,
    "is_deleted": 0,
    "task_due_date_utc": null,
    "checked": 0
  },
  "model": "projekta1.task"
},

```

Figure 11. A task object as seen in our data model

3 Design

3.1 Database Design of the System

As can be seen in figure 12 this is the database structure of our project. Below figure 12 is a full explanation of the tables as well as relations that allow and simplify working with data in our Todoist extension system.

First off, we have the basic Django [5] user model. It has a simple auto incrementing primary key that is id in addition to some basic fields that are used to add additional information about the user(not required). In addition to that, User model has a unique field username because username is used to access the system, therefore it cannot be non-unique.

Next is another table about user information which is called UserProfile. This table also has a simple auto incrementing primary key of id. In addition to that, there are a few unique fields such as userid, token and email. userid refers to the ID which todoist returns about the user, not a local one that we create and use. Token field refers to Todoist API token that is used every time user requests a sync. Email is the same as in the auth user table. Other simple fields refer to some additional information that is provided by Todoist's API. The other user id field is a foreign key to auth user model and links them together, so that each user instance would have one userprofile instance.

Going further into the data that we receive from todoist, we move to Collaborator table. This table refers to people that create or are assigned tasks. ID is a primary key and it refers to the user id which we receive from Todoist. Next two fields also refer to the same information, accordingly to the name and email of the users.

The two main tables are Project and Task. First of, Project table refers to all the projects of the user. Its primary key is Project_ID and it refers to the ID which our system receives from Todoist after syncing. It is unique for every project and therefore can be used as a primary key very well. Next, few fields are added to obtain additional information on the project itself. Project_name refers to the project name, Color field refers to the color of the bubble that is provided next to the project name in Todoist. Indent tells system the level of indentation of the project. It also helps us identify the project a little better, because a project with Indent 1 can never be a subproject, and a project with indent 4 cannot have subprojects of its own, etc. Is_deleted field refers to whether the project is deleted or not, however this field is not yet used in the system due to Todoist API. Is_archived refers to whether the project is archived or not. Parent_id is a foreign key to its own table. This field helps us identify whether the project is actually a subproject, and which project is the parent of a subproject. Project_token_id is a foreign key to UserProfile table and it simplifies writing queries in order to find all projects of a user. Item_order is a great and useful field that Todoist API provides us value for. It determines how projects should be printed(for example in a table), in other words, the ordering of all projects. This simplifies ordering in queries.

Last very important table is table Task. The primary key is task_id, which identifies each task uniquely. the id's are received from Todoist. Task_content simply show the text contained within the task object. The other basic fields have been explained in the previous section, therefore we are skipping them. After them is task_project_id which is a foreign key to Project table and helps identify which project does the task belong to. Next, task_uid refers to table Collaborators and identifies which person created this task. Similar to that, task_responsible_uid points to the same table, but this time it shows the person who is responsible for the task. Similar to the projects table, task_token points to UserProfile and helps identify who created these tasks. And lastly, item_order

helps with the printing of the tasks.

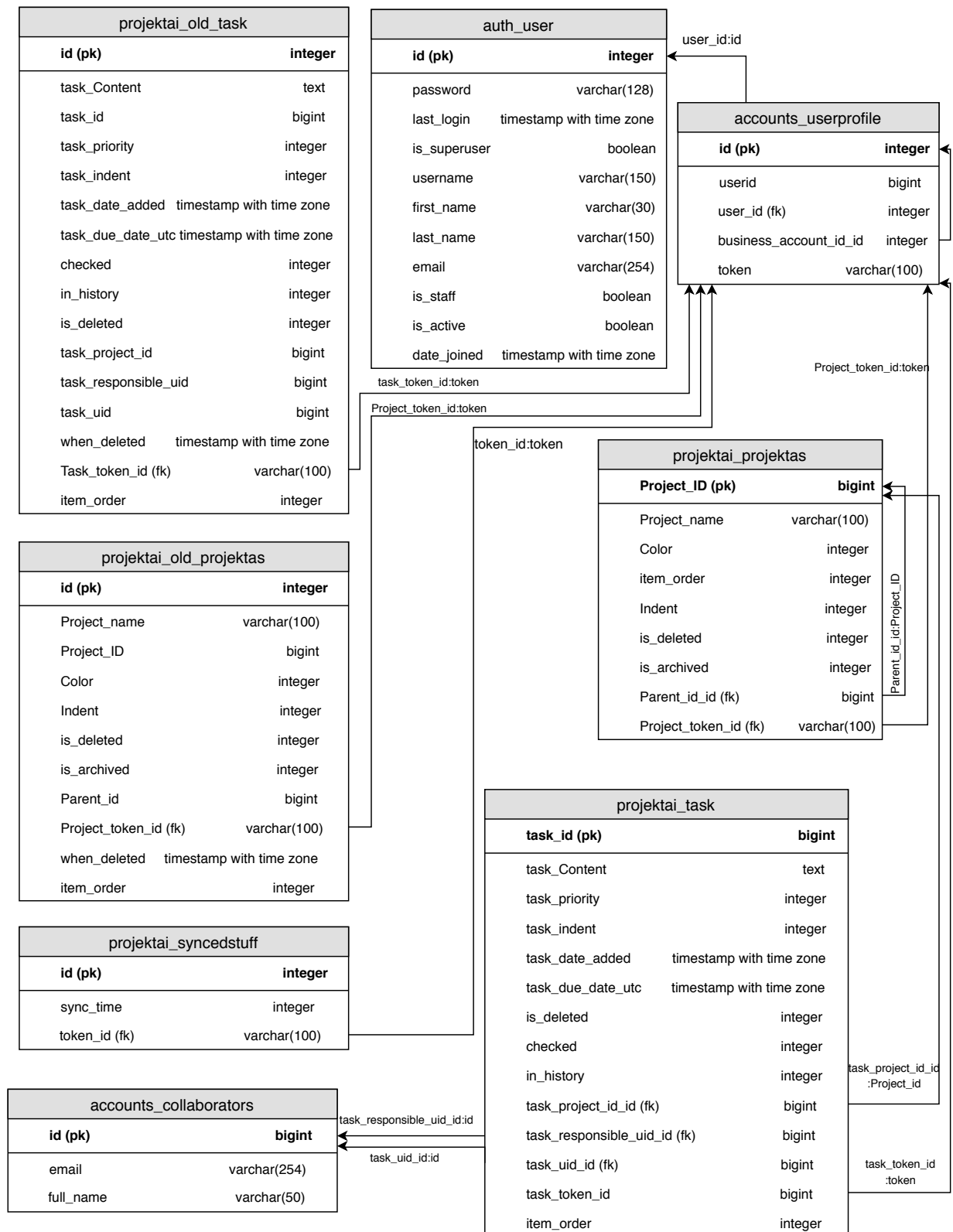


Figure 12. Database design of the system

Another table, SyncedStuff refers to the syncs that each user did. Its primary key is a simple auto incrementing id. It also has a field of DateTime, which identifies when a sync occurred. Also, it has a token_id field which is a foreign key to UserProfile and helps identify which user did the

sync at the time.

Lastly, there are two tables that contain Old Projects and Tasks. They are needed because our system does data analysis and previous data is important. It holds similar fields to the Project and Task tables, however, most of the relations are destroyed in order to keep the old data in check during resyncs.

3.2 Design of the Website

Before we begin describing the graphical part of the system, one must know what exactly is the design such as calendars used for. For this reason a Use Case diagram is presented in figure 13. As can be seen, the system has a simple authorization process, where unauthorized user can simply register and log in. An authorized user can view the rest of the website, that being : dashboard, charts, calendar. In addition to that, he can always request a manual sync which deletes and recreates new data using Todoist API. Also, the user can choose to keep his old data up to a limit which he can broaden or narrow till a certain point. This Old data is used in other graphs.

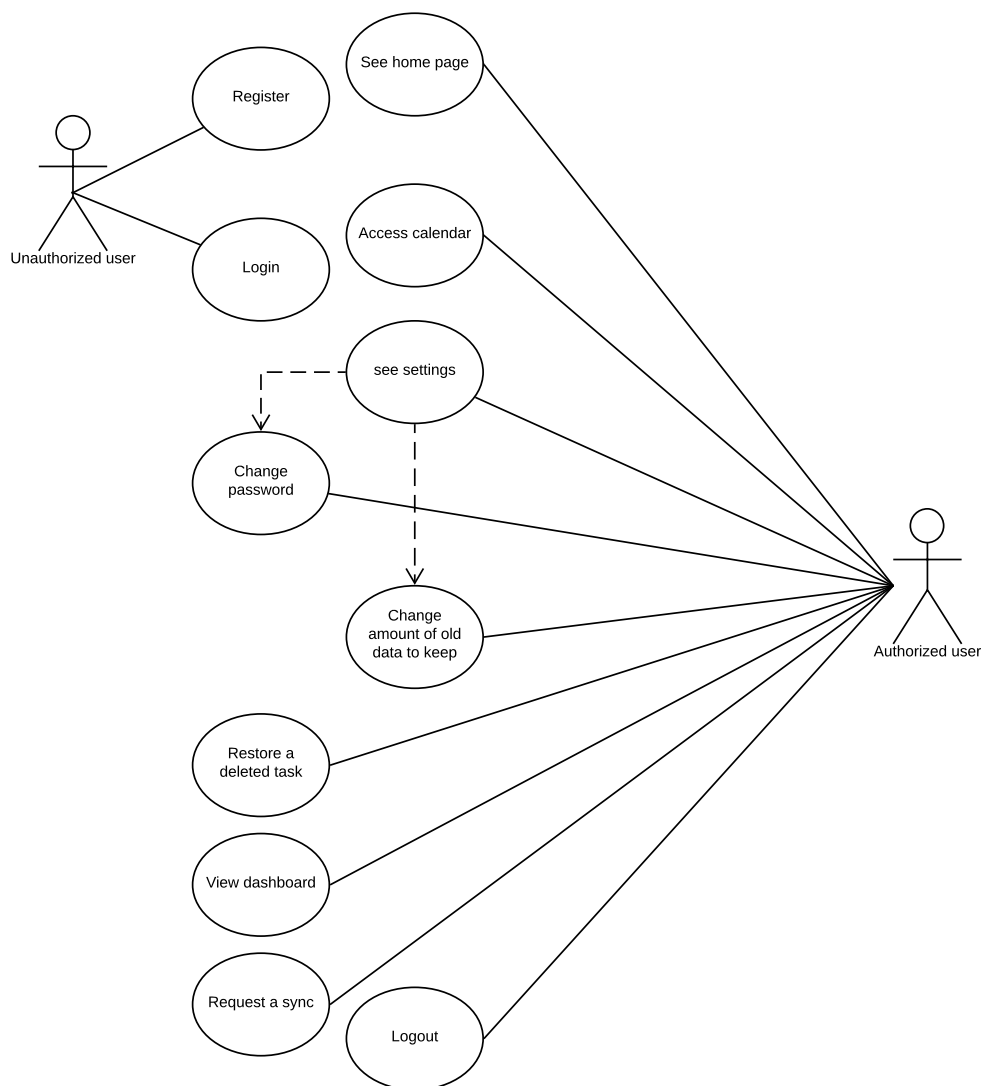


Figure 13. Use Case diagram of System ClapDoist

