

# UNIVERSITY OF GRANADA

MAJOR IN COMPUTER SCIENCE

---

## GeneSys

---

A BIOINFORMATIC TOOL FOR GENOMIC DATA ANALYSIS

---



**Author:** Bruno Otero Galadí

**Supervisor:** Dr. Fernando Berzal Galiano



UNIVERSIDAD  
DE GRANADA

---

**ETSIIT**  
Escuela Técnica Superior  
de Ingenierías Informática  
y de Telecomunicación

---



August, 2024. Granada, Spain.



# Genesys: A bioinformatic tool for genomic data analysis

Bruno Otero Galadí

**Keywords:** reverse transcriptase, amino acid, molecular biology, nucleotide, protein, DNA, RNA, fasta, PATRIC, Biopython, BV BRC, presence-absence matrix

## **Abstract:**

Recent decades' advancements in biological research have brought numerous benefits to our understanding of nature and society's progress. However, these advancements have also generated a vast amount of biological data that must be processed quickly to remain valuable for researchers. If the required speed in processing this data is not achieved, it could become a bottleneck, potentially slowing the current rate of scientific discoveries.

The majority of the problems are related to the preprocessing of big amounts of biological data stored in public databases, which are continuously updated to locate more and more examples of genomic information coming from all kind of sources. So, if researchers without advanced programming knowledge want to dive into these databases in search for a specific kind of genomes, they must be able to manipulate the data in a way that allows them to repeat the process with as many parameters as needed. Additionally, researchers may need to process the data through a series of tasks that must be separated and executed sequentially. This is where GeneSys comes into play.

GeneSys is a modular and scalable software tool with a user-friendly interface that allows researchers to define tasks within a workflow that can be executed and redefined freely in order to satisfy their researching needs, regardless of complexity.

The GeneSys software is designed to have a basic first layer that defines how tasks and workflows are related to each other. This structure allows developers to create modules that would address specific problems. This work includes an initial module designed to solve a real life issue involving reverse transcriptases, also known as RTs, a unique kind of proteins with significant research potential, many aspects of which re-

main unexplored. Such proteins are currently being studied by Dr. Francisco Martínez-Abarca Pastor at La Estación Experimental del Zaidín (EEZ) in Granada, Spain. The implemented module will help Martínez-Abarca to efficiently face his investigations involving RTs.

# Genesys: una herramienta bioinformática para el análisis de datos genómicos

Bruno Otero Galadí

**Palabras clave:** reverso transcriptasa, aminoácido, biología molecular, nucleótido, proteína, ADN, ARN, fasta, PATRIC, Biopython, BV BRC, matriz de presencia-ausencia,

## Resumen:

Muchos avances se han dado en las últimas décadas en la investigación biológica, todos ellos aportando progresos en la comprensión de la naturaleza y en el desarrollo de la sociedad. No obstante, estos avances han provocado la necesidad de procesar cada vez más datos biológicos a un ritmo que debe permanecer constante para resultar rentable. Si dicha eficiencia en el procesado de datos no se alcanza, existe el riesgo de que se convierta en un cuello de botella que, llegado el momento, reduja el ritmo con el que se han producido avances en esta materia hasta ahora.

La mayoría de los problemas van de la mano al preprocesamiento de información genética contenida en diversas bases de datos, que además se incrementa en volumen con el paso del tiempo, a medida que se descubren nuevos genomas. Cualquier persona investigadora que carezca de un nivel alto de programación y desee emplear información de una base de datos para acometer una tarea va a necesitar disponer de un mecanismo que le permita repetir el proceso aplicado a los datos tantas veces como desee, así como subdividir el trabajo a realizar en tareas distintas, en caso de que quiera separarlas en el tiempo y ejecutarlas una a una. Es aquí donde entra GeneSys.

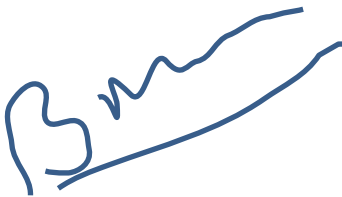
GeneSys es una aplicación modular y escalable con una interfaz de usuario fácil de usar, enfocada en ayudar en las tareas de investigación de datos biológicos. Permite a un usuario general definir tareas dentro de un flujo que podrá ejecutar y modificar según sus necesidades.

GeneSys incorpora una capa software básica que define la forma en la que las tareas y los flujos de tareas se relacionan en la aplicación. Partiendo de ahí, es posible implementar módulos personalizados e independientes que acometan tareas según las necesi-

dades específicas de las investigaciones que se estén llevando a cabo. Este trabajo, además de la capa básica, incluye un módulo diseñado para resolver un problema de pre-procesado de datos relativo a las reverso transcriptasas, también conocidas con RTs, un tipo de proteínas con un potencial investigador enorme de las que aún no se conoce mucho. El doctor Francisco Martínez-Abarca Pastor de la Estación Experimental del Zaidín (EEZ) de Granada, España, se encarga en la actualidad de estudiar dichas proteínas. El módulo implementado le servirá para progresar en sus investigaciones.

I, **Bruno Otero Galadí**, scholar of the **computer science** university degree at the “**Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**”, with a Spaniard national identification number of **75574203K**, authorize the placement of the present work at my school’s library so it can be consulted by anyone who wishes to.

Signed: Bruno Otero Galadí

A handwritten signature in blue ink, appearing to read 'Bruno', with a long, sweeping underline.

Granada, on September the 1<sup>st</sup> of 2024.

---

Mr. **Fernando Berzal Galiano**, teacher of the Computing Science and Artificial Intelligence Department of the University of Granada.

**Informs:**

That the present work entitled as **Genesys: A bioinformatic tool for genomic data manipulation**, has been realized under his guidance by Bruno Otero Galadí, and authorizes the defense of the aforementioned work under the collegiate tribunal that might correspond.

And so that it is stated, he issues and signs the present invoice in Granada on <month> the <day> of 2024.

**Supervisor:**

---

**Fernando Berzal Galiano**



## Acknowledgements

This work would have never existed without my supervisor, Fernando, whose suggestion to focus on bioinformatics was crucial in shaping the direction of my research. Additionally, I would not have been able to discover the significance of reverse transcriptases (RTs) and the reasons for their study without the assistance of Francisco Martínez-Abarca Pastor, a former researcher at the Estación Experimental del Zaidín (EEZ) in Granada, Spain. Francisco asked me to help him facing the RTs issue involving the preprocessing of amino acid data from the Bacterial and Viral Bioinformatics Resource Center (BV-BRC) online database. His role as a client in this work is the very reason it came into existence.

Furthermore, I want to give sincere thanks to Antonio Quesada Ramos, my former high school biology teacher, who facilitated my connection with Francisco. He is also the reason why I am so interested in biology as a field of research.

Finally, all of the time and resources I have dedicated to this matter are direct merit of my family —Dulcinea, David and Leonardo— whose support and understanding provided me with all the space I needed in order to achieve the main goals of this work. Their encouragement has been indispensable. So, thank you.

