Development of biotechnology data processing service

Martuslović, Marin

University of Rijeka – Faculty of informatics and digital technologies

Undergraduate study program in informatics

Marin Martuslović

Development of biotechnology data processing service

Bachelor’s thesis

Mentor: Prof. dr. sc. Ana Meštrović

Rijeka, August 2022

FAKULTET INFORMATIKE I DIGITALNIH TEHNOLOGIJA
SVEUČILIŠTE U RIJECI

Rijeka, 4.7.2022.

Zadatak za završni rad

Pristupnik: Marin Martuslović

Naziv završnog rada: Razvoj sustava za procesiranje podataka iz područja biotehnologije

Naziv završnog rada na eng. jeziku: Development of biotechnology data processing service

Sadržaj zadatka:

Zadatak diplomskog rada je razviti nadogradnju postojećeg sustava za procesiranje numeričkih podataka o aktivnostima vinskih mušica (*D.melanogaster*). Sustav je potrebno nadograditi funkcionalnostima koje omogućavaju korisniku upload podataka, filtriranje i obradu podataka te vizualizaciju podataka.
U radu je potrebno opisati postupak implementacije sustava i alate koji su korišteni u implementaciji.

Mentor

Prof. dr. sc. Ana Meštrović

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 4.7.2022.

Marin Martuslović

(potpis pristupnika)

Abstract

In this thesis, I developed an application called Biostat for processing raw data based on the *Drosophila melanogaster* (*D. melanogaster*) activities. A web-based software solution is presented to extend an existing tool's functionality including filtering, accumulating, and visualising data. It was applied to a particular use case wherein its data contained recorded sensor data for the activity of *D. melanogaster* over time, which varied both during the day and between days. However, the software's utility is widely generalizable, easy to scale, and applicable to many other use cases. The end-result web application offers an input form and file to upload, after which it converts the data to the pandas' dataframe format and displays a graph. Some limitations of the application are discussed, including the scarce time period in which a full-fledged solution, one which would cover all necessary functionalities, could not have been made, substantially due to a lack of testing and time it would take to fully implement. This thesis contributes to the understanding and usage of existing tools for researchers working in the field of biotechnology with Python.

Table of contents

1. Introduction

Conducting experiments in biotechnology is a task that combines traditional methods of research with modern technologies to improve human-made designs in various fields and advance technology altogether. This makes biotechnology an important research field in the 21st century with highly inspiring and challenging tasks.

To overcome many of the challenges associated with these methods, students of biotechnology must become proficient in many different areas of knowledge. Because of that it is important to facilitate this process as much as possible.

Often, the language of choice for students doing research will be Python due to its extensive list of available libraries in the field of data science. In order to operate Python scripts, one must learn the concept of computer programming, the associated syntax with a particular language, and the particular syntax of the chosen language library. It requires many years of learning and has a comparatively steep learning curve. It is thus sometimes debilitating in early research if the students' knowledge in these otherwise instrumental rather than terminal areas is insufficient.

The motivation for writing this thesis is to provide a tool for students of biotechnology to easily operate and continue carrying out important research in their field and further contribute to science. The solution is first and foremost practical, but carries with it explanations and instructions that teach the technologies used in detail as well.

I developed a web tool named Biostat for this purpose. The tool is, more concretely, data-processing software for tracking the activity of *Drosophila melanogaster*. *D. melanogaster* shares upwards of 60% of DNA with humans [1], and is hence pertinent to research on people. Raw data of its movement is given in a TXT file, collected by the DAM5M *Drosophila* Activity Monitor, and processed by the web-based application using the pandas' dataframe data format stored in Python variables. The application contains an input form and elements of the backend and frontend web development, and then displays it as a graph using the Python's Matplotlib library, with additional filtering methods. The animal's activity is thus immediately simplified, and easy to spot patterns in due to visualization—all while providing an intuitive graphical user interface without any necessary knowledge of programming or Python.

More related work about this topic can be found in research such as Seong et al. 2020 [2].

The rest of this thesis is organized as follows. In the next sections we describe the DAM5M *Drosophila* Activity Monitor, software requirements, the description of the software's development, and instructions. Further, limitations, challenges, and methods are discussed, as well as possible improvements.

