
This is the **published version** of the bachelor thesis:

Morente Ribera, Adrian; Garcia-Font, Victor, dir. Development of an application for the generation of KPIs. 2023. (958 Enginyeria Informàtica)

This version is available at <https://ddd.uab.cat/record/272790>

under the terms of the  license

Development of an application for the generation of KPIs

Adrian Morente Ribera

Resumen– El éxito empresarial requiere de medidas cuidadosas, planificación y análisis. Los Indicadores Clave de Rendimiento (KPIs) son un conjunto de métricas que permiten evaluar el rendimiento de una empresa y proporcionar información acerca de su potencial de mejora. La monitorización de los KPIs permite a las empresas medir el éxito de sus operaciones y identificar las áreas de debilidad. En el presente documento se expone el desarrollo de una aplicación para una empresa del sector publicitario, que permitirá acelerar la generación de estadísticas por parte de la empresa. Esta aplicación se encargará de generar gráficos relacionados con 8 KPIs diferentes, los cuales podrán ser exportados aprovechando los datos de las campañas publicitarias existentes.

Palabras clave– KPIs, campañas y subcampañas publicitarias, operaciones CRUD, Escalabilidad, Testabilidad, SonarQube, Arquitecturas Limpias, DTOs...

Abstract– Business success necessitates careful assessment, organization, and investigation. KPIs (Key Performance Indicators) are a collection of measurements utilized to assess the performance of a business and furnish understanding into the capacity for progress. By tracking KPIs and surveying the relevant information, businesses can evaluate the accomplishment of their operations and recognize regions of deficiency. This paper presents the formation of an application for a firm in the advertising field, which will hasten the production of reports by the company. The application will produce diagrams related to 8 distinct KPIs, that can also be exported through the utilization of the data of the present advertisement campaigns.

Keywords– KPIs, advertising campaigns & sub-campaigns, CRUD operations, Scalability, Testability, SonarQube, Clean Architecture, DTOs...



1 INTRODUCTION

MEASUREMENTS are indispensable in ascertaining business performance. One of the best means of collecting data concerning the performance of a company is through the utilization of KPIs (Key Performance Indicators). The objective of KPIs [1] is to assist in the decision-making process by constructing a defined action trajectory for the company. These have seen widespread application in the market since they allow for the gathering of valuable information, measuring of variables and results, determination of effective strategies, making of decisions, and evaluation of the consequences of past strategies. The success of KPIs has been on the rise recently, primarily because of the recent advancements in Information Technologies, which employ simpler, more objective, and reliable data as opposed to traditional

communication channels.

The aim of this project, conducted during an internship at the firm App2U, was to add functionalities for the automated generation of KPIs to an existing ERP (Enterprise Resource Planning) software utilized by a company in the advertising sector. For the sake of confidentiality, this company shall be referred to as X for the remainder of the project.

2 INITIAL STATE OF THE PROJECT AND STATE OF THE ART

The existing ERP had all the instruments necessary to manage advertising campaigns. Prior to this application, however, the company employed Excel to accomplish commercial monitoring, collecting data concerning all the campaigns and customers. Employing Excel to manage advertising campaigns, however, was not the optimal solution for our client since the information had to be re-collected each month to generate a new version of the statistics. Furthermore, after finalizing a contract, it had to be shared with the marketing department to generate

- Contact email: adrian.morente.r@gmail.com
- Specialization: Information Technology
- Work mentored by: Víctor García Font (DEIC)
- Course 2022/23

the performance indicators, which could potentially lead to divergences.

To address this issue, the company desired to incorporate a new tool that could automatically generate KPIS as pie charts to track the performance of the enterprise. My responsibility in this project would be to integrate the generation of KPIS within the existing ERP, allowing for the visualization of the company statistics in an effortless manner, as well as creating a new feature for exporting this data to an Excel format. With this application, administrators will be able to generate data from the existing database without having to recalculate any supplementary information, thereby reducing the time spent by the company in calculating the performance statistics.