# MAIN INDEX

## 1. INTRODUCTION

1.1. A brief overview of molecular biology.

1.2. Reverse transcriptases.

1.3. When more is less. Current problems involving genomic databases.

1.4. Memory structure.

## 2. OBJECTIVES

## 3. PLANNING

3.1. Development steps.

3.1.1. Researching phase.

3.1.2. Requirement analysis phase.

3.1.3. Design, implementation and testing phase.

3.1.4. Deployment and evaluation phase.

3.2. Estimated budget.

3.2.1. Human resources.

3.2.2. Hardware resources.

3.2.3. Software resources.

3.2.4. Indirect costs and materials.

3.2.5. Total budget.

## **4. PROBLEM ANALYSIS**

**4.1. Biological context: the prediction of unknown RTs' behavior patterns experiment.**

**4.2. The current problematic.**

**4.2.1. Tasks to accomplish.**

**4.3. Selected languages and tools.**

## **5. ARCHITECTURE AND DESIGN**

**5.1. Class diagrams.**

**5.2. Folder organization.**

## **6. IMPLEMENTATION**

**6.1. GeneSys' Kivy interface.**

**6.1.1. Screens related to all implemented modules.**

**6.1.2. Screens related to the module to process PATRIC proteins.**

**6.2. Inner logic implemented in GeneSys.**

**6.2.1. Task and Workflow, the key classes that define GeneSys' logic.**

**6.2.2. Specific tasks related to the module to process PATRIC proteins.**

**6.3. Utils folder.**

**6.3.1. Biopython related utils.**

**6.3.2. Format checking utils.**

**6.3.3. Fasta processing utils.**

## **7. GENESYS USER'S GUIDE**

## **8. CONCLUSIONS AND FUTURE WORK**

## **9. BIBLIOGRAPHY**

## IMAGE INDEX

3.1.4.1. Gantt diagram of GeneSys' development phases.

4.2.1.1 Interface of the National Center for Biotechnology Information's (NCBI) main page.

4.2.1.2. Results available for download in NCBI's website.

4.2.1.3. Screenshot of PATRIC's online tool for manually downloading nucleotide bases of a certain sequence given its bait.

4.2.1.4. An example of how PATRIC shows information about a certain fragment of a nucleotides sequence that might correspond to a valid protein.

5.1.1. GeneSys' class diagram.

5.2.1. GeneSys' folder organization.

**Note:** all the employed images have been taken/created by the author.

## TABLE INDEX

3.2.2.1. [Hardware budget.](#)

3.2.2.2. [Laptop characteristics.](#)

3.2.3.1. [Software licenses.](#)

3.2.5.1. [Final project budget.](#)

## **CODING INDEX**





# 1. INTRODUCTION

## 1.1. A brief overview of molecular biology.

If we assume selecting breeding as a form of molecular biology researching, we can affirm that genetics has been taking part in humanity's history since, at least, the Neolithic period. But it was not until nineteenth century and the appearance of Gregor Johann Mendel's works that a first theoretical basis for the principles of heredity was set. Since then, molecular biology has become one of the most developed researching fields, being continually adapted to answer new questions and to face new challenges. As a result, molecular biology went from studies about peas to relatively recent works that suggest the existence of life beyond planet Earth, always being strongly correlated to chemistry and incorporating key discoveries like the DNA structure, which also paved the way for other numerous applications such as the Polymerase Chain Reaction (PCR) with a major relevance in the understanding and diagnosing of several diseases, or the Human Genome Project in 1990<sup>4</sup>. All of this improvements have provided invaluable benefits for society. Biology and specially molecular biology are not just fields with history. They are fields with future.

Before we get deeper into biological concepts, we should remember what DNA, RNA and proteins are and how they are related to each other. DNA<sup>5</sup> and RNA<sup>6</sup> sequences are identified as sequences made up of repetitions of up to four nucleotides, represented as A, C, T, G for DNA strings and A, C, U, G for RNA ones. Apart from the composition, the main differences between both structures involve their spacial distributions (with a double-helix polymer structure for DNA and a single-stranded biopolymer for RNA) and their biological functions, with DNA serving as a codification of the genetic information and RNA using DNA to synthesize the proteins that are stored in cells<sup>7</sup>. Proteins are chemical components made of elements called amino acids. The amino acids that might be found in proteins' cells differ between species, but there are no more than twenty different amino acids that occur naturally in any living being's proteins<sup>8</sup>.

To sum up, DNA defines the genetic composition of a living being, and RNA replicates that composition in order to define the structure of proteins. But, what is the mechanism that translates RNA into proteins? The nucleotides contained in a RNA sequence (and therefore in its equivalent DNA sequence) are read in intervals of three each, and they can be read starting from anyone of the first three nucleotides that compound the aforementioned RNA

(or DNA) sequence, onward and backward, which gives us up to six different ways of getting a protein from a same RNA string. A protein is properly identified when, using one of those lecture ways, a specific set of three nucleotides that marks the end of the lecture is found. This sets of nucleotides are called stop codons, and correspond to a certain amino acid that serves as a delimiter of the protein<sup>9</sup>.

## **1.2. Reverse transcriptases.**

Reverse transcription refers to the process of turning specific RNA sequences into DNA. Not all RNA strings are valid for this issue, and those that indeed are are formally called “RNA-dependent DNA polymerases”, “reverse transcriptases” or “RTs”<sup>10</sup>. As not all RNA strings serve as RTs, they need to be specifically recognized before researchers start experimenting with them. As we have stated before, RNA sequences are translated into proteins. That process can be done backwards, too, which means that it is possible to find a specific protein that is configured by a RNA sequence that in fact is a RT. In other words, we can identify RTs by observing proteins.

Reverse transcriptases have remarkable biotechnological applications, such as molecular cloning strategies or in the field of synthetic biology. But the most important use they have provided to humanity, or at least the most widespread one, might be the detection of viral RNA in SARS-CoV-2 testings<sup>1</sup>, as they serve as a key element in the propagation of genetic elements across specific DNA structures.

Since 2020 COVID-19 pandemic, the lock-down and the infection waves, the interest in molecular biology seems to have gained so much popularity, being mentioned in the news, in social networks or even at the dinning room with our families. However, and despite the crucial role they have played throughout all these years, reverse transcriptases have not become that popular. And as researchers and diverse studies point out that pandemics would be more common in the future, it is quite clear that RTs will keep being at the spotlight of scientific investigations. The main arguments that are exposed to support the assumption of pandemics becoming more likely to happen concern topics such as climate change<sup>2</sup>, the destruction of the environment or the increasing contact between humans and disease-harboring animals<sup>3</sup>.

In a post-COVID world, it is crucial to be prepared for upcoming similar events. RTs take part in that process by playing a key function in PCRs, which play a potentially high disease detection role.

### **1.3. When more is less. Current problems involving genomic databases.**

Nowadays, biologists tend to work obtaining their genetic data from enormous public domain databases whose volume of biological information is increasing at a relatively faster rhythm than the stored datasets of other scientific disciplines, with the amount of raw data corresponding to genome sequencing experiencing the biggest growth along with exome sequencing data<sup>11</sup>. This has led to an overwhelming amount of genomic data that needs to be correctly preprocessed in order to start searching for valuable knowledge. And RTs are taking part in that problem, too, as new examples of them are being included in those databases month by month, increasing the difficulty to distinguish which are recent discoveries from those which are not, as well as requiring more complexity in the computing of all the existing samples in order to identify common patterns between them.

Francisco Martínez-Abarca Pastor is a researcher from the Estación Experimental del Zaidín (EEZ) in Granada, Spain. Among his current issues there is the exploration of RTs' datasets in search of undiscovered correlations between reverse transcriptase samples that are separated in evolutive terms. In the year 2019, he supervised a work involving RTs' written by the former postgraduate degree student Mario Rodríguez Mestre. The aforementioned work was entitled "Analysis of Novel and unexplored groups of prokaryotic Reverse Transcriptases" and consisted of the extraction of all the available datasets of RT's stored in certain databases, its preprocessing, its classification through clustering algorithms and the seeking of undiscovered common behavior patterns between the RT's contained in those clusters<sup>12</sup>.

The results of the study were considered successful, and Martínez-Abarca decided he would repeat the experiment once the databases were updated with new samples. The problem is that in order to repeat the process, all the steps of preprocessing the raw data and applying the clustering had to be done manually again, which required so much effort in terms of time to be worth, so even though Martínez-Abarca wished to repeat the study, he was not able to do so.

But if he had a tool that at least could automatize the preprocessing of the raw data downloaded from the databases just as Rodríguez Mestre did back in time, he would be capable of make the same experiment whenever he wanted, so in the long term he could exploit all the advantages RTs have. GeneSys serves as a software tool that solves Martínez-

Abarca's issue.