Before creating any real web templates, the team gathered and came up with a mockup for the base User Interface. It was decided that it should have a table with important information such as projects, tasks, overdue tasks, tasks that have a deadline for today etc. In addition to that, it was made clear that the system would need various graphs and charts that would show the data analysis part of the program as well as a calendar, where the user could see the tasks on their deadline dates, do some action with them. A first UI mockup can be seen in figure 14 . As expected of a mockup, it mostly displays the idea which the team desired to implement in the system rather than the actual UI.



Figure 14. First user interface mockup

After the mockup it was decided to split the whole user interface into three main parts. The dashboard with important information about overall projects, with the addition of moving to a project page to see its details, tasks and other important information, a page for various charts with the addition that users themselves could choose which charts to see, and a calendar page where a manager can see all tasks with deadlines, click on them for additional information etc. In addition to these three parts, they all share a similar design. This is done by designing a base page with a navigation bar on both top and the left side of the website, placing all the important navigation and adding styling to make it look more lively and interactive to use. Then, each of the main parts of the UI extend the basic design by adding their own elements to create a unique page with unique functionality.

3.2.1 Dashboard Design

In order to make the website appealing, a nice and simple, but elegant design had to be implemented in the system. A nice template was created, which can be seen in figure 15 . The most important page a user always sees is his dashboard, therefore it was created to be informative, clear, beautiful. As dashboard template is made using mostly Javascript, Bootstrap and base Css, it should work on most computers and browsers. In addition to that, the template is fully scallable, meaning it looks well on any screen size, cofiguration etc. Dashboard provides user with all the necessary information, such as project and task count, overdue tasks, tasks due today, some basic charts and graphs that visualise this information for those who enjoy visual information more than written one.

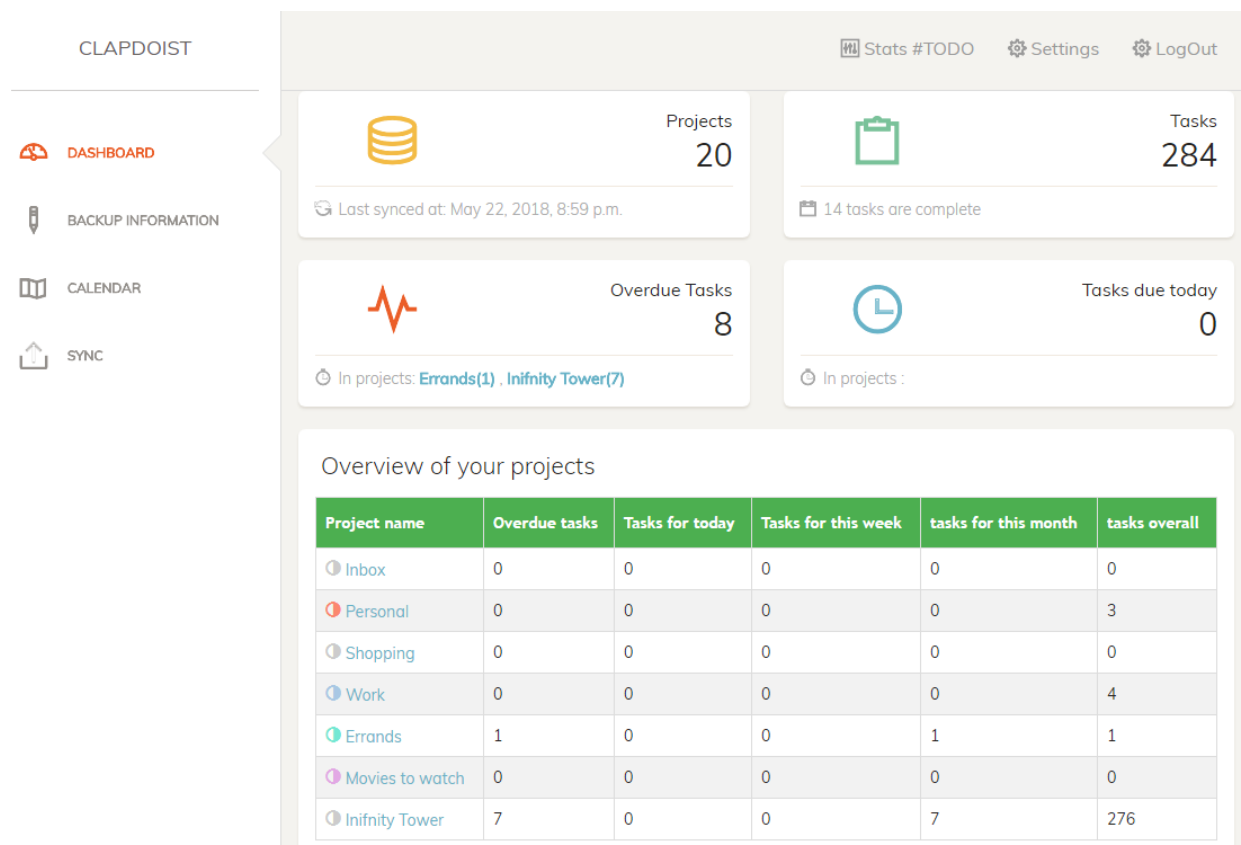


Figure 15. Top part of the Dashboard in ClapDoist system

Obviously, more deeper page had to be created in order to truly see and understand how your projects are doing. After clicking on any of the projects, the user is redirected to a project page. On first glance, this page is exact the same as the dashboard. However, it has some distinct differences. First of all. projects can now be filtered by both people responsible for tasks and simple filter for complete, overdue, etc tasks. There were a few different ideas on how to display this filtering, but it was decided to create buttons for each filter instead of creating two lists. Perhaps this approach is less effective, but it is more elegant. As can be seen in the figure 16 , differences are visible. The second difference is the table itself. This table only displays information about one master project and its subprojects. In addition to that, with each filter applied, the table reduces its data and tells the user which projects still have tasks in them (indicated by a click to expand message). In order to not fill the page but still display everything of importance, an accordion was added to

the table of tasks and they can be opened and closed freely. Also, the same four containers are present just like in the dashboard. However, this time they only display information based on the one main project and not all projects. The exact same goes for the graphs, where their information is properly filtered to produce correct results.

Filter projects by :

Overdue
Today
This week
This month

Uncompleted

Filter tasks by a collaborator

morehacking
Rokas Alechnavicius

Overview of project A master project

Project name	Overdue tasks	Tasks for today	Tasks for this week	tasks for this month	tasks overall
👑 A master project (Click to expand)	5	0	0	0	20
📁 a master subproject i would hope (Click to expand)	3	0	0	0	14
Tasks of a master subproject i would hope	Overdue	Due today	Due this week	Due this month	Is complete
* just meaningless tasks without deadlines					
* and more					✓
* alright, back to useful tasks	!				
* just kidding, but we are now					✓
* just kidding, but we are now					
* more useful tasks	!				
* allrighty, subtasks it is					✓
* more subtasks					✓
* last subtask	!				
* sub sub tasks already??					✓
* this is crazy i tell you					✓
* nania					

Figure 16. Distinct changes in Project Dashboard page

3.2.2 Plotly Framework, Charts and Graphs in ClapDoist

After doing some research work and trying out quite a few different frameworks for drawing charts we have decided to stay with plotly.js [7] since it provides us with powerful scientific graphing possibilities while being easy to learn, maintain and integrate into our system.

Other frameworks we have tried are Dash [8], plotly.py and chart.js but none of them met our expectations since they were either too hard to integrate without being too time consuming or were just inferior compared to plot.js. All of ClapDoist graphs are made with plotly.js while the data is filtered from our database and using django queryset API. In conjunction to being very simplistic for inexperienced Javascript developers plotly also offers a lot of built in functions which come with every chart and are very easy to customize. Figure 17 is a plain bar chart made using plotly.

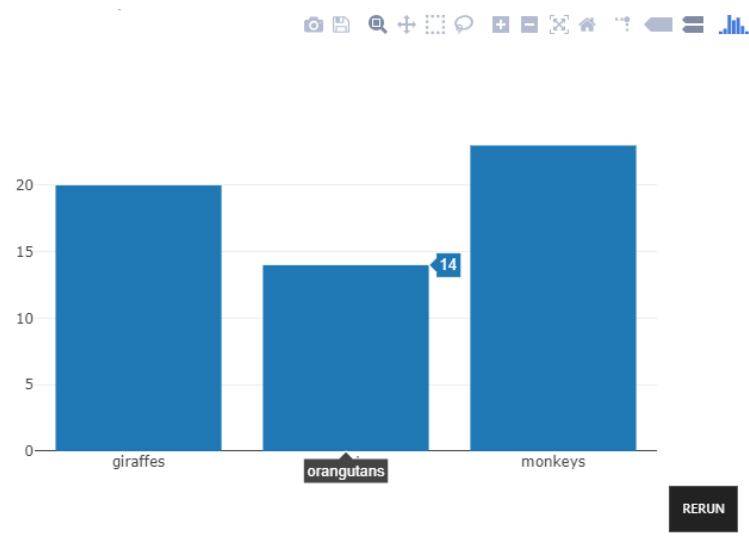


Figure 17. Basic chart using plotly

The Javascript for such a chart is as simple as follows :