State of the art Key Performance Indicators (KPIS) are used to measure the performance of a business in areas such as cost, customer service, employee productivity, and overall efficiency. Companies often use KPIS to track progress and provide insight into how well their business is performing. They are also used to help identify areas where the company needs to improve and provide the necessary resources to do so. By using KPIS, companies can ensure they are running their business as efficiently as possible and make informed decisions that will help them achieve their goals.

3 OBJECTIVES

The main purpose of the project was to incorporate the KPI generation tool into the existing ERP system. The objectives were broken down into parts according to the client's demands and the current status of the product.

The MoSCoW prioritization method [10] was implemented to all the project's objectives, classifying the requirements into four distinct groups: Must-have (M), Should-have (S), Could-have (C), and Will-not-have (W).

- Must have (M): needs that are non-negotiable and necessary for the project's completion.
- Should have (S): encompasses important initiatives that are beneficial, but not essential
- Could have (C): requirements that are desirable but lack a significant impact if omitted
- Will not have (W): this category consists of tasks that are not priority and will only be completed if there is ample time

3.1 Export and visualize KPIS

For the generation of Key Performance Indicators (KPIS), the client requested that the existing entity (campaigns) be divided into two new entities, namely campaigns and sub-campaigns. Furthermore, new fields were required to be added to both entities. Following the division of campaigns into sub-campaigns, the statistics about the sub-campaigns could be extracted and subsequently exported into an Excel spreadsheet for the generation of various diagrams.

In this first table the objectives related to exporting and visualizing the KPIS are listed.

A. Database structure

Ref	Objective	Priority
A.1	Divide entity campaign into campaigns and sub-campaigns	S
A.2	Add new fields to campaigns	M

B. CRUD operations for sub-campaigns

Ref	Objective	Priority
B.1	Create new screen for sub-campaigns	S
B.2	Enable creation sub-campaigns	M
B.3	Enable remove sub-campaigns	M
B.4	Enable update sub-campaigns	M

C. Visualize KPIS

Ref	Objective	Priority
C.1	Create API for KPIS	C
C.2	Create screen for KPIS	M
C.3	Create drop-down for displaying the different KPI categories	M
C.4	Format pie charts	S
C.5	Export circular diagrams and sub-campaigns to Excel	M
C.6	Generate KPIS (Sectors, Supports, Cities, percentage closed contracts, contract typology, season, duration, no engagement reason)	M

3.2 Scalability and testability

Apart from the requirements imposed by the client, the company had internal objectives to be achieved in order to make the code testable and scalable.

Scalability [2] prepares software for future growth while creating a leaner product that is tailored to current needs without excessive complexity. A system is deemed scalable when it does not necessitate a redesign to maintain effective performance after an increase in workload.

Furthermore, a significant objective of the development was to test all potential scenarios to guarantee that the application operates as anticipated. Having automated tests that cover all conceivable cases is essential for identifying and resolving bugs or unexpected behaviors prior to releasing the ultimate version of the solution.

These objectives have been evaluated using two distinct metrics.

- Scalability has been measured using SonarQube [3], an automatic code review tool that assists in the delivery of clean code. It boasts numerous functionalities; however, I focused on the release of quality code (D.1) and technical debt (D.2). The former metric assesses the existence of potential bugs, which may lead to unexpected behaviors that influence the end user, whereas the latter ensures the codebase is both clean and maintainable for subsequent iterations.

- Testability has been measured by assessing the coverage of our tests. Code coverage [4] is a percentage measure of the degree to which the source code of a program is executed when a particular test suite is run. For our company standards the domain coverage is required to be 100%, whereas the coverage of the whole product (domain, application, and infrastructure) is required to be higher than 80% (D.3 & D.4) (The concepts of software layers, domain, application, and infrastructure will be introduced later in section 4.2)

Even though this was not an initial requirement of the customer, both scalability and testability are essential for granting the quality of the code. Thus, they have been included as objective for the project and they were analyzed at the final stage of it.