## 1.4. Memory structure.

In order to facilitate the reader's task, here are mentioned the main parts of this memory and what each exposes:

- **Introduction:** this chapter exposes briefly the context in which GeneSys is made. It provides a generic view of the problem as and helps those readers with basic biology knowledge to understand what reverse transcriptases are and what they are used for.
- **Objectives:** includes the objectives to accomplish with GeneSys development. Such accomplishments will be analyzed in the "Conclusions and future work" section.
- **Planning:** organization, estimated developing time and hypothetical budget it might require.
- **Problem analysis:** statement of the preprocessing tasks that GeneSys must accomplish and the biological reasons that justify why they must be done that way.
- **Architecture and design:** it provides various diagrams that explain the architecture of the application. Also, justifies why that architecture has been chosen and what requirements are crucial to satisfy.
- **Implementation:** abstract of GeneSys' coding process and the development stages that occurred while implementing the tool.
- **GeneSys user's guide:** A friendly-user manual that explains how to use the app. It is aimed to be understood by any researcher how wishes to employ GeneSys in their investigations.
- **Conclusions and future work:** it compares the accomplished objectives against those exposed in the "Objectives" section. It also proposes improvements that can be made to the application in the future.

## 2. OBJECTIVES

The main objective is to provide a functional application that can be intuitively employed by an user experienced with biological terminology but with a basic programming knowledge, which accomplishes properly the issue of RTs' preprocessing. In order to achieve the proposed goal, we can distinguish the following objectives in the development of the application:

1. To develop a bug free logic that works exactly as the user needs it to work.
2. To provide an intuitive yet attractive friendly-user interface.
3. To properly recognize which tasks must be automated by the application, so that the user receives a window to modify certain parts of the preprocessing of the RTs in order to adapt their experiments freely while not losing the automatization benefits that GeneSys provides.
4. To provide a software framework that unifies all the preprocessing tasks in an unique execution context. In other words, we do not want the user to open any other application than GeneSys to achieve the proposed preprocessing goals.
5. To make an application that does not freeze or crash when it is executing a long task.
6. To give the user the capacity to apply changes to the task that it is going to be executed, such as the pathnames where to save the results. We assume that it would be always better to give the user as much freedom as possible when defining parameters related to biological terms.
7. To properly inform users about what is happening in the preprocessing of the RTs, so that they can study the results returned at all the steps of the process and draw their own conclusions for their research.

## **3. PLANNING**

### **3.1. Development steps.**

Here are the phases that the application's development process had and how much time was required for each.

#### **3.1.1. Researching phase (April 2024).**

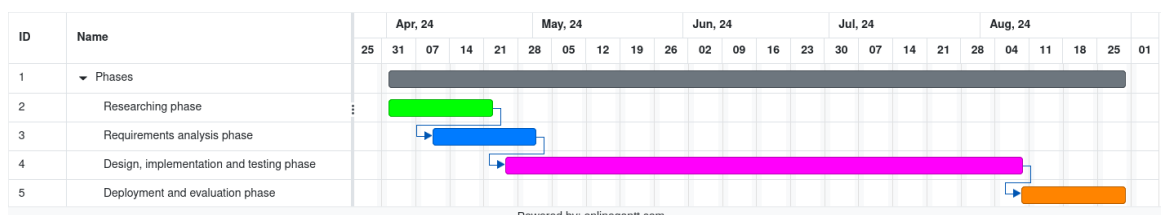
As molecular biology and reverse transcriptases were unexplored fields for the student, it was crucial for him to properly understand what they are so that he could have an approximated idea of how to bring up the interface design and the formats with which manipulate the data. Some meetings at Dr. Martínez-Abarca's office took place during the month of April, along which the student received detailed answers to all his inquiries. Among other issues, he was informed about the existence of fasta files, a special kind of format employed for representing genetic data that is used by almost every bioinformatic tool in the world. He also was introduced to Rodríguez Mestre's work<sup>12</sup> and received context about the way in which the experiment was done back in 2019. A lot of notes were taken, specially those concerning the first steps of the development. From the beginning, the application was meant to be deployed as a desktop application, as Martínez-Abarca was not interested in paying the use of some cloud services for completing the preprocessing tasks.

Finally, there was a virtual meeting between the student, his supervisor Fernando Berzal Galiano, and Martínez-Abarca, which ended with the supervisor approving the proposed project and giving his own advises and opinions about the issue.

#### **3.1.2. Requirement analysis phase (April 2024).**

The student sought for potential tools to develop GeneSys. Finally, he opted for VS code as his main coding environment, Python as the programming language with which develop the app and Kivy for the user interface.

As GeneSys was going to be a tool designed for working with genetic data, it was necessary to decide from which database the data should be taken. The selected one was PATRIC, the official public database of the Bacterial and Viral Bioinformatics Resource



Center, also known as BV BRC<sup>21</sup>. All the application would be designed in order to manipulate data downloaded from PATRIC, even though the resulting information would be returned in an universal format easily readable by any other tool.

For managing backups, a GitHub repository was created. At first, it was private, but by the end of the development it was made public<sup>20</sup>.

### 3.1.3. Design, implementation and testing phase (May-July 2024).

At first, some trial genetic data were downloaded from PATRIC database and deeply studied in order to understand the format they had. After that, the student resolved to divide the preprocessing task that GeneSys would apply to the data into smaller tasks which would be executed in a workflow.

Next, the student focused in learning Kivy in order to understand how to use it to make interfaces in Python. A prototype of interface was made, and the tasks that would be applied to the workflow were implemented by receiving their parameters from that interface. For each new task of the preprocessing that was implemented, a new Kivy screen was designed for getting its parameters.

Throughout the process, there were cyclical testings of GeneSys' made features so far. As a result, some modifications were applied to the structure of the code that leded to obtain the final design that can be found in "[architecture and design](#)" section.

Finally, once the final task to accomplish worked properly, The efforts were putted into giving a prettier design to the user interface. Some colored images were created employing different tones of green, and all the buttons and boxes of the application were decorated with those colors.

### 3.1.4. Deployment and evaluation phase (August 2024).

The ending of the project began with the redaction of the memory, and also with the final testings of the application. All the available information about the development process was gathered and included in this work. Also, the student notified to Dr. Martínez-Abarca that the

application was done and that he would be enchanted to teach him to use it.

Image 3.1.4.1, Gantt diagram of GeneSys' development phases.

## 3.2. Estimated budget.

Now, it is time to estimate how much money might a project like this cost. There are some important matters to look at when making a budget. All of them have been studied one by one in this section.

### 3.2.1. Human resources.

The next scenario will be considered:

- The development of the application takes place in **Andalusia, Spain, in the year 2024**.
- The raw monthly wage for each developer will be of **2000€** on a full time contract of forty hours per week.
- There is only **one developer**, Bruno Otero Galadí.
- The development of the application involves **five hours a day**, from Monday to Saturday, which is equivalent to thirty hours a week.
- The development will extend **from April to August**, five months in total.

Considering the above, human resources' cost will be of **1500€ of raw wage** per month, during a total of five months, 7500€.

Applying the corresponding Andalusian taxes, we get a **net wage of 1247€** per month, from which have been discounted 157€ corresponding to Andalusian "IRPF" and 96€ relative to social security. For the company that has hired the developer, a raw wage of 1500€ means to pay 471€ for each payed month corresponding to company imposing taxes, which leads to a total budget of **1971€ per month for the company**.

1971€ per month during five months corresponds to **9855€**, from which 7500€ will be the worker's raw wage, from which 6235€ will be the worker's net wage<sup>13</sup>.

### 3.2.2. Hardware resources.



The hardware tools that are going to be employed in the application's development are specified in the next table.

Table 3.2.2.1

Hardware tool	Total cost	Average lifespan	Cost for five months
ASUS VivoBook 14/15 laptop	850€	4 years <sup>14</sup>	88.54€
HP monitor	100€	15 years <sup>15</sup>	2.78€
Xiaomi Redmi Note 12 smartphone	150€	3 years <sup>16</sup>	20.83€

In total, hardware's budget rises up to **112.15 €**.