2. Domain Description

2.1. DAM5M *Drosophila* Activity Monitor Data

This application is created for processing raw data from DAM5M. DAM5M Drosophila Activity Monitor is the sensor that measures the locomotor activity of 32 individual flies in tubes 5 mm wide in diameter and 65 mm in length. The way it does this is by containing 4 infrared beams which are interrupted when the fly walks back and forth, leaving thus an exact and detailed record of time and location history inside the tube, including dwelling time. Each tube is held firmly in place by the device, and due to its relatively small size, it can be carried and operated anywhere.

The device generates the following data for each individual fly:

- Counts (integer numbers greater than zero)
- Moves (integer, number of moves)
- Dwelling time (percentage, value between 0-100)
- Resting time (integer, number of seconds without movement)
- Position data (integer, indicates which beam was most recently moved into)

Currently the application supports data only from DAM5M sensors, but in later updates the data parsing part of the algorithm could be generalized to support data from various sensors.

2.2. Problem setup

The key objectives of the solution provided in this thesis focus on three main functionalities and methods—namely: (i) creating a graphical user interface (GUI) for researchers with little or no experience in the Python programming language, (ii) providing an optimal and appropriate tool for the given use case with thorough and documented research, (iii) parsing data and implementing already available solutions to plot graphs based on the given data.

2.2.1. Problem 1: Creating an apt GUI

The target user may not be familiar with Python, and may not be particularly proficient at using it. Providing a solution in this regard will save them indispensable time and consequently provide better research results.

The GUI has to be fast, intuitive, and generally simple to use.

2.2.2. Problem 2: Seeking the optimal tool

The created application should be easily accessible from any computer environment regardless of the operating system installed on the machine. Moreover, the programming language chosen for the project must be compatible with already existing Python scripts. The GUI must be simple and in line with relevant and modern UI standards.

These considerations might appear to be fairly undemanding at first glance, but there are no specific tools or frameworks that resolve given requirements all at once. A web application approach seemed like the best fit for the task given the universal nature of browsers and their unanimous and congruous adoption standards.

Furthermore, there are no publicly available open-source tools to host Python scripts, which is why I choose Django. Django offers the opportunity of using all the Python libraries in conjunction with the features of JavaScript, CSS, and HTML.

2.2.3. Parsing data and plotting graphs

Pandas [3] already offers solutions to calculate the mean and sum of a data set and plot time series charts, but the input data must be parsed and prepared in a form suitable for pandas to use. Parsing algorithms should be created to prepare the data.

Once the data is ready, Pandas functions can be used to alter the data that will later be used to plot time series line charts.

3. Environment setup

The initial development environment setup is relevant for a suitable user experience. Likewise, it gives the software engineer an enhanced ability to modulate tasks, deploy, scale, and maintain the project [4]. Scaling, in particular, relies on the environment and is arguably one of the most important aspects to keep in mind.

As briefly mentioned, the solution of this thesis will be created and prepared using Django—an open-source, fullstack web application development framework using Python. Django was made using the model–template–views architectural pattern, and as such is fitting for this project [5].

The environment will be prepared as per instructions in the following manner (see fig. 1):

- Configure a local Visual Studio Code (VSCode) project with GitHub
- Connect the GitHub repository to Heroku
- Deploy the application

Figure 1 Environment setup workflow

3.1. VSCode and Github

It is a valuable piece of advice for software developers of any kind to keep diligent track of their changes. Thus, code changes will be recorded using Git—the de facto standard version control software in the computer science industry at large. A remote repository will be opened on GitHub for this project. Actions will be configured to directly interface with VSCode's version control GUI.

After creating a Django project with the django-admin command utility, a Python virtual environment will be created such that the pertinent Python interpreter, libraries, and scripts installed are isolated from the global operating system environment as a whole, as well as any libraries already installed in it. After activating the virtual environment, the repository can be initialized in the Source Control panel and the first changes can be committed to the main

branch which will then be pushed to a new private or public remote repository. (Note: this main branch will be later used by Heroku).