D. Test and release

Ref	Objective	Priority
D.1	(SonarQube) ¹ Minimize quantity of bugs and code smells ²	C
D.2	(SonarQube) Reduce technical debt ratio ³	C
D.3	Domain coverage	S
D.4	Application coverage	S
D.5	Responsible design for screen widths between 769 and 1800 pixels	M

4 METHODOLOGY

4.1 Planning methodology

I have been collaborating with a small team of three developers, one of which was the project manager, following an Agile methodology [7]. We divided our meetings into daily and weekly sessions, along with sprint planning and sprint review meetings. The daily meetings served to ensure everyone was aware of their tasks and any doubts or issues could be reported to the project manager; these generally took approximately 15 minutes. At the end of the week, an hour-long meeting was held to discuss project-related matters. The project manager additionally reviewed the code and functionalities of the development. Redmine, a project management application with time tracking, role-based access system, Gantt charts, and other functionalities, was the tool utilized for organization. The

1. These objectives will be measured by the SonarQube application

2. Code smells [5] are structures in the code that indicate the violation of fundamental design principles, they are not bugs since they are not technically incorrect and do not prevent the program from functioning; nevertheless, they indicate weaknesses in design that may slow down future development or increase the risk of bugs or failures.

3. Technical debt [6] refers to the amount of time that we will need in the near future to refactor the code in order to correct wrong patterns in our code. For instance when we duplicate code instead of creating a class or duplicate literals instead of introducing constants. In this case, the code is submitted earlier but then in the future, we will have to return that time to avoid errors.

product was delivered to the client after each sprint for testing the implemented functionalities.

4.2 Software architecture methodology.

This project has been developed using a Clean Architecture structure [8] [9], which has recently become an area of increasing attention. Ideas such as Hexagonal Architecture or Onion architecture have risen in popularity, with each demonstrating different implementations yet sharing the same objective of concern separation by dividing the software into layers. The main characteristics of these systems are:

1. Independent of frameworks. The architecture is not tied to a specific framework, instead, the framework is just used as an additional tool for our system.
2. Testable. Business rules can be tested without needing UI, database, or external elements.
3. Independent of the UI. Changing the UI does not imply changing anything about our business rules
4. Independent of the database. Business rules are not bound to the database.
5. Independent of external agents.

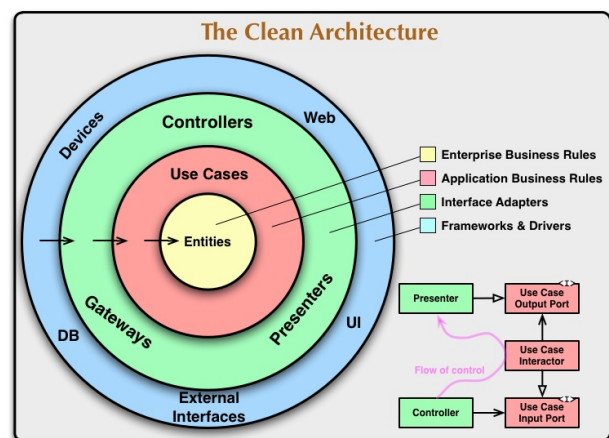


Fig. 1: Representation of the distribution of concerns in hexagonal architecture

Figure 1, illustrates the distribution of concerns within hexagonal architecture, wherein the concentric circles are representative of different areas of the system, with deeper circles corresponding to higher-level software. The Dependency Rule is the main principle of clean architectures, which dictates that source code dependencies must only point inwards, nothing in an inner circle can know anything about an outer circle, and data formats used in external layers cannot pass to the internal ones. Four circles can be distinguished:

1. Entities. They encapsulate business rules. No change in navigation, security, or database should affect this layer.

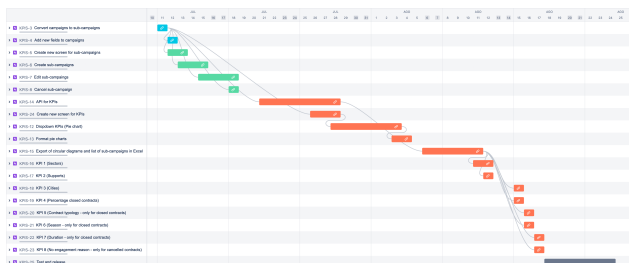
- Functional requirements:

Priority	Functional Requirement
M	Database modification.
S	Add topology and sector to new campaign class.
M	Add navigation KPIs button.
M	Create screen for sub-campaigns and KPIs.
M	CRUD operations for sub-campaigns.
M	Statistics for the 8 KPIs.
S	API for retrieving the KPI data.
M	Excel generation.
S	API for exporting the Excel.