Here is another table that specifies the laptop's characteristics:

Table 3.2.2.2

Component	Characteristics
Laptop model	ASUSTeK COMPUTER INC. Vivo-Book_ASUSLaptop X421JAY_X413JA
CPU	Intel Core i7-1065G7 CPU 1.30GHz × 8
RAM memory	16.0 GiB DRAM
Disk memory	1.0 TB SSD
GPU	Mesa Intel Iris(R) Plus Graphics (ICL GT2)
Operative System	Ubuntu 22.04.4 LTS 64 bits
Dimensions	229 mm x 18 mm x 360 mm <sup>17</sup>

### 3.2.3. Software resources.

The next software tools will be employed, all of them at a free cost as they are open-source software. All of the licenses have been selected as open-source because they are tools to which the developer is used and that help making the budget cheaper.

Table 3.2.3.1

Software tool	License
Visual Studio Code text editor	Microsoft Software License
Ubuntu 22.04 operative system	GNU General Public License version 2 (GPLv2 for the Kernel)
GitHub online repository	As it is an online tool, the user is subscribed to GitHub's Terms of Service in-

	stead of a license.
Git tool for coding backups	GNU General Public License version 2 (GPLv2 for the Kernel)
Kivy 2.3	MIT License
Python 3.10	Python Software Foundation License (PSFL)

#### 3.2.4. Indirect costs and materials.

The average electricity costs in Spain from April to August of 2024 are<sup>18</sup>:

- April: 85.58 €/MWh
- May: 99.67 €/MWh
- June: 117.02 €/MWh
- July: 132.58 €/MWh
- August: 144.06 €/MWh

These prices result in an average cost of **115.72 €/MWh** per month. Let's assume **our laptop consumes 100 W per hour, our monitor consumes 30 W and our smartphone's consumption is too insignificant to be worth counting**. Keeping in mind that we will be using this tools for thirty hours a week, per four weeks a month has, that equals to 120 hours a month.

100 W of laptop's consumption · 120 hours of usage per month = **12000 W of laptop's consumption in a month** = 12 kW = 0.012 mW.

30 W of monitor's consumption · 120 hours of usage per month = **3600 W of monitor's consumption in a month** = 3.6 kW = 0.0036 mW.

Both tools have a consumption of 0.0156 mW per month. If electricity costs 115.72 €/Mwh, then the electricity cost per month equals to 1.81€ per month.

Also, we will consider that **our office consumes a total of 50 kW per month**, which adds  $0.05 \text{ mW} \cdot 115.72 \text{ €/Mwh} = 5.79 \text{ €}$  to the final invoice. In total, electricity costs rise up to 7.60 € per month, which results in **38€ across five months**.

For the Internet connection, our reference will be Digi's plan, which costs 15€ per month<sup>19</sup>, which results in **75€ across five months**.

Considering there might be also requirements to satisfy concerning office materials such as pens or notebooks, an extra of **30€ will be added to the budget**.

In total, this section has a cost of  $30 + 75 + 38 = 143$  €.

### 3.2.5. Total budget.

Here is a final table of the budget that this project might require.

Table 3.2.5.1

Resource	Cost
Human resources	9855 €
Software resources	0 €
Hardware resources	112.15 €
Indirect costs and materials	143 €
Total	10,110.15 €
Spanish “IVA” of 21%	2,123.13 €
<b>Final budget</b>	<b>12,233.28 €</b>

## 4. PROBLEM ANALYSIS

In this section, we will dive into the reasons that justify the existence of this work and explain step by step the specific problem that it must solve.

### 4.1. Biological context: the prediction of unknown RTs' behavior patterns experiment.

Mario Rodríguez Mestre's post degree work is not the only one that focuses on the RTs' experiment. Indeed, a similar article signed by Rodríguez Mestre himself along with Dr. Martínez-Abarca and others<sup>22</sup> was published back on December 2020.

The key of the study resides in functional association. As it has been mentioned in the [introduction](#), the research was aimed to find common behavior patterns between RTs that were separated in evolutive terms. In other words, the value of the study resides in discovering remarkable unknown correlations between RTs that despite of being far away from each other in terms of genetic evolution, present similar amino acid patterns and therefore common researching applications.

In nature, proteins are never found isolated. Instead, they are found as parts of a very large sequence of amino acids in which they are presented among many other proteins. One of the main researching tasks to accomplish with this matter is to recognize all the valid RTs contained in these chains, which usually implies to search for stop codons in the converted RNA/DNA sequence.

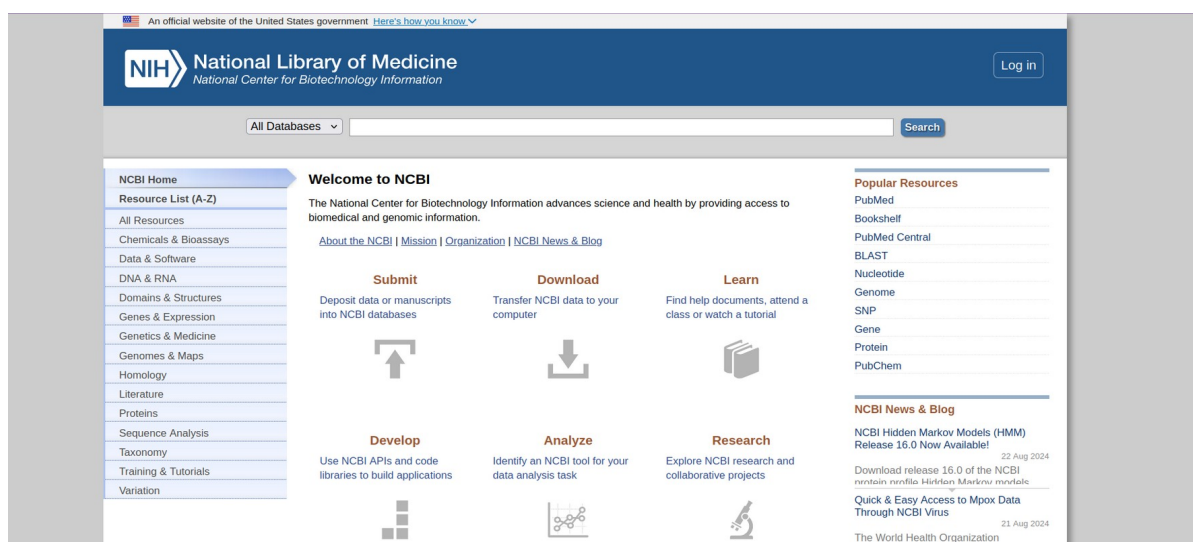
But, how can those long nucleotides sequences be found in the database? Researchers never have the complete string they are seeking. Instead, they search for a far more brief protein that they already know is contained in the sequence they are looking for (usually, each protein of a database is identified by an ID, so researchers use that ID in the search bar instead of the amino acid sequence that forms the protein. It is less prone to errors and more comfortable). This brief protein is known as a bait. Once the nucleotides sequences related to that bait are returned, researchers start looking for all the valid RTs that are contained in the nucleotides surrounding the bait.

In the study, researchers had an initial set of 198,760 annotated RTs, from which 9141 were finally selected as those that were representative of each evolutive branch. It is

important to remember that the main objective was to discover unknown patterns between non-related samples, because it was assumed that related samples would show a similar behavior, so their matches would have considered noise and consequently they needed to avoid. The filter applied consisted in comparing each protein with the rest of the dataset, and if an equivalent protein at 85% sequence identity was found, the selected protein was removed from the set. In the end, there were only samples corresponding to proteins having a minimum difference among each other of 25%. In the process of filtering the samples, a filogenetic tree was created.