```
var data = [{  
  x: ['giraffes', 'orangutans', 'monkeys'],  
  y: [20, 14, 23],  
  type: 'bar'  
}];  
  
Plotly.newPlot('myDiv', data);
```

Figure 18. Javascript script for a chart

As can be seen, only the data was specified but when Plotly is called it also wraps the charts with its own interactive tool set. All of the tools are provided by default but there are options to remove some and to add some for each chart. Tools are as following:

- Download as png
- Edit
- Zoom
- Pan
- Box Select
- Lasso Select
- Zoom in
- Zoom Out
- Autoscale
- Reset Axis
- Toggle Spike Lines
- Show Closest Data on Hover
- Compare Data on Hover

Regarding more interactivity using plotly.js we also see that, by default, hover text is enabled, all of the bars have an option to be enabled and disabled. Graphs, in general look clean and tidy. However, plotly provides us with many powerful customization options which can be used to create something completely new and adapt to every situation. Of course, finding the right settings might take some time but that also is a possibility. So far plotly has been an invaluable tool. So far we have created the following graphs :

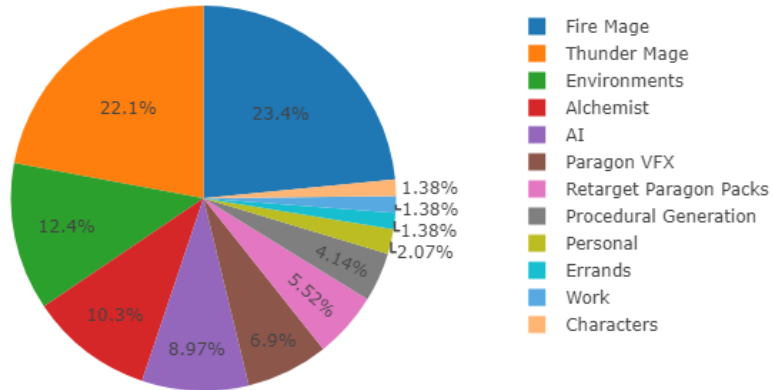


Figure 19. Pie chart of tasks in projects

- Basic Pie chart, shown in figure 19, which shows how many tasks each project has. This Chart has been implemented in both a view of all projects and a view of a single master project along with all of its subprojects all the way down to project indent of 4.

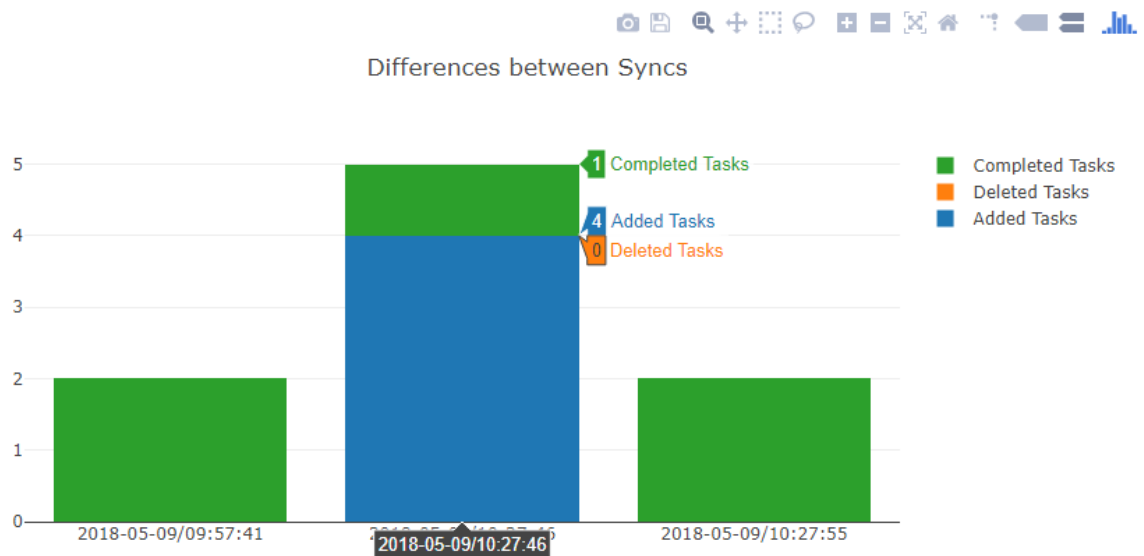


Figure 20. Difference graph of users tasks

- A stacked bar chart (figure 20) which provides us differences of the database during each sync. For example, if two tasks were completed and one deleted, then the stacked chart should display them so that user could see how his projects and tasks have changed. The exact same chart is used to display diffs between syncs in one project tree, meaning that it only displays changes that happened to one master project and all of its subprojects.

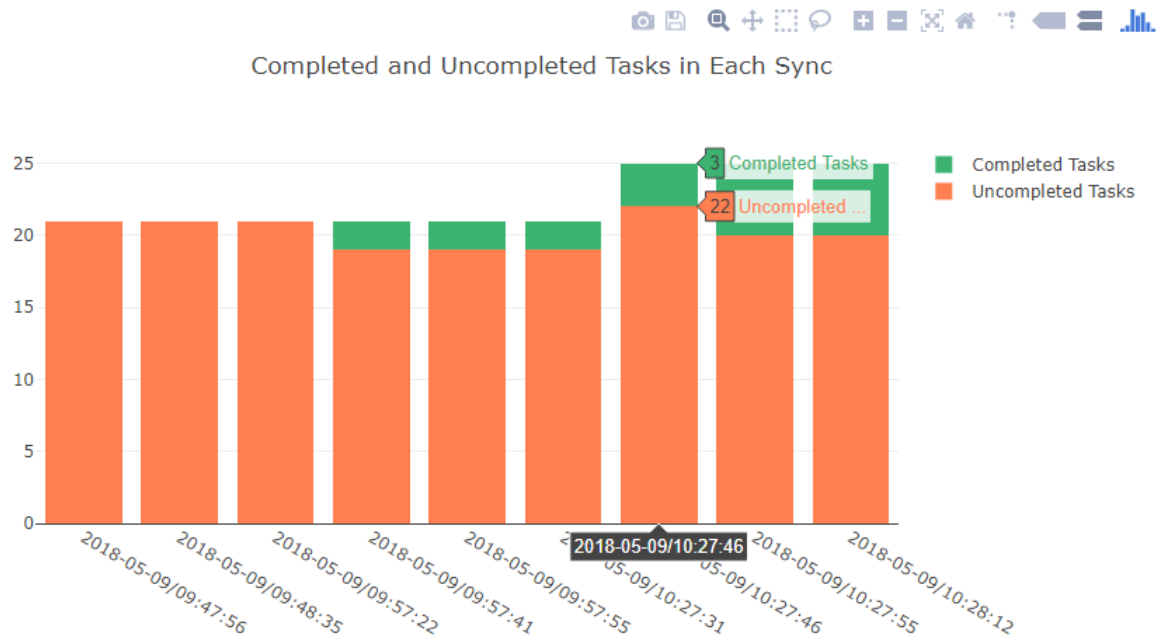


Figure 21. Bar chart that displays completed and uncompleted tasks

- Figure 21 shows a bar chart to provide information about difference in completed and uncompleted tasks in a currently selected project. Additionally, same goes for all projects, and filtered projects etc.

3.2.3 Calendar

Another important part of the website is the calendar as it provides a real view on your tasks. Instead of looking at tables or charts, you can truly see all the tasks that need to be completed in a certain amount of days, hours etc. Our calendar is an extension of a fullcalendar, which is a javascript jquery plugin for making great looking and functional calendars. In our case, the calendar is customized to be able to display information for a month view, week view or a day view. The month view can be seen in figure 22 where it displays task content for every task that has a headline on the shown day. Tasks without deadlines are not displayed as they would only litter the view.

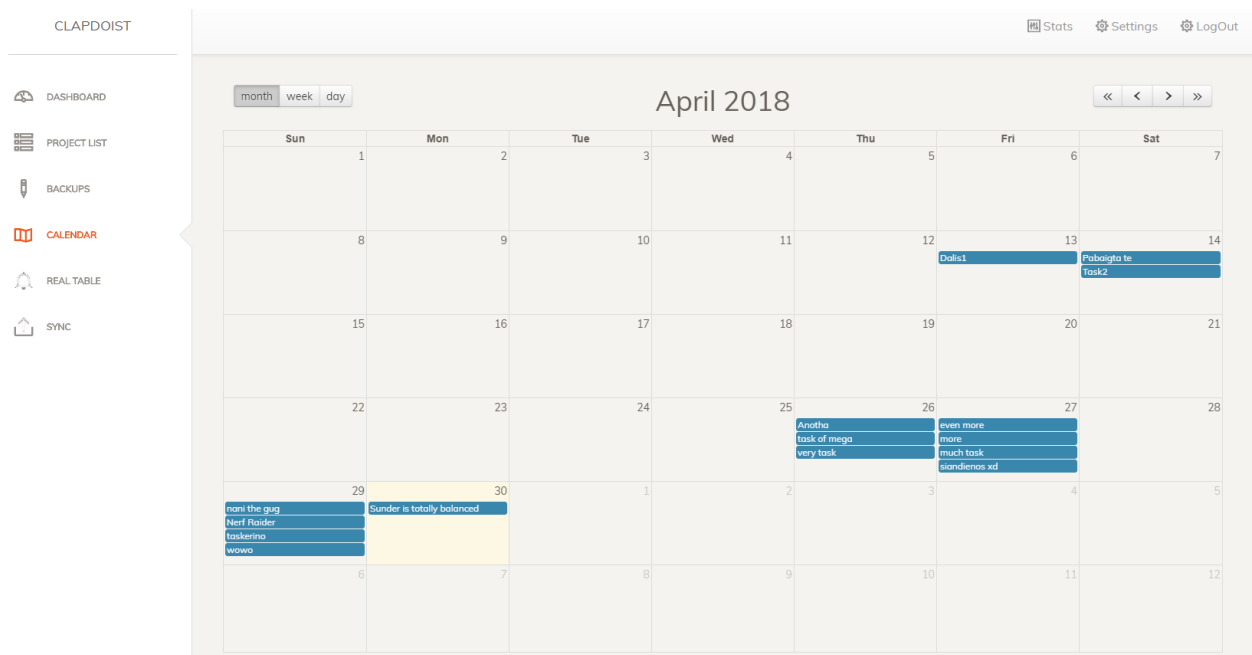


Figure 22. First version of the Calendar plugin

4 Implementation

4.1 Developing Tools

Decision was made to use Django Web Framework for the base and the backend of our project because it is up to date powerhouse for website creation(most frameworks are up to date, really) and also because it is based on Python programming language that team members are familiar with. Another agreement was made to use Django 2.0.2 as a stable and constant version. This is important to mention because Django updates itself quite frequently, and it is entirely possible that at the time of project defense, a new stable Django version will be out and running. Having web framework based on Python also helped us out because another key tool in creating our system is Todoist Application Programming Interface which is also made for Python in our case. This means that there was no need to switch between languages when making the system, making the work flow much more efficient and understandable.

To create the calendar, a Javascript event calendar was used. We chose FullCalendar [9] because it is a great solution for displaying events and that was exactly what the system needed to show.

To create various charts and graphs about projects, tasks and data analysis, a JavaScript library Plotly.js was used. It is built on top of d3.js and stack.gl, making it a high-level declarative charting library.

In order to deploy the website on the Internet, Apache web server was used. Apache is a sturdy and trustworthy way to deploy your applications, therefore it was chosen. In order to make it compatible with a Django website, a `mod_wsgi` was used. `Mod_wsgi` is a package which implements an Apache module which can host any Python web application. This makes deploying our project to the Internet a breeze.