Non-functional requirements:

Priority	Functional Requirement
C	Color palette for KPI charts.
S	Format KPI charts with legends and percentages.
M	Format Excel in a structure imposed by the client.
W	Correct previous bugs of the ERP.
S	Domain coverage (100%).
S	Application coverage. At least (80%).
M	Responsible design for computers
M	Keep with the style of the existing project.
M	Use Django 2.2 and Python 3.7 for the back-end.
S	Use jQuery for the front-end.
S	Pass SonarQube.

1. Database structure (Blue)
2. CRUD operations for sub-campaigns (Green)
3. KPI visualization (Orange)
4. Test and release (Grey)



6.2 System use cases

The use cases included in the application can be divided into two categories, those that are available to all users and those that are only available to administrators. Updating or deleting an existing sub-campaign and generating new sub-campaigns are restricted to administrators, whereas all system users can access the sub-campaigns, export KPIs and visualize KPIs.

6 SYSTEM DESIGN

6.3 System classes

The most important features of this project are the exporting the excel containing all the statistics and the generation of KPIs within the existing ERP.

Thus, the class diagram will be focused on showing the main classes that had to be used to reach this purpose. The classes have been divided into three different categories: persistent classes, DTOs for generating the KPIs, and DTOs for exporting the Excel.

As it can be seen in figure 4 the persisting classes are only composed of the campaign and sub-campaigns. The

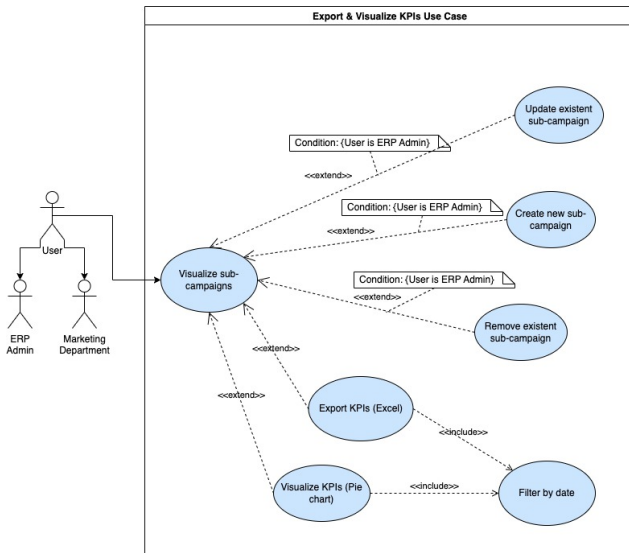


Fig. 3: System use cases

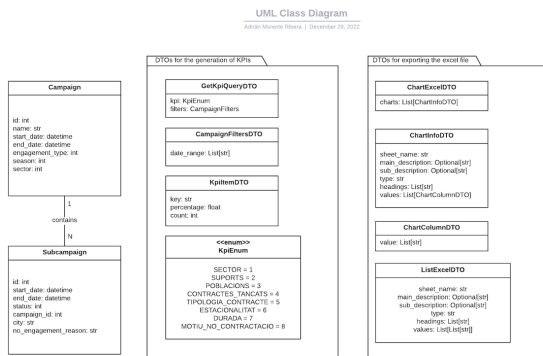


Fig. 4: Class diagram of the system. From left to right: persistent classes, DTOs for generating the KPIS and DTOs for exporting the Excel.

campaign class can have many sub-campaigns related to it. It can be noticed that no additional information is persisted in the database for the generation of KPIS or Excel. This happens because each KPI needs different data to be built which can be retrieved from the campaign and sub-campaign entities.