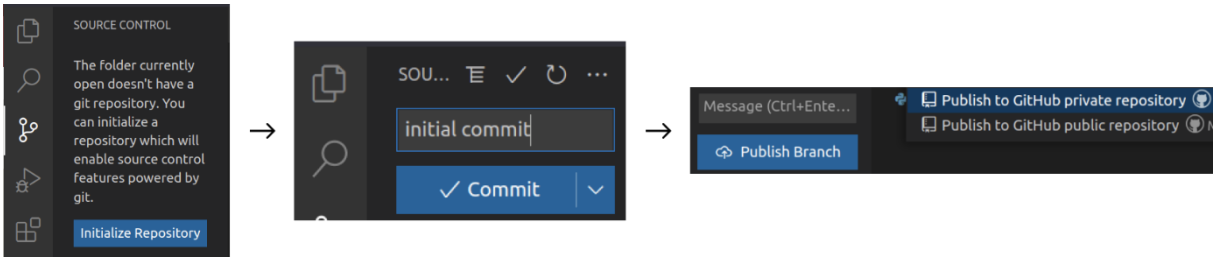

Figure 2 Initializing a Git repository

3.2. Heroku

3.2.1. Setting up the Django project for Heroku deployment

Since the service will be available online, the project needs to be hosted on a cloud platform. In this case, the platform of choice is Heroku [6]. Heroku is a popular cloud platform that supports multiple programming languages. As of now, Heroku offers a free tier option, which makes it great for the early part of the app's lifecycle when there are far more pressing problems. In this case, Heroku will be used to display a proof of concept that later may be easily released to production [7].

There are a few steps necessary to get this application ready for deployment. It needs a web server gateway interface to deal with actions and events between the web server and the web client, the software solution. A popular choice that will get it to run on the server is Gunicorn [8]. Gunicorn communicates with multiple servers, reacts to web requests, helps to distribute the load and the application traffic across multiple servers, and keeps multiple processes running. All the required packages can be installed with the Python package installer pip.

```
$pip install gunicorn
```

Figure 3 Installing Gunicorn

After installing the required Gunicorn packages, the application needs a runtime file that tells Heroku what version of Python is used and a Procfile that specifies all the commands that are

executed by the application on start-up [9]. Usually, the Procfile is used to declare a variety of process types, such as:

- The application's web server.
- Multiple types of worker processes.
- A singleton process, such as a clock. The reason for this is that some apps need to run jobs at scheduled times. For example, an application might need to make a request to an API every few minutes. In the case of this application, it might be used in a future release where it pulls data directly from the sensor at a scheduled time instead of reading the data from a file.
- Tasks to run before a new release is deployed.

Inside the Procfile we need to declare a process type with the following format:

```
<process type>: <command>
```

Figure 4 Procfile content

- <process type> is a name for the command such as relating to the web server and worker.
- <command> indicates the command that every dyno of the process type should execute on the startup.

```
web: gunicorn bioStat.wsgi
```

Figure 5 Runtime file content

The next step is setting up the project's static files. Normally, most projects will have some static files, including images, CSS scripts, or JavaScript files that are stored in a folder called "static". Considering that our project will use these types of files, a new directory called "static" needs to be created to store all JavaScript and CSS files.

In order to serve static files, a piece of software called WhiteNoise needs to be installed that will enable us to do so. WhiteNoise allows web apps to serve their own static files, making it a self-contained unit that can be deployed on various services without relying on any other

external service [10]. It is designed to work with a content delivery network (CDN) for high-traffic sites so it does not require sacrificing performance for simplicity.

```
$pip install whitenoise
```

Figure 6 Installing whitenoise

After installing WhiteNoise with pip, global variables need to be declared in the settings.py file that will enable the use of the WhiteNoise features.

```
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
STATIC_URL = '/static/'
STATIC_DIRS = os.path.join(BASE_DIR, 'static')

STATICFILES_STORAGE = "whitenoise.storage.CompressedManifestStaticFilesStorage"
```

Figure 7 Staticfiles variables

The static root is the single root directory from which the Django application will serve static files in production. As part of deploying the application, ./manage.py collectstatic needs to be run to put all the static files into the root of the Static file. Heroku does this automatically without explicitly running the command. The static root line works only in production mode, i.e. when the debug variable is set to “false”.

WhiteNoise comes with a storage backend that automatically takes care of compressing files and creating unique names for each version so they can be safely cached. To use it, STATICFILES_STORAGE needs to be defined as shown in the code snippet above. To complete the integration, WhiteNoise needs to be added to the project middleware like this:

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    "whitenoise.middleware.WhiteNoiseMiddleware",
    ...
]
```

Figure 8 Application settings file content

The last item that has to be added to the project is a file that will tell the Heroku server what application dependencies and libraries are needed for the application to run.