Lastly, we have used a database for our project. Django has a built-in SQLite3 database which works wonders for most projects. However, in our case this simply was not enough as the system is quite largely scaled and a large amount of data is stored within. For this reason we have chosen PostgreSQL 10.3 database server as it is fast, powerful and trustworthy. In addition to that, switching the database in Django project is very simple, therefore using Postgres cost no problems to the implementation process itself.

As a worthy note, to program the website itself, a PyCharm IDE was used. Normally, these things would not be mentioned, however, this IDE allows the programmer to remotely connect to a virtual machine where our system is hosted, meaning code can be changed and then remotely pushed with one button, therefore seeing immediate changes in the website and providing more accurate testing.

4.2 Website Implementation

This chapter will discuss main properties of website ClapDoist. Main functionality could be split into three major parts: Calendar implementation, charts and graphs implementation, various data presentation implementation. In order to show which parts require which functionality, a website hierarchy diagram is presented. As can be seen in the figure, the website hierarchy is extremely simple. First off, most of the website is protected by a simple authorization of registering and logging in. After logging in, the user goes to either a confirmation page, or a form where he can set up his Todoist API token(if he just registered and has no API key assigned to himself).

After providing the key, users are redirected to their dashboard page, where they can see various information and access other pages with content, calendars etc.

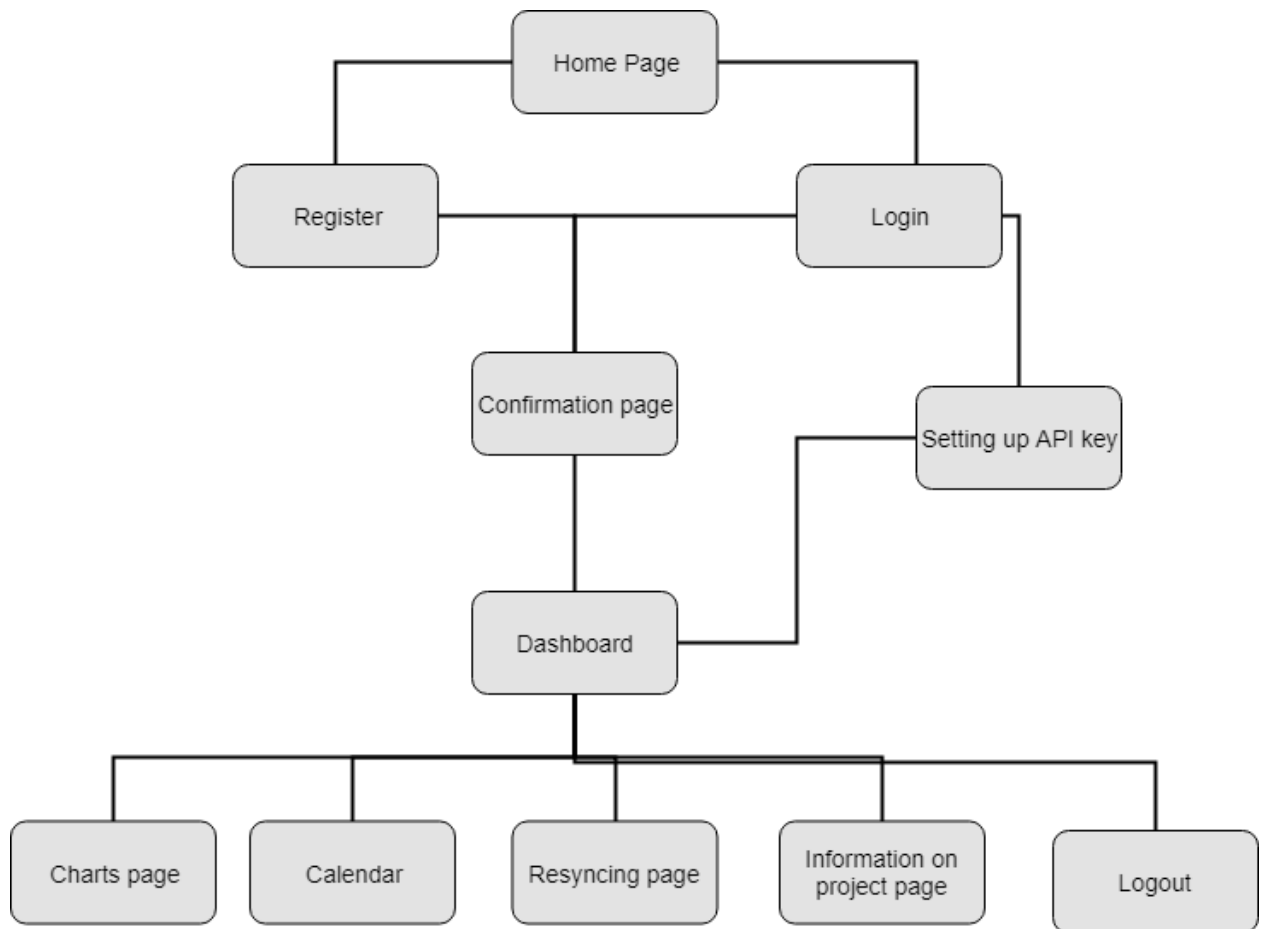


Figure 23. Hierarchy of the website pages

4.2.1 Synchronization, Backup implementation

In order to provide a clear explanation on the full process of a synchronization for one user, a mixed activity diagram 24 is provided. This diagram represents not one, but three functions, which together form a full cycle of synchronization.

The sync happens as follows in the diagram: if it is the first synchronization, a simple sync with Todoist API happens. During it, projects, tasks and collaborators are created and inserted into the database. If, however, there were previous syncs, the system first of all creates old projects and tasks, then creates a sync point which can be later referenced in order to get all old projects and tasks that were deleted during the sync at that time. Then a checking function is called, which compares amount of old syncs with the amount of syncs that user chose to keep. If there are too many syncs, the oldest one gets deleted until the wanted amount of syncs is reached. Then, the base sync function occurs and projects, tasks, collaborators get created.

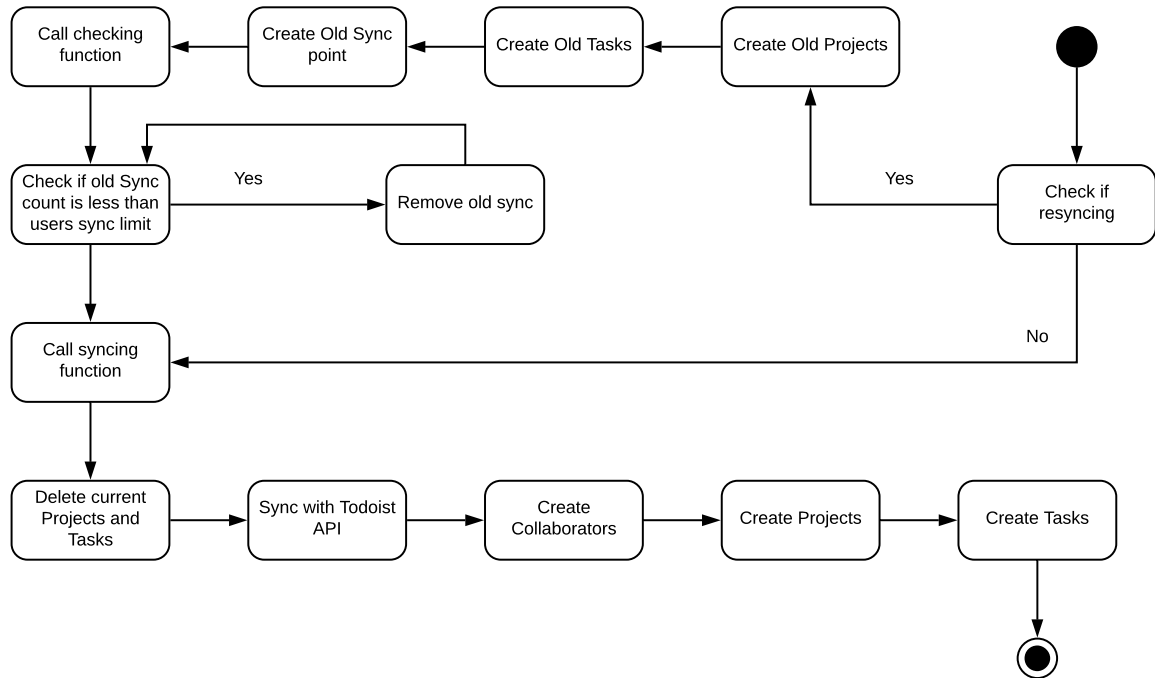


Figure 24. Synchronization diagram

Backups are a vital part of this system. Just to make it clear, when one talks about backups in this system, an action where user can restore a deleted task back to both our system and Todoist project management system is meant. In order to explain how it works, a full explanation on functionality is provided :

- The user browses through backups of different syncs and chooses a task to restore
- as the task is not present anymore, it does not exist in Todoist, and a new one is created and given attributes such as content, person responsible
- Task is then sent to Todoist and the exact same task is created in our system. Also, as it is a new task and has new task id attribute, all the existing Old tasks that are representing the currently deleted task are updated to match task ids with eachother.

This way one can comfortably backup tasks that were deleted and no additional sync to see the results is required. This helps with speed as well as simplifies the process of recovering deleted tasks for a ClapDoist user.

4.2.2 Table Data Implementation Throughout the Website

It might not seem like it, but tables in ClapDoist system actually hold a vast amount of difficult information that is all being constantly calculated in the backend of the system. As the website is made with Django, the backend runs on pure python functions, therefore it is quite fast and optimized.

When talking about tables, first off one must find out which projects are subprojects and which are the parent projects. The difficulty with that comes to the fact that Todoist only returns the very highest level relationship, meaning that a sub sub project will be a children of a sub project,