Thus, the use cases for visualizing KPIS and exporting KPIS will only use the information available in these classes. As it has been mentioned in section 4.2, the information that crosses the infrastructure, application, and domain layers will travel within DTOs (Data Transfer Objects) [11], which are immutable classes that encapsulate the data to be transferred.

6.3.1 Visualize KPIS DTOs

When the user requests a specific KPI, the *GetKpiQueryDTO* will be generated in the view and will travel to the module responsible for retrieving the sub-campaigns. Once the sub-campaigns are retrieved the module responsible for building the KPI will create a list of *KpiItemsDTO* that contain the information about every KPI that will later be used to create a pie chart in the front end.

6.3.2 Export KPIS DTOs

The case of exporting the excel is similar to the previous one; however, apart from the KPIS, the excel must include additional information about the sub-campaigns. Thus, the DTOs must contain significantly different data. The *ListExcelDTO* will contain the information needed to build the first page of the excel. Apart from containing the values, this DTO contains the headings for the data, the main description, and the sub-description for the page as well as the sheet name.

The remaining pages, which contain one KPI in each page will be built using the *ChartInfoDTO*, which contains the same values that the *ListExcelDTO*. However, they have been separated into 2 different DTOs in case the implementation changes in the future.

For more information about the specific format of the Excel please check the complete demo from the link present in section 6.4 or check appendix A.2

6.4 UI & UX Design

For the design of the KPIS page, a navigation button was added to the menu of the existing ERP to allow access to the page. The two new views to be added to the ERP were designed using Figma [12], a design tool frequently used by companies for designing graphical interfaces for applications.

The design of the landing page can be seen in Figure 5.

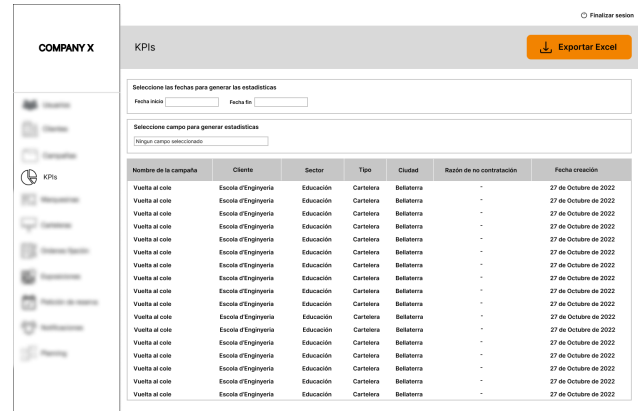


Fig. 5: Landing KPIS page contained within existing ERP

In this design, it is evident that the website has a left navigation bar, providing access to the new feature of the application. Upon navigating to the KPIS page, the list of existing sub-campaigns contained in the specified date range can be seen (if no date range is specified, all campaigns will be visible in a paginated view).

The sub-campaigns will be presented in a list comprising of their details, including name, client, sector, type, city, no engagement reason (if relevant), and creation date.

The page will also feature a filter for dates and a selector for the different types of pie charts which can be generated from the data. Upon selecting an option to display a pie chart, the design can be seen in Figure 6

The demo for checking the appearance of the Excel exported file can be found in Appendix A.2. Data for the demo has been generated exclusively for this purpose, as

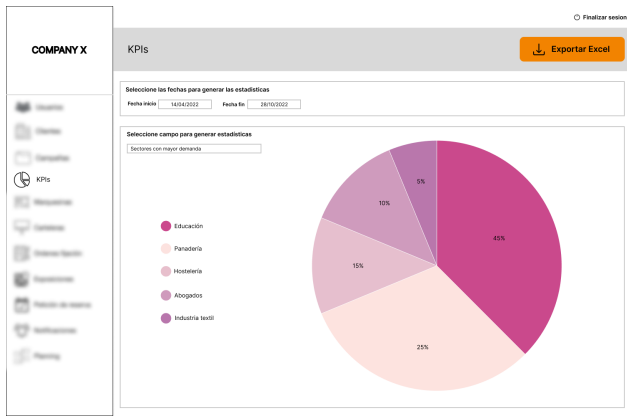


Fig. 6: Selector and pie charts containing the KPIs

the original data is confidential and cannot be disclosed. For more information, please refer to the following link: **DEMO KPIs**.