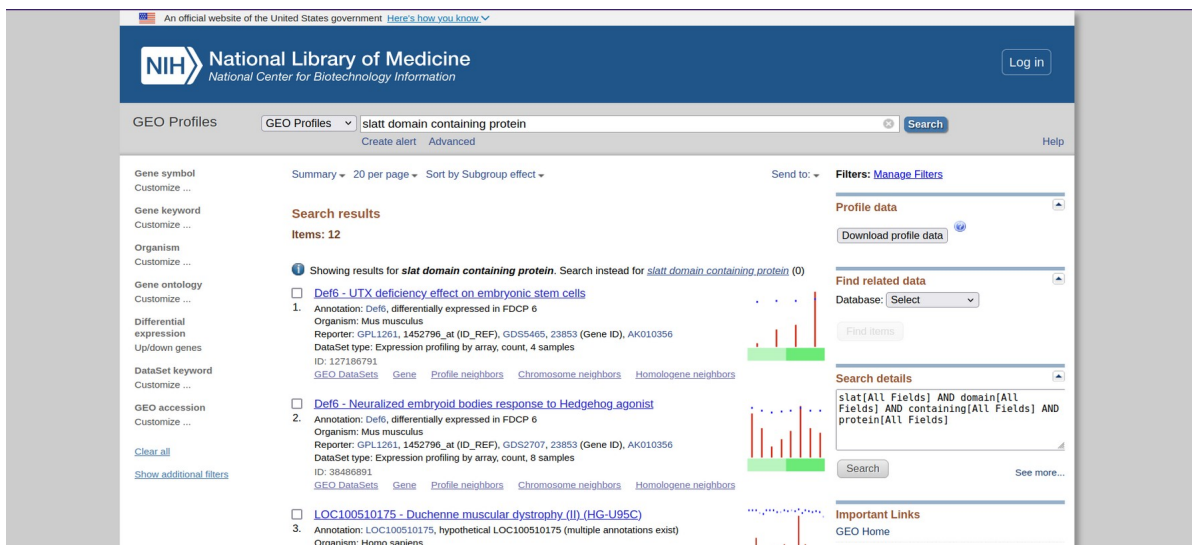
Each of the selected 9141 samples was employed as a bait for which were searched up to 30,000 nucleotides positioned before and after it (up to 60,000 nucleotides for each bait). Once all the nucleotides were isolated, the efforts were putted into finding all the proteins contained in each one of them. Finally, the clustering was applied to the final obtained set.

A comparison between two RTs can be done just by looking for common amino acid sequences spotted in both proteins. That is why a clustering algorithm is the perfect approach for this task, as a good implementation can help researchers discovering new patterns by just organizing the clustering results. As more than 60,000 clusters were returned by the clustering algorithm, in order to optimize the computational time required to process all of them, it was established a cutoff of the minimum samples that a cluster had to contain in order to consider that cluster to have a significant amount of proteins. After that, the number of clusters was reduced to 5413.



The final results were showed in a presence-absence matrix (PAM), which consists of a matrix where cells only get binary values. Usually, rows define a certain characteristic, columns another one, and each cell indicates if there is a value in the dataset that matches both characteristics at the same time. In this experiment, the PAM had the rows

corresponding to the RTs ordered by the position on the previously constructed filo-genetic



tree, and the columns corresponded to the various neighboring protein clusters, ordered by size (the first column represented the cluster that contained the major amount of samples, the second column, the second largest cluster, and so on). This matrix was then analyzed in order to get the results of the experiment.

## 4.2. The current problematic.

It has been explained how the experiment was made. But, how was the process of obtaining the samples? And how were they preprocessed before applying the clustering algorithm? In Rodríguez Mestre's post degree work<sup>12</sup>, the preprocessing of the samples implied using multiple tools, formats and manual operations that would be impossible to repeat in an affordable amount of time if the experiment wanted to be done again. It has been mentioned previously how Martínez-Abarca wished to analyze the new RT samples that were being included periodically in the databases, but that idea could not prosper as he had not the appropriate tool to automatize the preprocessing.

Long story short, there were no existing implementations of this issue, as it was a very specific and relatively new challenge. As a result, there were no similar tools that accomplish what the application had to do yet. In that way, GeneSys started from scratch, so at first the student's efforts were put into understanding what exactly the preprocessing to accomplish was. The tasks that compose the preprocessing are stated down below.

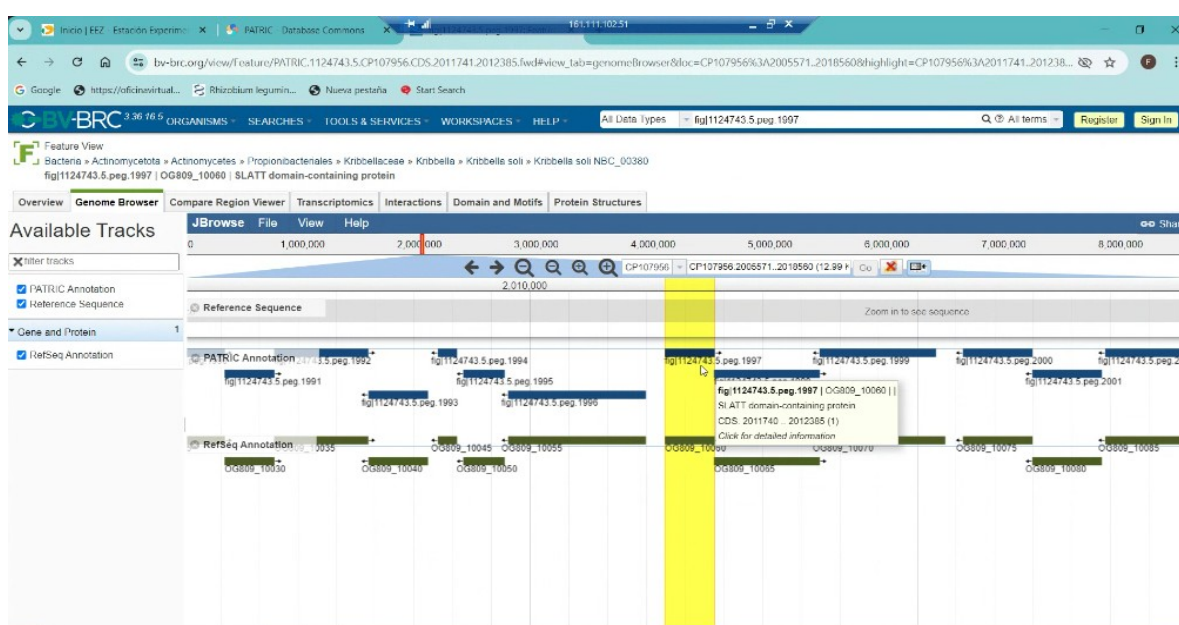
### 4.2.1. Tasks to accomplish.

- **Isolate useful information from RTs downloaded from databases.** Usually, genetic

databases incorporate a search bar and a friendly-using online interface to download the data recovered after searching for a specific term. It is possible to access to a given set of data and download it in some easily manipulable format.

Image 4.2.1.1 Interface of the National Center for Biotechnology Information's (NCBI) main page<sup>23</sup>.

Image 4.2.1.2. Results available for download in NCBI's website.