however he will have no direct relationships with the main parent project, meaning that if we wanted to extract all tasks of a full project we would have to manually descend all the way down into fourth level of indentation and calculate task count accordingly. In order to optimize time consumption as much as possible, carefully chosen queries must be created so that Django does not litter database with unneeded requests and unimportant data. In order to achieve that, some algorithms were made for universal use in the website, and a few of them will be presented in this subsection. The algorithms can be split into two main parts: Project calculations and Task calculations.

Starting of with the Project calculations, it is important to make clear why are we using this function. Basically, it is used in order to calculate how many tasks and their statuses that each project holds, in addition to counting how many does each project tree hold. These are used to clearly represent status and importance of each project and the tree that they belong to. In addition to that, the function is made universal, meaning that it can also be used to calculate information about one project tree for another table by just limiting the data which you pass on to the algorithm. To understand the function more clearly, a step-by-step explanation is provided:

1. function defines some base time parameters in order to be able to calculate number of tasks for each of the following: Overdue, Due Today, Due This Week, Due This Month
2. function begins the iteration process. First iteration is only through the main parent project of a project tree, meaning that it is a project with no parent ID present. Function calculates the count of his own tasks that correspond to the fields mentioned in the last point.
3. Project calculations then moves down a level and iterates through level one subprojects of a parent project, calculating all their own tasks, in addition to adding their task counts to their parent project accordingly
4. This iteration repeats itself until the very bottom of level four is reached. Then all the results are inserted into a list of dictionaries and returned to the page view function in order to be used for other calculations and displayed.

The one unfortunate part about this algorithm is the fact that project indentation which is gained from syncing with Todoist API is not always as correct as it should be, meaning that our functions cannot rely on indent alone to make sure that a project is a subproject or not. This causes some chaos in the code, making it a bit less readable and slower.

Second important algorithm for gaining enough important data for tables is the Task calculations function. This function is responsible to find out whether a task is Overdue, belongs to someone etc. As starting parameters, it receives a user whose tasks it will be searching, a special query which defines if initial filtering is required, a list of project id's whose tasks it will be counting and lastly a collaborator id, if the function should only look for tasks that a person is responsible for. This is a more simple function, making its flow more basic. The whole algorithm can be broken down into few points:

1. First off, the function checks whether a query is provided, and if so, it filters the whole task pool in order to reduce the amount of time needed to complete the algorithm. The exact same is applied for the collaborator ID that is either given or not

2. A for cycle is run through the tasks that remain and information about each and every one of them is gained
3. The new information is inserted into a list of dictionaries and the data is returned back, where it is used for injection into the template and chart related calculations.

These two functions together provide most of the information needed to provide users of the system a detailed overview of their projects and tasks, deadlines, places where more attention should be paid to. In addition to that, they can be quite useful when creating charts, reducing amount of calculations that need to be made separately.

4.2.3 Calendar Implementation

Finding tasks with deadlines is not a difficult thing to do. The working part comes when one wishes to visualise these things in different ways. One of chosen ways was a simple calendar. In order to make it work, most of the work was done in Javascript, meaning that there is no real need to explain the pythonic backend this time. However, this is a good chance to provide a step by step explanation on how the calendar works and what kind of information it provides. In order to create the calendar itself, a Javascript plugin named fullcalendar is used. This provides the system with all the basics needed for a great calendar. In order to customize it, some fields are added and removed as can be seen in the figure 25