7 IMPLEMENTATION

7.1 Technologies

The project's back-end was developed in Django, a back-end server-side web framework written in Python which is scalable, secure and maintainable. Django was chosen due to the fact that the previous stages of the project had been implemented with it.

Although the back-end was the primary focus, it was necessary to make use of a front-end library to create the KPI diagrams. ChartJS was eventually chosen, primarily for its minimalism and its comprehensive documentation. SCSS was employed for the styling of the front end. This is an extension of CSS, allowing for the use of variables, nested syntax and a more organised structure.

Lastly, Apache Subversion (SVN) was used as the project's Version Control System as it is the tool currently in use by the company.

7.2 Main features

In this section the two main process interactions of my project will be shown by using a sequence diagram.

7.2.1 Export Excel

The excel containing all the statistics of the company from a given date range can be downloaded from the KPIs page.

As it can be seen in figure 7, the user interacts with the view by selecting the date range. Then, the list of sub-campaigns used to generate the statistics will be within the range selected by the user. This date filter travels to the *GetExcelHandler* class, which is responsible for calling the *SubcampaignFinder*. The latter one is responsible for getting the sub-campaigns from the database that match the specified criteria. Once the list of sub-campaigns has been retrieved they travel back to the handler.

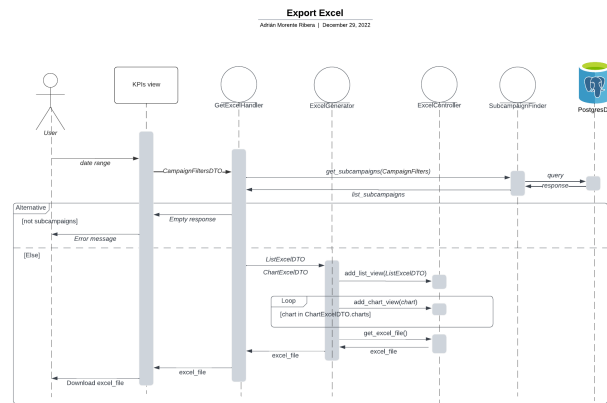


Fig. 7: Sequence diagram containing the logic behind the Export Excel Use Case

After that, there are two possibilities. In the case, there is not any sub-campaign matching the criteria the handler will send an empty response to the view, which will be responsible for showing an error message to the end user. If some sub-campaigns have been retrieved, the *GetExcelHandler* will create the *ListExcelDTO* and *ChartExcelDTO* that will go to the *ExcelGenerator*. The *ExcelGenerator* is responsible for creating the structure of excel and creating its pages; however, it delegates the responsibility for creating the graphics on each page. This last part is done by the *ExcelController*, who is responsible for filling the excel with the appropriate information and then sending back the file.

7.2.2 Generate KPIs

Apart from downloading the complete set of statistics via excel, the users can also visualize the KPIs in the same website by the generation of pie charts that include the statistics of the category selected by the user.

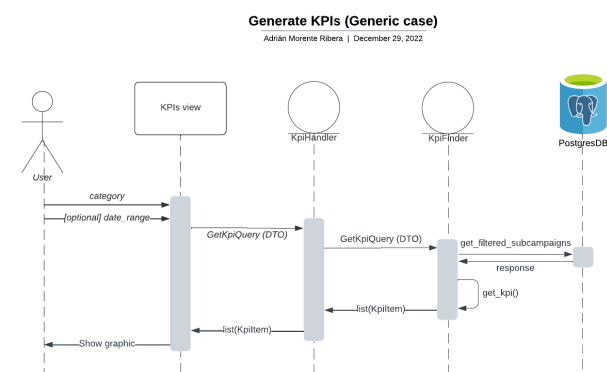


Fig. 8: Sequence diagram containing the logic behind the Generate KPIs Use Case

In the diagram present in figure 8, it can be seen that the user selects the category and date ranges that are gathered by the view. The view sends the information about the category within the *GetKpiQuery* that will travel to the *KPIHandler*. The only function of the *KPIHandler* is to call the finder and check if there are any errors.