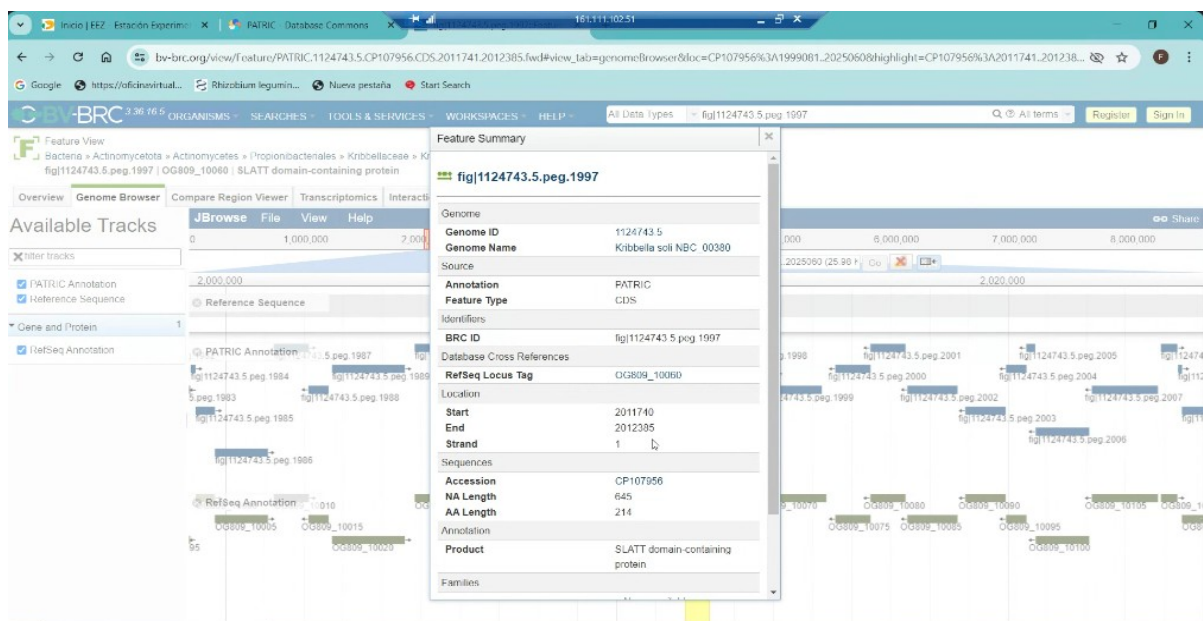


In many cases, the downloaded samples are not given as protein strings, but as its corresponding IDs, among other characteristics. In that case, it is mandatory to isolate proteins IDs' and remove the rest of the downloaded information, as we would need only the IDs' in order to get the amino acid strings corresponding to each ID.

- **Obtain the proteins from its corresponding IDs.** In case we had downloaded proteins' IDs, we would have to find a tool that returns each protein's full amino acid sequence given its ID, and integrate that tool into our application. In case a protein is repeated in the final dataset, it should be removed.

- **Isolate one sample from each evolutive branch.** Once all the proteins are stored without repetitions, it is time to apply the previously mentioned filtering to discard those samples that are closer to others in terms of evolution, so that we end up with a final dataset that stores one sample from each evolutive branch. The default way to decide whether a protein is in the same evolutive branch as another would be to compare each one with the rest and discarding those which are equivalent to at least another one by 85% of coincidence. However, we should let users to change that percentage in case they wanted.

Note that no filo-genetic tree is being created in this step as it is not a necessary action in



order to accomplish the task. However, Martínez-Abarca stated that it could be useful, but unfortunately there was no available time to include such functionality in the application.

- **Get 30,000 nucleotides to the right and to the left from each sample.** The next phase consists of employing each selected protein as a bait from which obtain up to 30,000 bases of nucleotides. The associated nucleotide sequence to which each bait is associated can be found in databases such as PATRIC<sup>21</sup> and can be manually downloaded from it. This phase must automatize the extraction of such large nucleotides sequences for each bait, so that researchers do not have to do it by themselves.

Image 4.2.1.3. Screenshot of PATRIC's online tool for manually downloading nucleotide bases of a certain sequence given its bait. The bait can be recognized in the image with an emphasized yellow color.

- **Recognize all the available RTs in the nucleotides sequence associated to each bait.**



Finally, before giving the preprocessed data to researchers so they can apply clustering, each obtained sequence of nucleotides must be read in order to get all the proteins it stores. Fortunately, it has been previously mentioned that the classification of potential proteins in a large sequence of nucleotides is already done by the databases that contain them, and when a large amount of RNA/DNA nucleotides is downloaded from them, some fragments of such nucleotides sequences that might correspond to proteins are already tagged. We would only have to observe those specific fragments and confirm if they have a valid stop codon or not.

The key of the process lies, indeed, in recognizing stop codons. The available combinations of bases that correspond to that case are: “UAA”, “UAG” and “UGA” for RNA sequences and “TAA”, “TAG” and “TGA” for DNA sequences. If a stop codon is not found in a candidate protein, it must be reversed and the process must be repeated. If still there is no valid stop codon at the end of the protein, the protein is discarded. A protein would also be discarded if it stores a stop codon in any part of the sequence but the end, and if it has not a length equivalent to a multiple of three, too (remember that nucleotides were read three by three in order to convert a string of bases into its corresponding string of amino acids).

Image 4.2.1.4. An example of how PATRIC shows information about a certain fragment of a nucleotides sequence that might correspond to a valid protein.

### **4.3. Selected languages and tools.**

Once all the required task to achieve are defined, it is time to select which tools to employ to face the development. The main issue is to select the database to work with. There are multiple available options for downloading genetic data: from the already mentioned PATRIC<sup>21</sup> and NCBI<sup>23</sup> to others such as PDB<sup>24</sup> (Protein Data Bank), CD Genomics<sup>25</sup>, the European Nucleotide Archive<sup>26</sup> (ENA) and Mgnify<sup>27</sup>.

When deciding which database should be employed, the major characteristic to look for is the disposition of a tool that allows to consult the database from a local script, so it can be called by the main GeneSys desktop application. It turns out that PATRIC has a command line set of tools<sup>28</sup> compatible with Linux Debian systems which can be employed to consult information from the database when needed. So, as PATRIC gives the option to design simple bash scripts that can be launched from our main application with the specific parameters to consult given by the user, that database is the chosen one for our work.

As the main programming languages, Python and Kivy were selected for the development. Kivy<sup>29</sup> is a very comfortable language that allows the creation of user interfaces in a very customizable and intuitive way. On the other hand, Python was chosen for two major reasons. The first one is the large amount of libraries that it provides to easily manipulate bioinformatic data at a high abstraction level without requiring to write complex code, specifically, the Biopython package includes a lot of classes aimed to manipulate DNA, RNA and amino acid strings. The second reason is that Python also provides a full package to develop Kivy applications directly from Python code, which will be so helpful in the developing process, making it easier to integrate the components of the user interface with the logic that is spotted below them. Using Kivy package for Python, Kivy interface objects can be created by just calling to Python constructors. The final design of the interface can be detailed with extra Kivy files, too.



## 5. ARCHITECTURE AND DESIGN

Now it is going to be exposed how the application is actually designed.

It would be inefficient to implement an app exclusively aimed to solve RTs' issue. Indeed, it would be much better to approach the problem from a more complete point of view. It is quite obvious that biology is a field in constant expansion that continually evolves and faces new challenges. We have stated previously that the volume of data related to bioinformatics is increasing through time. As a result, it is not strange to suggest the development of a tool that can be easily expanded and is aimed to incorporate new features. Even though at first the app will only solve RTs' problem, if it is designed in a very extensible way, it will be possible to employ its framework as a starting point for future bioinformatic data manipulation tools, so that in the end GeneSys can incorporate a lot of functionalities aimed to solve many bioinformatic challenges.

In order to accomplish that, it is crucial to design GeneSys attending to two major requirements.

- **Maintainability.** The software must be correctly designed so it is easy to update and modify through time, as well as having an easy-readable code and proper comments that explain how it works.
- **Extensibility.** It is fundamental to develop GeneSys as a software that allows the incorporation of future functionality aimed to solve new problems along with the RTs' experiment. All the incorporated new features must not change (or change as little as possible) the existing code and architecture of GeneSys.

Also, other requirements must be satisfied. Among them, all those related with making the application desirable for a generic user are crucial: usability, consistency and accessibility. Security, scalability, interoperability, reliability, availability, portability efficiency are important as well, and deserve to be mentioned.

To sum up, the project seeks to develop a flexible, scalable software framework catering to the needs of bioinformatics experts, ultimately enhancing their research endeavors for current and future challenges they might face.

### 5.1. Class diagrams.

Image 5.1.1. GeneSys' class diagram.

Apart from the initial GenesysApp class, which starts the Kivy application itself, there are two major sections in which the app is divided: Screens and Modules.

Screens contains all the Kivy interfaces defined in Python that show different windows, each one with a different function. Also, it is divided into two other sections. Firstly, GeneSys screen, corresponding to all the windows that are common to any module included in GeneSys aimed to solve any task.