```
$('#calendar').fullCalendar({
  height: 'parent',
  header: {
    left: 'month,basicWeek,basicDay custom1',
    center: 'title',
    right: 'custom2 prevYear,prev,next,nextYear'
  },
  selectable: true,
  eventLimit: true,
  dayClick: function() { tooltip.hide() },
  eventResizeStart: function() { tooltip.hide() },
  eventDragStart: function() { tooltip.hide() },
  viewDisplay: function() { tooltip.hide() },
  events: [
    {% for i in events %}
    {
      title: "{{ i.task_Content }}",
      start: '{{ i.task_due_date_utc|date:"Y-m-d" }}',
      end: '{{ i.task_due_date_utc|date:"Y-m-d" }}',
      {% if i.task_responsible_uid is None %}
      description: 'No one has been assigned this task',
      {% else %}
      description: 'person responsible for the task is: {{ i.task_responsible_uid.full_name }}',
      {% endif %}
      created: '{{ i.task_date_added }}',
    },
    {% endfor %}
  ],
},
```

Figure 25. Initialization of Fullcalendar calendar

As per the options, the calendar is made to fit the screen, some different view buttons are added in order to make it possible to browse through month, week and day views, browsing throughout months is made available too. In order to switch views, calendar is selectable. Also, because a person might have a lot of tasks due to one day, event limit variable is also initialized in order to properly display many tasks. Additionally, events are created for each task that has a deadline. As can be seen, every task gets its content, start and end dates and a different description depending on whether a person is responsible for that task. After initializing the calendar, additional function is created which provides pop-ups which show additional information about the task. This function, as can be seen in the figure 26 creates a function which happens every time a user clicks on an event(task). It creates a pop-up box, which has its styling defined much earlier, and assigns content to be displayed. This provides user with additional information and makes finding important tasks much easier, in addition to not littering calendar with full content about each and every task when there is no need to do that.