```
$pip freeze > requirements.txt/
```

Figure 9 Exporting dependencies

3.2.2. Deploying the application using Heroku

In the Heroku dashboard, a “new app” needs to be made.

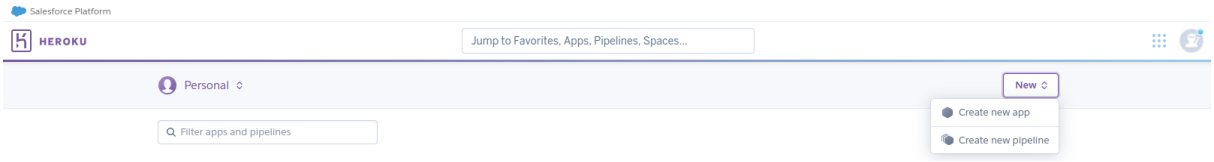

Figure 10 Heroku dashboard

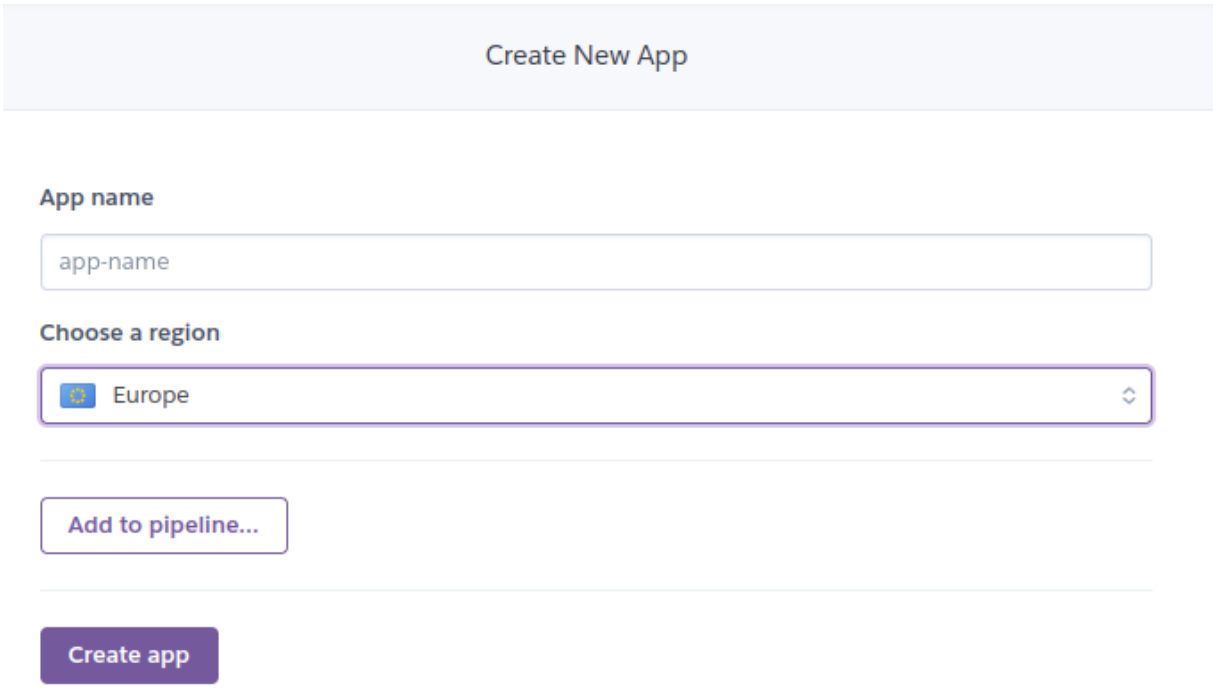

Figure 11 Creating a new application on Heroku

Once the application is finished creating, three deployment methods are offered:

- Heroku Git

- GitHub
- Container Registry

As the project described in this thesis is connected and available on GitHub, the method of choice is to deploy it with GitHub. The project will automatically redeploy with the newest changes every time a new change occurs on the master/main branch if the “Automatic Deploys” option is activated, hence it is advisable to do so.

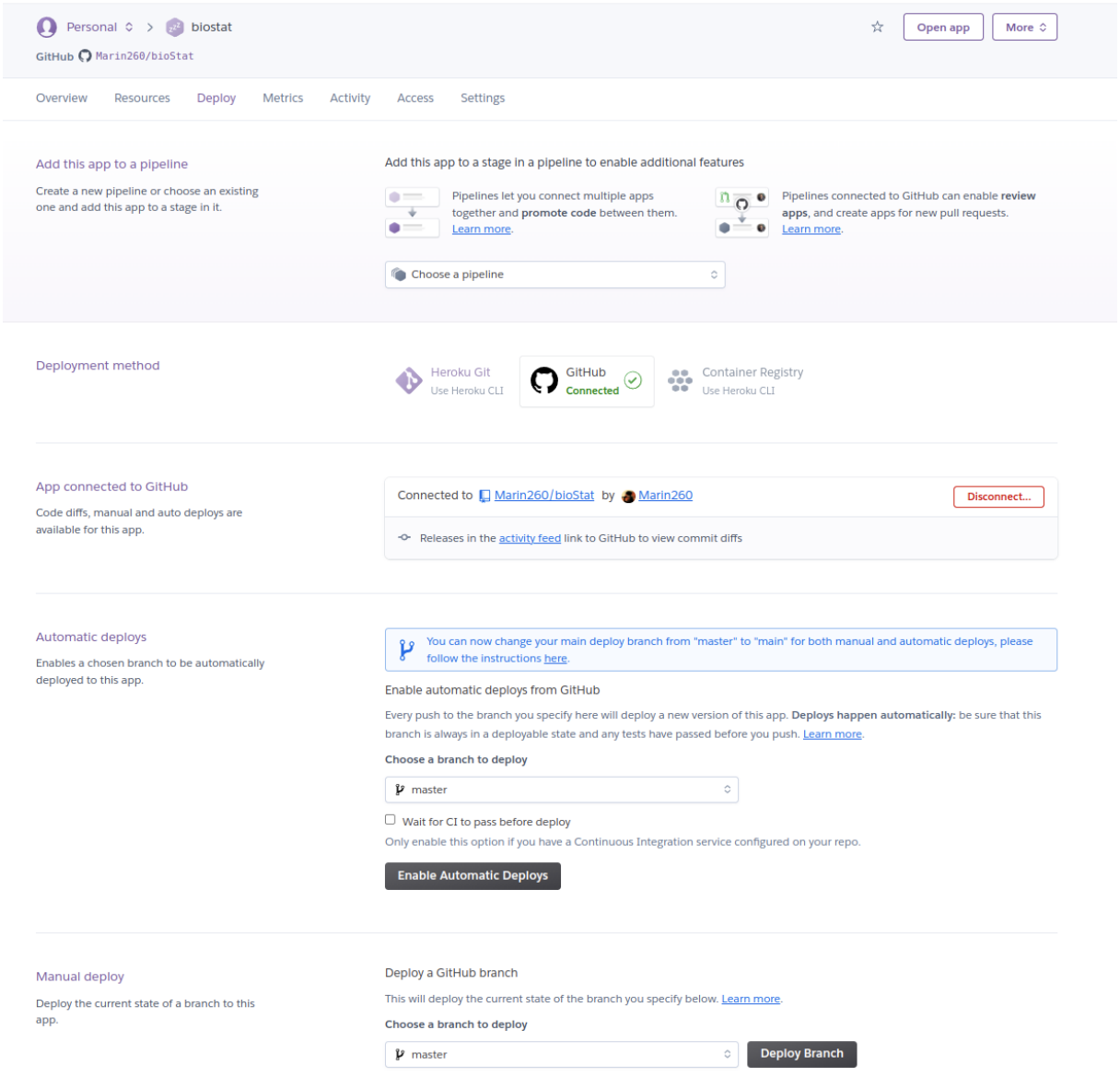

Figure 12 Heroku application dashboard

Once the deployment is successful, the application is available on the respective URL. In this case, the application is deployed at the following URL: https://biostat.herokuapp.com/

4. Web application structure and implementation

The main idea of the application is to simplify the everyday use of Python scripts in biotechnology research. I decided to use the web approach because other solutions would have incompatibility issues depending on the user's operating system.

An early straightforward structure shows the leading idea of how the application works.