- **MenuScreen** is the window that shows the user all the available issues for which a workflow can be defined. For now, it only has a button corresponding to the definition of a workflow to solve the RTs' issue. It has been designed this way so the architecture does not change in case new modules are added in the future.
- **SelectResultsPathnameWorkflowScreen**. Once a problem has been selected in the previous window, this one opens and asks the user to introduce where a txt file with the execution results of the workflow should be created. All the executed workflows will create a file in the specified path that provides important information about all the stages of the workflow execution.
- **WorkflowScreen** is the window where the user finds all the possible functions to apply to the workflow. Here, new tasks can be added to the workflow, all workflow's tasks or just the last one added can be removed, the workflow can be executed, the execution can be canceled and the workflow can be completely destroyed by returning to the main menu. Also, it allows the user to save the workflow in JSON format and to load a workflow from a previously defined JSON file.
- **GenerateWorkflowFromJsonScreen** is the window that opens when the option to load a workflow from a JSON file is selected in the WorkflowScreen window. It asks the user the pathname from which load the workflow.
- **GenerateJsonScreen** is the window that opens when the option to save a workflow into a JSON file is selected in the WorkflowScreen window. It asks the user the pathname to which save the workflow.

Secondly, Patric protein processing screen, which defines all the windows related to the tasks that must be accomplished by the GeneSys module that solves the RTs' issue. In the future, each new module that would be added to GeneSys will have a new section like this

one, including all the screens corresponding to the definition of the tasks that the module implements.

- **PatricTaskScreen** is the window that opens when the user selects the option to add a new task to a workflow aimed to solve the RTs' issue. It has five buttons, each one corresponding to a new window that allows the user to define the task to be added to the workflow.
- **IsolateCodesScreen** is the window that allows the definition of the task that isolates a specific column corresponding to the IDs of proteins downloaded from PATRIC.
- **FastaGenerationScreen** is the window that allows the definition of the task that obtains the amino acid sequences of a set of proteins given a csv file with an unique column that contains proteins' IDs.
- **ReduceSampleScreen** is the window that allows the definition of the task that reduces the sample of proteins so in the end the dataset contains only one sample of each evolutive branch.
- **Get30KBScreen** is the window that allows the definition of the task that obtains 30,000 nucleotide bases to right and to the left from a set of proteins that work as baits each.
- **RecognizeCodonsScreen** is the window that allows the definition of the task that searches for valid stop codons in a large set of nucleotides, recognizes the valid proteins associated with each bait and organizes the resulting matches in a final xlsx file that can be readable by researchers.

The Modules section includes the logic to solve the tasks that the user defines in the interface provided by the Screens section. For now, it includes two major parts.

- The **Task** class is the key of GeneSys' logic, from which all the classes in the Modules section inherit. It includes some abstract methods that are specifically implemented when the class is inherited.
- The **Workflow** class inherits from class, and defines how a workflow can be defined by the user and how it can be manipulated. A workflow is essentially defined as a list of tasks, which may also correspond to a workflow, which opens the window for implementing new modules in the future that might contain other workflows as tasks to be executed.

In essence, Task and Workflow are very flexible classes oriented to provide a first logic layer from which all the logic of any GeneSys module can be defined according to the challenge it tries to overcome.

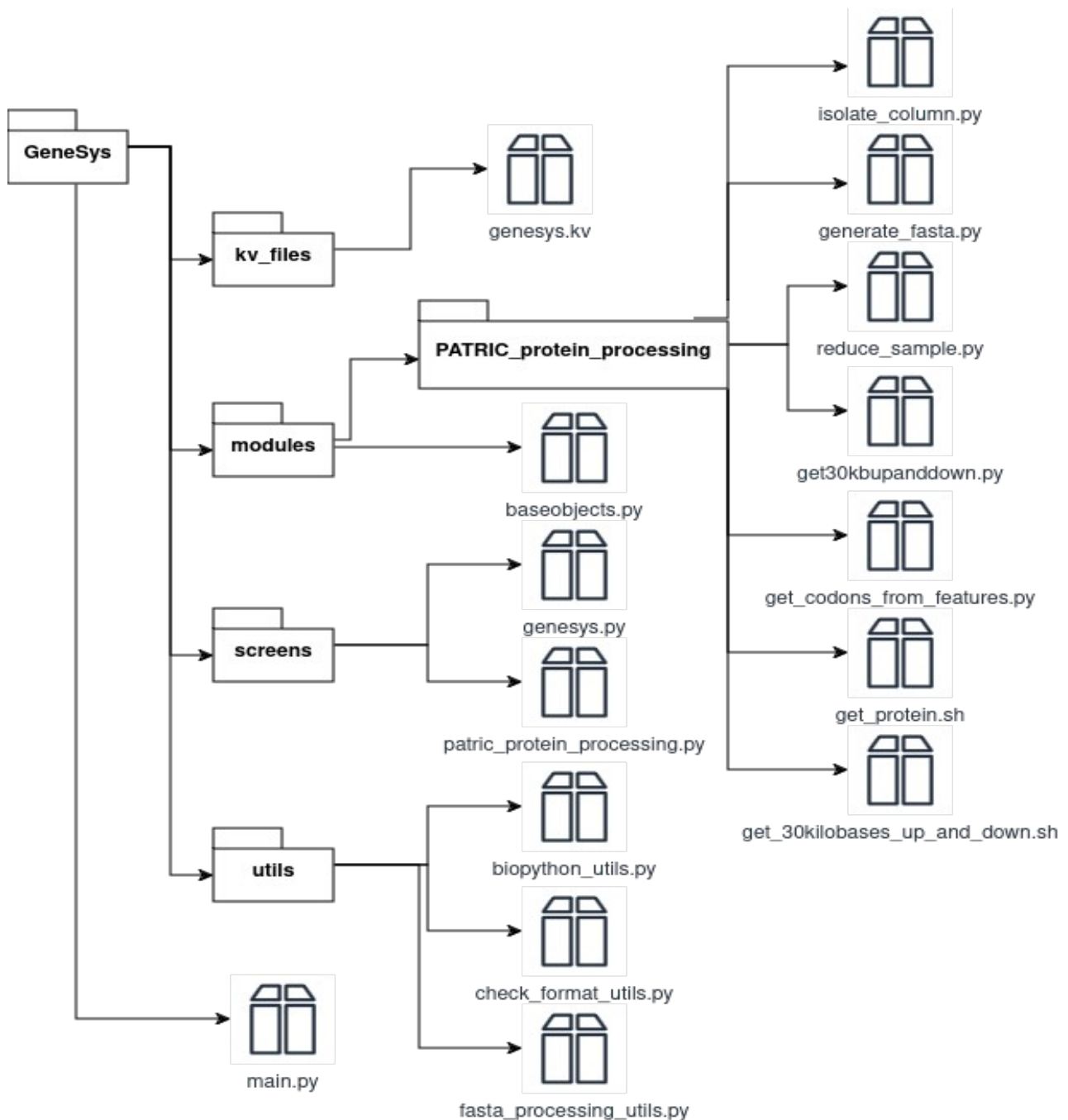
The second part of the Modules section corresponds to the specific tasks defined in order to solve the RTs' issue. Whenever a new module is implemented, a new section like this will be added to the application.

- **IsolateColumn** implements the logic given by the user in the IsolateCodesScreen associated window.
- **GenerateFasta** implements the logic given by the user in the FastaGenerationScreen associated window.
- **ReduceSample** implements the logic given by the user in the ReduceSampleScreen associated window.
- **Get30KbProteins** implements the logic given by the user in the Get30KBScreen associated window.
- **GetCodonsFromFeatures** implements the logic given by the user in the RecognizeCodonsScreen associated window.

The given permissions to all class' attributes and methods is always the most restrictive possible, for security reasons.

There are no sequence diagrams nor communication diagrams as the methods of all models tend to interact exclusively with their own classes or with just one object of another class. The cases where an object of a class calls to an object of another class are exposed verbally down below.