```
eventClick: function(data, event, view) {  
    var content =  
        '<h5>' + 'Task: ' + data.title + '</h5>' +  
        '<h5>' + 'Responsibility: ' + data.description + '</h5>' +  
        '<h5>' + 'Created on: ' + data.created + '</h5>'  
    ;  
  
    tooltip.set({  
        'content.text': content  
    })  
    .reposition(event).show(event);  
},
```

Figure 26. EventClick function of fullcalendar

4.2.4 Graphs implementation algorithms

In order to be able to provide our users with history of change regarding the flow of tasks, we had to implement a feature which calculates every change in the database using synchronizations we get from the TodoistAPI.

To implement this feature, every synchronization is backed up in our database and users are able to choose how many backups they wish to hold at once. Then we used a method similar to sorting, where two of the older synchronizations are compared to each other for differences in their task ID's, completion values of those tasks and whether they were deleted. The programming part of the implementation is really just nesting a lot of for sentences in order to compare and extract all the possible information. The first FOR cycle goes through the list of all syncs, taking the oldest one and the one after it. Then there are three nests of for loops for each of the tasks states - completed tasks, deleted tasks and added tasks.

Completed tasks - two lists of tasks (i and i+) are compared for difference, mainly looking if the task in list i with values of 0 (uncompleted) are changed into values of 1 (completed) then the

third list is compiled out of the tasks that changed their states between these two syncs and returned as completed tasks. We mainly looking at numbers of tasks completed, so the exact contents of the task are not checked. After finding a number of completed tasks between the the synchronization points, the numbers is appended to a list of “Completed Tasks” and is done so again until all of the middle points between synchronizations have gone through the process of this extraction.

Deleted tasks - the idea behind creating lists of deleted tasks is ordering tasks with Queryset before working with them so the least amount of instructions is used. Nevertheless, this operation is the most costly, since we haven not optimized it and it just compares every single task against the list of tasks. So you take a task from list *i* and compare it to a list of all tasks from *i+1*. If the task is present in both lists, it is then skipped and the code goes on to look for another. If there is no task present in the list *i+1*, it means that it has been deleted and is noted by us as such. Following the same principle as done with completed tasks, the numbers of all deleted tasks between the closest synchronization points are compiled into a list of their own and exported over to plotly.js as context data for creating diagrams.

Added tasks - extracting added tasks is done by creating a list of all tasks from “*i+1*” and then removing all tasks from that list that are present in list “*i*”. With this simple logic we can reliably check whether we have any new tasks between these two backup points. As always, the numbers for added tasks in between the closest synchronization points are compiled into a list of their own and exported over to plotly.js for building diagrams.

It is worth noting that all of these functions have to be completed one after another, in no particular order, but in the same block as the constant appending to lists of context data has to follow the same pattern. This means that we have calculate the differences of Added, Completed and Deleted tasks and add them to the list so they would all be on the same indent in different lists when exporting over to plotly. Completing all the calculations for one function before moving to another might result in plotly.js not accurately describing the ongoing situation and, in turn, rendering the whole diagram part meaningless.

Figure 27 shows how the code looks for calculating the differences in backup pairs. In this snippet Queryset API filtering and ordering is already done. These three nested FOR loop blocks create arrays for one of the each states we are looking for - Added, Deleted, Completed, in that order Please note that they are all inside another FOR loop as to be completed one after another in one cycle. After the cycle ends, backup points are moved by one indent forward and the whole calculation is done anew until the latest synchronization is reached.

While the differences are accounted for in pretty much the same way for tasks in a single project, the method of interest here is getting the list of all tasks that belong to a single project and all the information in general, since we need ID’s of tasks provided by TodoistAPI, as well as their indents and priorities.

This is done by creating a block of four nested loops, since fourth loop is the deepest indent allowed by Todoist, so we can safely assume that it is going to be the furthest one and can adjust our calculations to work until the fourth indent or less rather than over-encumbering ourselves with intricate calculations of the deepest node.

These nests take the root project and using the identifiers of indent and parent project id compile a list of all projects that belong to the said root project. The list is then given over to Django QuerySet API, which returns us the list of all tasks present in that tree. This new list is simply given over to to the difference calculations and the same process is repeated as is done with the whole database differences, except this, only the tasks of one root project are used.