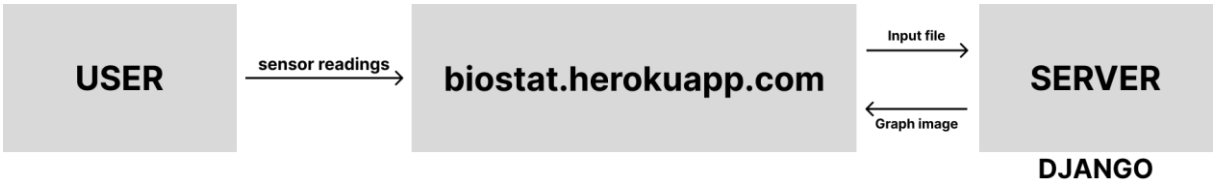

Figure 13 Application workflow

On the landing page, every user is offered a form field with a file input field and various other filter fields. The submitted file is then parsed by a python script running on the server, that returns a graph image to the client.

These were the technologies used to create the previously described application:

- Python, including the Matplotlib library, for the backend web development,
- Tailwind and JavaScript for the Django frontend web development,
- and Heroku for deployment and hosting

4.1. Implementation

The inner working of the application is divided into multiple parts that work together to create a seemingly single process:

- Frontend file parsing
- Backend file parsing
- Graph creation
- General UI Design

4.1.1. Frontend file parsing

Before submitting a file, the user needs to apply filters that will parse data in order to create the wanted dataset for the next process. Without visual feedback, data filtering gets extremely tedious.

When the user uploads a file to the browser, the content needs to be displayed to the user. This necessitates frontend parsing.

Every input file should have the same predefined form that is easily displayed in a table form. Since there are no column headers defined in the file, there is room to create custom headers that will give the user a better understanding of what they are editing.