The *KPIHandler* is then responsible for getting the filtered list of sub-campaigns in case the user has specified a date range. This is done to reduce the computing time for following queries, by collecting the statistics only from a small set of sub-campaigns instead of getting it from all of them. Then, once the sub-campaigns that contain the information that is needed for generating the KPI have been collected, the *KpiFinder* will create the list of *Kpiltem* that will be used by the view to build the pie charts.

8 RESULTS

During this section we will analyze the degree of completeness of the original objectives of the project and we will evaluate the final result of the project using some metrics.

8.1 Analysis of initial requirements

The functional requirements were successfully accomplished; however, the final stage of the project, centred around release and testing, was delayed. This was due to the fact that after submission of the results to the client, modifications were requested. These modifications, along with some additional testing, caused an 1-2 week delay in the project.

Due to this lack of time, some non-functional requirements had to be postponed for subsequent releases. The list below comprises of all non-accomplished requirements not related to test coverage nor to vulnerability analysis.

- Color palette KPI charts: a basic color palette of twenty colors has been created to manage most of the KPIs. However, should the number of categories grow, new colors will be randomly generated, which could potentially lead to accessibility issues if two adjacent colors are too similar.
- Correct previous bugs of the ERP (NF4): due to the limited time remaining in the project, only minimal effort was allocated to rectifying any existing errors.

8.2 Evaluation. Test coverage and vulnerability analysis

At this stage, the outcome of the project will be evaluated using the test coverage and the vulnerability analysis generated by SonarQube, an application that reviews the code in order to prevent bugs and other problems.

The test coverage, as previously mentioned, measures the proportion of code that is covered by tests, guaranteeing that the code uploaded to production works as intended. The test coverage will be examined firstly for the domain layer and then for the entirety of the application.

- Domain coverage: the requirement for all new projects in the company is to have 100% of domain coverage we cannot say that it was accomplished; however, this 5% remaining coverage will be added in future iterations.

Coverage report: 95%

Module	statements	missing	excluded	coverage %
/Users/adrian/Documents/Projects/rtm/adrian/gpu/ib/python3.7/site-packages/domain_utils/_init_.py	2	0	0	100%
/Users/adrian/Documents/Projects/rtm/adrian/gpu/ib/python3.7/site-packages/domain_utils/architecture/_init_.py	0	0	0	100%
/Users/adrian/Documents/Projects/rtm/adrian/gpu/ib/python3.7/site-packages/domain_utils/architecture/entity.py	3	0	0	100%
/Users/adrian/Documents/Projects/rtm/adrian/gpu/ib/python3.7/site-packages/domain_utils/architecture/value/_init_.py	0	0	0	100%
/Users/adrian/Documents/Projects/rtm/adrian/gpu/ib/python3.7/site-packages/domain_utils/architecture/value_objects/paid.py	8	0	0	100%
/Users/adrian/Documents/Projects/rtm/adrian/gpu/ib/python3.7/site-packages/domain_utils/architecture/value_objects/identifier.py	3	0	0	100%
server/backend/campaign/application/_init_.py	0	0	0	100%
server/backend/campaign/application/update_campaign_data.py	21	0	0	100%
server/backend/campaign/domain/_init_.py	0	0	0	100%
server/backend/city/application/_init_.py	0	0	0	100%
server/backend/city/application/add_city.py	17	0	0	100%
server/backend/city/application/delete_city.py	16	0	0	100%
server/backend/city/application/update_city.py	24	0	0	100%
server/backend/city/domain/_init_.py	0	0	0	100%
server/backend/city/domain/entities.py	19	0	0	100%
server/backend/city/domain/value_objects.py	7	0	0	100%
server/backend/company/application/_init_.py	0	0	0	100%
server/backend/company/domain/value_objects.py	3	0	0	100%
server/backend/engagement_type/application/_init_.py	0	0	0	100%
server/backend/engagement_type/application/add_engagement_type.py	17	0	0	100%
server/backend/engagement_type/application/delete_engagement_type.py	16	0	0	100%
server/backend/engagement_type/application/update_engagement_type.py	24	0	0	100%
server/backend/engagement_type/domain/_init_.py	0	0	0	100%
server/backend/engagement_type/domain/entities.py	19	0	0	100%
server/backend/engagement_type/domain/value_objects.py	7	0	0	100%
server/backend/exposition/application/_init_.py	0	0	0	100%
server/backend/exposition/application/show.py	10	0	0	100%
server/backend/exposition/application/new_subcampaign_exposition.py	33	0	0	100%