- The build method in GenesysApp calls to MenuScreen constructor.
- The open\_patric\_workflow\_menu method in MenuScreen calls to SelectResultsPathname-WorkflowScreen constructor.
- The create\_workflow and return\_to\_main\_menu methods in SelectResultsPathname-WorkflowScreen call respectively to WorkflowScreen and MenuScreen constructors.
- The open\_add\_tasks, save\_workflow, load\_workflow and return\_to\_main\_menu methods in WorkflowScreen call respectively to PatricTaskScreen, GenerateJsonScreen, GenerateWorkflowFromJsonScreen and MenuScreen constructors.
- The generate\_workflow and return\_to\_workflow\_screen methods in GenerateWorkflowFromJsonScreen call both to WorkflowScreen constructors.
- The generate\_json and return\_to\_workflow\_screen methods in GenerateJsonScreen call both to WorkflowScreen constructors.



- The `open_isolate_codes_menu`, `open_fasta_files_menu`, `open_reduce_sample_menu`, `open_get30KBupanddown_menu`, `open_get_codons_menu` and `open_workflow_menu` methods in `PatricTaskScreen` call respectively to `IsolateCodesScreen`, `FastaGenerationScreen`, `ReduceSampleScreen`, `Get30KBScreen`, `RecognizeCodonsScreen` and `MenuScreen` constructors.
- The `generate_task` and `return_to_task_screen` methods in `IsolateCodesScreen` call both to `PatricTaskScreen` constructor.
- The `generate_task` and `return_to_task_screen` methods in `FastaGenerationScreen` call both to `PatricTaskScreen` constructor.
- The `generate_task` and `return_to_task_screen` methods in `ReduceSampleScreen` call both to `PatricTaskScreen` constructor.
- The `generate_task` and `return_to_task_screen` methods in `Get30KBScreen` call both to



PatricTaskScreen constructor.

- The `generate_task` and `return_to_task_screen` methods in `RecognizeCodonsScreen` call both to `PatricTaskScreen` constructor.
- The `process_codes` method in `IsolateColumn` calls to `csv.DictReader` constructor.
- The `access_codes` method in `GenerateFasta` calls to `csv.DictReader` constructor.
- The `save_results` method in `GetCodonsFromFeatures` calls to `pd.DataFrame` constructor.

## 5.2. Folder organization.

Image 5.2.1. GeneSys' folder organization.

Concerning the folder disposition of the application, there is a main `.py` file at the root path that contains `GenesysApp` class. As a consequence, all the relative paths specified at any other file in the application will be given from the route at which the `main.py` file is located.

The `kv_files` folder contains a `.kv` file that specifies extra design specifications that affect to the user interface.

The `modules` folder includes all the implemented logic of the application. Inside, there are the `baseobjects.py` file, which contains the definition of the `Task` and `Workflow` base classes, and the folder associated with the tasks that are needed to solve RTs' issue. In the future, when a new module is added to GeneSys, all its tasks will be defined inside a new folder in this path.

The PATRIC\_protein\_processing folder contains a .py file for each task that must be solved in the RTs' issue. Also, it includes two basic bash scripts that are called while executing the tasks defined in generate\_fasta.py and get30kbupanddown.py files. get\_protein.sh receives a protein PATRIC ID as a parameter and calls to the specific command line BV BRC tool that returns the amino acid string corresponding to that protein ID, while get\_30kilobases\_up\_and\_down.sh receives a path to a fasta file and an undefined number of protein PATRIC IDs, calls to the command line tool that gets 30,000 nucleotides to the right and to the left of each of those IDs and stores the result in the specified fasta file.

The screens folder contains two Python files corresponding the first one to all the classes that define the screens common to all GeneSys modules (such as the workflow screen or the screen to save the workflow in a json file) and the second one to all the screens exclusive to the RTs issue (all the windows that allow users to manage the tasks associated with this matter).

Finally, the utils folder does not contain any Python classes. Instead, it stores some generic functions that might be employed by any task in any module. It includes three files, each one corresponding to functions oriented to solve similar problems. biopython\_utils.py has functions that employ classes from the Biopython package in order to cover specific necessities. check\_format\_utils.py is exclusively aimed to contain function that check the format of a path given by the user, such as csv, fasta, json... And fasta\_processing\_utils.py is oriented to provide generic functions that unify the way in which fasta files are read and written by all modules, apart from avoiding to rewrite the same code every time a task needs to fill or read a fasta file.

## **6. IMPLEMENTATION**

### **6.1. GeneSys' Kivy interface.**

How Kivy interfaces are managed in the application. Which widgets do you use. How is a Kivy application launched in Python. How do you launch yours through GenesysApp class. Include the specific .kv file that manages the final look of the interface.

#### **6.1.1. Screens related to all implemented modules.**

List the screens and explain how they work.

#### **6.1.2. Screens related to the module to process PATRIC proteins.**

List the screens and explain how they work.

### **6.2. Inner logic implemented in GeneSys.**

#### **6.2.1. Task and Workflow, the key classes that define GeneSys' logic.**

Explain how this classes work and what do they store.

#### **6.2.2. Specific tasks related to the module to process PATRIC proteins.**

Include the .sh scripts. Include context about how data are stored in PATRIC database.

### **6.3. Utils folder.**

Briefly remember why we use utils.

#### **6.3.1. Biopython related utils.**

#### **6.3.2. Format checking utils.**

### **6.3.3. Fasta processing utils.**

amino acid sequences that work as baits for RTs (in other words, a very large string of amino acid bases that potentially contains RTs within them and also stores a known protein that is employed to recognize the aforementioned long amino acid sequence as a whole)

write in the search bar of the database for a far more brief protein that they already know is contained in the sequence they are looking for (usually, each protein of a database is identified by an ID, so researchers put that ID in the search bar instead of the amino acid sequence that forms the protein. It is less prone to errors and more comfortable)

And finally, the Python library that supports bioinformatic functions, Biopython, was installed in the system.

## **7. GENESYS USER'S GUIDE**

## **8. CONCLUSIONS AND FUTURE WORK**

## 9. BIBLIOGRAPHY

1. [Reverse Transcriptases: From Discovery and Applications to Xenobiology](#)
2. [Factors that may predict next pandemic](#)
3. [Statistics Say Large Pandemics Are More Likely Than We Thought](#)
4. [Methods in molecular biology and genetics: looking to the future](#)
5. [DNA chemical compound](#)
6. [RNA chemical compound](#)
7. [What Is the Difference Between DNA and RNA?](#)
8. [Protein. Biochemistry](#)
9. [Translation of RNA to Protein](#)
10. [Reverse Transcription—A Brief Introduction](#)
11. [Data volume growth in genomics versus other disciplines](#)
12. Mestre MR (2020) “Analysis of novel and unexplored groups of prokaryotic Reverse Transcriptases”. Trabajo Fin de Master Universitario en Biotecnología (Universidad de Granada, curso 2019-20). University of Granada repository.
13. [Calculadora de IRPF Andalucía](#)
14. [What is the average lifespan of a laptop?](#)
15. [How Long Does a Computer Monitor Last: Lifespan Expectations Explained](#)
16. [How Long Can A Smartphone Last? \(With 6 Real Examples\)](#)
17. [Asus VivoBook S15 OLED K5504](#)
18. [Precio medio de la electricidad por meses: 2024](#)
19. [Qué fibra es mejor en España en relación calidad precio](#)
20. [GitHub repository: GeneSys.](#)
21. [BACTERIAL AND VIRAL BIOINFORMATICS RESOURCE CENTER, BV BRC.](#)
22. [Systematic prediction of genes functionally associated with bacterial retrons and classification of the encoded tripartite systems](#)

23. [National Center for Biotechnology Information main page.](#)
24. [Protein Data Bank.](#)
25. [CD Genomics.](#)
26. [European Nucleotide Archive, ENA.](#)
27. [Mgnify.](#)
28. [PATRIC command line set of tools.](#)
- 29.. [Kivy language documentation.](#)