```
const fliesInfo = ['ID', 'Date', 'Time', 'A', 'B', 'C', 'D', 'E', 'F', 'G']; // Other da
ta
const flies = Array.from({length: 32}, (_, i) => ((i + 1)+9).toString()); // flies measu
red
const tableHeaders = fliesInfo.concat(flies); // headers for the "display" data table
```

Figure 14 Creating table headers

An “on change” event listener is put in place on a file input field to make changes happen on the page. When the event listener is triggered, the file content is getting parsed and the dynamically created HTML table is filled with data.

```
const reader = new FileReader()
reader.onload = function() {
        data = reader.result // read file and store in var
        data = data.split(/\r?\n/);
        data = data.map(item => item = item.split(/\t/));
}
```

Figure 15 File reading

Date time fields are automatically filled with the calculated outer bounds read from the file.

```
let startDateDate = new Date(startDate);
let endDateDate = new Date(endDate);

let startDateInputwtf = document.getElementById("id_dateFrom")
let endDateInputwtf = document.getElementById("id_dateTo")
startDateInputwtf.value = startDateDate.toISOString().substring(0, 16);
endDateInputwtf.value = endDateDate.toISOString().substring(0, 16);
```

Figure 16 AutoFill date input fields

Upon completing frontend parsing, the result is a dashboard-like site where the users can manually select columns that they do not want to include into data processing.

4.1.2. Backend file parsing

When the server receives the file sent by the user, a Python script transforms the file content from plain text to a pandas' dataframe. The data can now be parsed and filtered with the user input field values.

```
def dataToDF(data, userInput):
    toDelete = parseRange(userInput['columns']) #columns that the user wants deleted
    df = pd.DataFrame.from_records(data)

    #create a dateTime column by concatenating the date and time column
    df['datetime'] = df.iloc[:, 1] + ' ' + df.iloc[:, 2]
    df['datetime'] = pd.to_datetime(df['datetime'])
    df = df.set_index(df['datetime'])

    #eliminate unnecessary columns
    df.drop(df.columns[[0, 1, 2, 3, 4, 5, 6, 7, 8, 29]], axis = 1, inplace = True)
    df.drop(toDelete, axis = 1, inplace = True) # delete selected columns

    #filter data by date
    df = df[~(df['datetime'] > np.datetime64(userInput['dateTo']))]
    df = df[~(df['datetime'] < np.datetime64(userInput['dateFrom']))]
    df = df.drop(columns=['datetime'])
    df = df.astype(int)

    #use the user selected function
    if userInput['selected_function'] == "MEAN":
        df = df.resample(userInput['sampling']).mean()
    elif userInput['selected_function'] == "SUM":
        df = df.resample(userInput['sampling']).sum()
```

Figure 17 Parsing data with Python

4.1.3. Graph creation

The final step is creating a graph with the parsed dataframe and then sending it back to the user. The Matplotlib Python module is used to create the graphs, but due to the fact the image cannot be sent back directly, it first needs to be stored in a buffer as bytes that will later be decoded.