```

for task in tasksSecond:
    for taskB4 in tasksFirst:
        if task.task_id == taskB4.task_id:
            addedTaskDiffList.remove(task.task_Content)

for task in tasksSecond:
    for taskB4 in tasksFirst:
        if task.task_id == taskB4.task_id:
            deletedTaskDiffList.remove(task.task_Content)

for task in tasksSecond:
    for taskB4 in tasksFirst:
        if task.task_id == taskB4.task_id and task.checked == 1 and taskB4.checked == 0:
            completedTaskDiffList.append(task.task_Content)

```

Figure 27. Calculations of differences

The figure 28 displays main method for extracting all of the child projects from a specified root projects. It is worth noting that we have two distinct model types in our database, Projects and Tasks are first one and Old Projects and Old Tasks the second one. The main reason such peculiar method was chosen, is because we need an easy way to access the backups in our database and to be able to quickly discern whether any of the particular objects we are interested in are still relevant in the latest sync or have they been deleted or completed in previous ones. The main difference between these two model types is that the models with Old prefix have a date of when a respective synchronization occurred. As mentioned above, figure 28 displays the extraction of all tasks relevant to one root project:

```

def get_project_tree(token, profile, projectinstance, synctime):
    projlist = []

    for project in Old_Projektas.objects.filter(Project_token=token, Parent_id=None,
                                                Project_ID=projectinstance.Project_ID, when_deleted=synctime).order_by('item_order'):
        projlist.append(project.Project_ID)
        for subproject in Old_Projektas.objects.filter(Project_token=token, Parent_id=project.Project_ID, when_deleted=synctime).order_by(
            'item_order'):
            projlist.append(subproject.Project_ID)
            for subsubproject in Old_Projektas.objects.filter(Project_token=token,
                                                            Parent_id=subproject.Project_ID, when_deleted=synctime).order_by('item_order'):
                projlist.append(subsubproject.Project_ID)
                for subsubsubproject in Old_Projektas.objects.filter(Project_token=token,
                                                                    Parent_id=subsubproject.Project_ID, when_deleted=synctime).order_by(
                            'item_order'):
                    projlist.append(subsubsubproject.Project_ID)

    taskIDList = []

    for i in range(len(projlist)):
        for task in Old_Task.objects.filter(task_project_id=projlist[i], when_deleted=synctime):
            taskIDList.append(task.task_id)

        i += 1

    return taskIDList

```

Figure 28. Extraction of child projects algorithm

4.3 Configuration of Virtual Machine, Apache, Custom Django Commands

In order to deploy our system to the world wide web, we had to use a trusty web server. The first one that came to mind was Apache. Idea was stuck there to stay, so the website was hosted on

Apache2. In order to make Apache work with Django and other python made websites, an Apache package called mod wsgi [10] was a necessity. In order to make the website work properly, some configurations needed to happen. First of all, since it was time to deploy not in development mode, Django would stop serving static files to the website. This meant that Apache itself should do it, or that another webserver should be created in order to serve static files to the system. This meant customizing the configurations, which can be seen in figure 29

```
Alias /static /var/www/html/static
<Directory /var/www/html/static>
    Require all granted
</Directory>

<Directory /var/www/html/ClapTodoist>
<Files wsgi.py>
    Require all granted
</Files>
</Directory>

WSGIDaemonProcess var/www/html/ClapTodoist python-home=/var/www/html/ClapTodoist/myprojectenv python-path=/var/www/html
WSGIProcessGroup var/www/html/ClapTodoist
WSGIScriptAlias / /var/www/html/ClapTodoist/wsgi.py
```

Figure 29. Apache2 configuration file

Starting from the beginning, first we make sure that Apache can access our static files. For this reason, we give Apache access to static directory of the website. Next up, it is needed to give Apache access to wsgi python file, which holds an WSGI application project. As a good practice, it is a good idea to use the mod wsgi "Daemon mode" as it creates a daemon process group and delegates the Django instance to run it. In order to do that, one needs to add a WSGIDaemonProcess and WSGIProcessGroup directives. Last but not least, the WSGIScriptAlias is the base URL path that we want to serve our application at (root url) and the location of wsgi python file. This tells apache to serve any request below the given url using the WSGI application which is defined in the wsgi.py file.

Next steps after configuring the virtual machine and Apache web server were to implement a cronjob in order to make daily syncs for our users. This at first was quite problematic due to the fact that a sync happens using three different python functions that all require Django. This, however, was built around using custom django-admin commands. The flow of creating a cronjob is fairly simple, therefore main focus will be put on the management command itself. The process of creating a custom command is fairly simple too. First off, a special folder management/commands must be created in order to make Django recognize what kind of function it will be. Then, a python script named after your function must be created. In it, one must define their command which executes their own custom code. In our case, the code goes through every user that possesses an API key and commits a synchronization action for each and every one of them. This way, even if the user is not using our system fairly often, the system still can present various graphs and information important to the user when he comes back to visit. After creating a custom command, one can call it using manage.py file. This way, the command is then passed to a daily crontab and a sync occurs on daily basis.

4.4 Heuristic Evaluation of the Website

The ClapDoist system does not implement all the heuristics that were developed by Jakob Nielsen. It does, however, implement some of them and the ones implemented will be mentioned and explained shortly.

First off, the website is made simple, easily usable, understandable. It is made for everyone who uses Todoist and knows english, meaning there are no difficult phrases or concepts which would be hard to understand for an everyday person. In addition to that, information on the site appears just like it does in Todoist making it more readable for the users and easier to understand. This corresponds to the second principle of matching between system and the real world.

Next, according to the heuristic of user control and freedom, various go back and navigation buttons have been added. As far as "Undo" goes, our system supports backups of tasks that the user accidentally deleted in the main Todoist system.

Every button does exactly what is written on them and there are no different buttons which do the same thing, therefore fourth heuristic of consistency is also approved.

Fifth heuristic of error prevention is enforced in the site. However, it is most likely not perfect and there are simply unknown situations which would definitely crash the system.

We believe that eighth heuristic of aesthetic and minimalist design is also implemented in the system.

Documentation of the system is provided as a report. However, there are little to none "Help" windows therefore the tenth heuristic is not fully implemented.

Conclusions and Recommendations

ClapDoist system which extends Todoist project management system was a beneficial experience for the group members, as knowledge about Todoist API, Django web framework and databases was gained and improved. The system could be applied perhaps not officially, but it could definitely be used by a group of people to help them get a better idea about their projects and their work flow. A project like this still has a lot it can do in order to improve and turn into a perfect product which could be used by everyone who owns a Todoist account. Currently, ClapDoist system has various functionality which would be beneficial for Todoist itself. First off is the possibility to return deleted tasks . Next is the possibility to have a calendar of their tasks without having to integrate your account with widely used calendar plugins such as Google Calendar. In addition to those two, ClapDoist system has some data analysis and visualization tools which produce graphs and charts that give user detailed information about their overall status of projects and tasks.

Although this system could be really useful when used with Todoist, its problem is that changing it to work with other system would be really difficult. That is due to the fact that most of the graphs, tables would be difficult to complete in addition to the fact that database itself would overgo a crisis. In addition to that, the system itself could be changed to feel a bit more responsive as some functions such as syncing with the API may take a long time for a user with hundreds of tasks and there is currently no way to differentiate whether the site is loading or whether the server is offline.

References

- [1] Klipfolio system that uses todoist api for retrieving data with various requests, 2018.
<https://www.klipfolio.com/integrations/todoist-dashboard>.
- [2] Microsoft. Microsoft project information and purchase page, 2018.
<https://products.office.com/en/project/project-and-portfolio-management-software>.
- [3] Stewart Butterfield. Cloud-based set of proprietary team collaboration tools and services(slack), 2018.
<https://slack.com/>.
- [4] Todoist project management system, 2018.
<https://todoist.com>.
- [5] Django Software Foundation. Django framework for website developement documentation, 2018.
<https://docs.djangoproject.com/en/2.0/>.
- [6] Todoist. Application programming interface documentation for todoist project management system), 2018.
<https://developer.todoist.com/sync/v7/>.
- [7] Plotly javascript library documentation, 2018.
<https://plot.ly/javascript/>.
- [8] Plotly. Dash framework for data visualisation user guide, 2018.
<https://dash.plot.ly/>.
- [9] Fullcalendar javascript package for making calendars documentation, 2018.
<https://fullcalendar.io/docs>.
- [10] Mod wsgi package for apache server documentation, 2018.
<http://modwsgi.readthedocs.io/en/develop/>.