```
def get_graph():
    buffer = BytesIO()
    plt.savefig(buffer, format='png')
    buffer.seek(0)
    image_png = buffer.getvalue()
    graph = base64.b64encode(image_png)
    graph = graph.decode('utf-8')
    buffer.close()
    return graph

def get_plot(df):
    plt.switch_backend('AGG')
    plt.figure(figsize=(10,5))
    df.plot.line()
    graph = get_graph()
    return graph
```

Figure 18 Creating and loading graphs

```
{% if graph %}
    <img src=" {{ graph|safe }}" alt="">
{% endif %}
```

Figure 19 Loading created graph

See Appendix A for the applications full code.

4.1.4. General UI Design

The UI design follows modern design standards and places an elementary focus on readability and intuition. The application consists of three states:

- Idle state (See Fig. 20.)
- Loaded file state (See Fig. 21.)
- Result page state (See Fig. 22.)

In the idle state the application waits for the user to input a file, until then an animation is looped and a message with instructions is displayed to the user. The user can already apply filters although they can't submit a request to the server until a file is loaded.

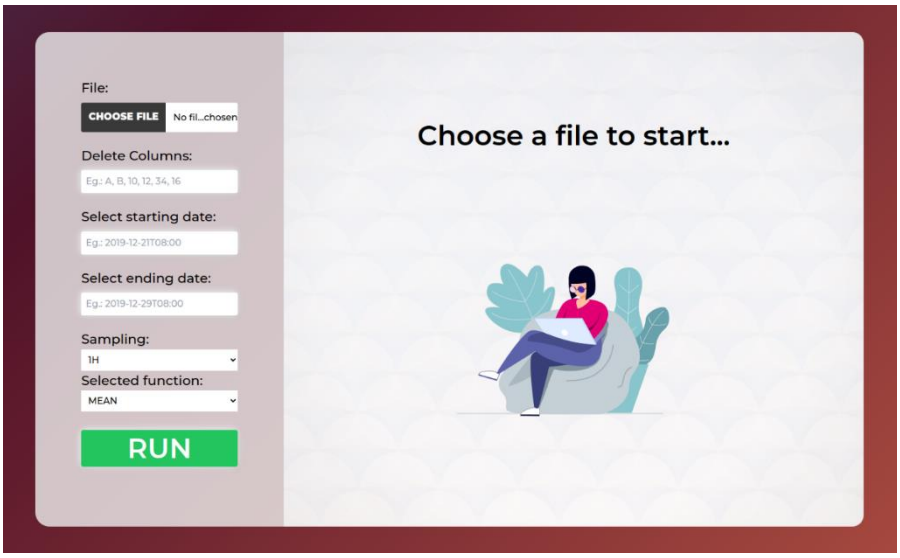

Figure 20 Application idle state

Once a file is loaded, the idle screen is replaced with an interactive table and date-time filters are updated with the date-time limit values from the file. The user can now click on the table headers to select columns instead of typing them manually.

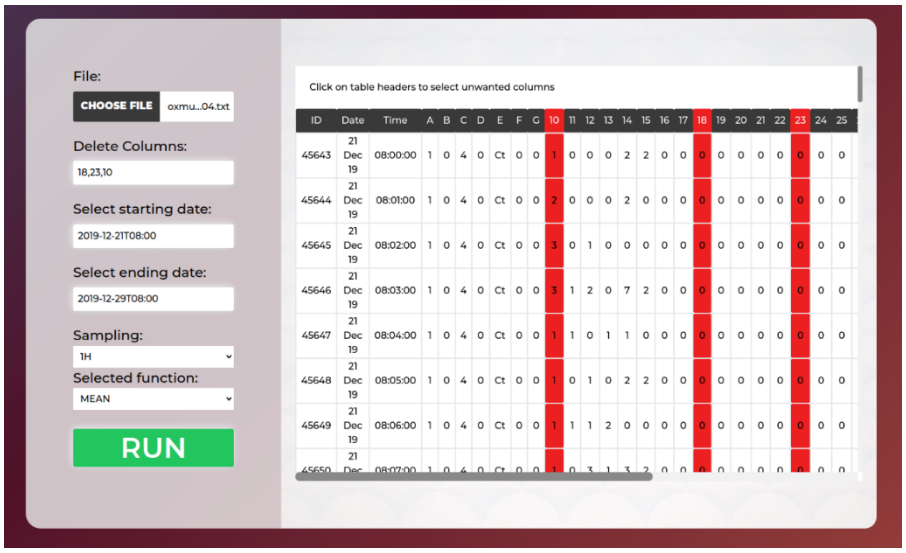

Figure 21 Loaded file state

After submitting a request, the application processes the data and returns a graph image that replaces the table. The user can normally upload a new file that then puts the application back in the loaded file state. This cycle can continue for as long as the user needs it to.

Figure 22 An example of the result page

5. Limitations

Developing web application for biotechnology data processing is a challenging task. Within this thesis I have covered development of the basic system features. However, this application can be further extended. There was not enough time within the given time frame to create a full-fledged application that would include many features that would need to be tested extensively, such as complete accessibility, but also additional features related to frontend development.Other limitations of the application include:

- Inability to filter data twice using the same file: the page needs to be refreshed after each form submission, and the user has to return to the initial page state before refreshing the page.
- The current project does not support other file reading formats from other sensors, such as DAM4M.
- The application does not have an informative or landing page that would tell the user more concretely the background of not only how to use the application, but who the author is and how to contact them.
- No additional file formats are supported, such as XML. To scale, this application would need other commonly accepted data file format standards.
- Limited functionality in the drawn graph: it would be far simpler for the user to navigate their data if they could get information about a particular data point on the graph upon hovering over it. Likewise, it would be beneficial if the user could determine the width and height of their graph, and export it as an image so that it can be used in a research project.

5.1. Possible future improvements

Many of the above mentioned limitations will be addressed in future improvements. However, it is likely that depending on the success of the application, not all of them can be covered, as some would likely require significant restructuring and the use of different technologies. For example, the Matplotlib library may not be sufficient to meet all the requirements of mentioned improvement requirements.

However, improvements such as adding the landing page, an about page, a redesign of the submission form, and adding compatibility with other sensor reading formats are bound to occur

in the future. These will significantly improve the user experience of the software and provide it with a palpable increased potential for scalability.

6. Conclusion

In this thesis, I describe the development of the web application called Biostat for processing the raw data of *D. melanogaster* activities. The application is based on web technologies Django and Heroku, and programming languages Python and JavaScript, as well as frameworks such as TailwindCSS [11].

This application offers users an analysis of their data without any knowledge of Python thanks to a modern and intuitive GUI. They can simply click and select rows they want to filter out, dates they want to consider, and choose the sampling methods and functions relevant to their research. Afterwards, a graph is generated to visualize the data. This way, the application offers them a much faster analysis and eliminates the immediate necessity of high proficiency in the programming language Python.

Despite many initial challenges in finding and creating an optimal solution for this use case, and finding a way to implement it, a solution was indeed found and proved to be sufficient to complete all the necessary requirements.

With further improvements, this application could easily scale and apply to various other projects, and could, in fact, be widely adopted. At the moment, it is tailored to the DAM5M measurements of *D. melanogaster* activities As such, it solves an important problem, and it is exciting to think about its potential.

7. List of images

8. Literature

[1] "Why do scientists investigate flies," [Online]. Available: https://www.mpg.de/10973625/why-do-scientists-investigate-fruit-flies.

[2] B. C. o. M. U. S. Hugo J Bellen, "The Drosophila Individual Activity Monitoring and Detection System (DIAMonDS)," 10 Noember 2020. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7655107/#article-1aff-info.

[3] Pandas, "User Guide," [Online]. Available: https://pandas.pydata.org/docs/user_guide/index.html#user-guide.

[4] A. Marget, "Development and Test Environments: Understanding the Different Types of Environments," [Online]. Available: https://www.unitrends.com/blog/development-test-environments#:~:text=One%20of%20the%20most%20important,controlled%20setting%20without%20impacting%20users..

[5] S. Jaiswal, "Django MVT," [Online]. Available: https://www.javatpoint.com/django-mvt.

[6] Heroku, "Heroku Products," [Online]. Available: https://www.heroku.com/products.

[7] Heroku, "Configuring Django Apps for Heroku," 28 July 2022. [Online]. Available: https://devcenter.heroku.com/articles/django-app-configuration.

[8] G. Developers, "Gunicorn - WSGI server," [Online]. Available: https://docs.gunicorn.org/en/stable/.

[9] Heroku, "The Procfile," 16 February 2022. [Online]. Available: https://devcenter.heroku.com/articles/procfile.

[10] Heroku, "Django and Static Assets," 28 July 2022. [Online]. Available: https://devcenter.heroku.com/articles/django-assets.

[11] Tailwind Labs, "Get started with Tailwind CSS," [Online]. Available: https://tailwindcss.com/docs/installation.

[12] Django Software Foundation, "Django documentation," [Online]. Available: https://docs.djangoproject.com/en/4.1/.

[13] M. D. John D. Hunter, "Users guide," [Online]. Available: https://matplotlib.org/stable/users/index.html.

9. Appendix A

The code is stored on a publicly available GitHub repository and can be found on the following URL:

https://github.com/Marin260/bioStat