Fig. 9: Domain coverage report

- Application coverage: as it can be seen, the result is far away from the requirement, which was having at least 80% of application coverage.

This was mainly due to the fact that there is a lot of legacy code, developed 1-2 years ago that did not follow the architectural patterns present in the company nowadays, introducing intricate code that is difficult to test.

Coverage report: 55%

Module	statements	missing	excluded	coverage %
server/gpu/controllers/filters_controller.py	64	55	0	14%
server/gpu/controllers/timeline_controller.py	125	105	0	16%
server/gpu/controllers/exposition_controller.py	15	10	0	33%
server/gpu/controllers/order_controller.py	29	19	0	34%
server/gpu/controllers/bus_rent_controller.py	44	28	0	36%
server/gpu/controllers/rent_request_controller.py	13	8	0	38%
server/gpu/controllers/alert_controller.py	18	11	0	39%
server/gpu/controllers/fixation_controller.py	18	10	0	44%
server/gpu/managers/bus_rent.py	32	18	0	44%
server/gpu/models/bus_rent.py	75	42	0	44%
server/gpu/controllers/campaign_controller.py	31	17	0	45%
server/gpu/managers/support_rent.py	20	11	0	45%
server/gpu/models/rent_request.py	69	38	0	45%
server/gpu/controllers/user_controller.py	61	33	0	46%
server/gpu/controllers/rent_controller.py	15	8	0	47%
server/gpu/controllers/subcampaign_controller.py	15	8	0	47%
server/gpu/models/fixated_item_fixation.py	43	23	0	47%
server/gpu/controllers/bus_controller.py	10	5	0	50%
server/gpu/controllers/company_controller.py	14	7	0	50%
server/gpu/controllers/fixated_item_controller.py	10	5	0	50%
server/gpu/managers/billboard_rent.py	8	4	0	50%
server/gpu/managers/marquee_rent.py	8	4	0	50%

Fig. 10: Application coverage report

When analyzing the metrics generated by SonarQube, it can be seen in figure 11, that some of the conditions were accomplished, but most of them were not fulfilled. The main reason for this is the legacy code previously mentioned.

76	Bugs	Reliability	E
0	Vulnerabilities	Security	A
37	Security Hotspots	0.0% Reviewed	Security Review E
5d 6h	Debt	173 Code Smells	Maintainability A

Fig. 11: General status of the project

This can be easily seen if we analyze figure 12 and 13.

These figures compare the number of issues before and after uploading the final code to the main branch of the version control system.

Before September 2022:

- 77 bugs
- 166 code smells
- 0 vulnerabilities

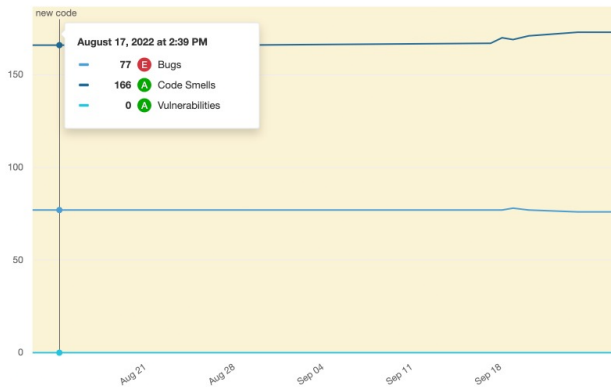


Fig. 12: Initial status of the project

After September 2022:

- 76 bugs
- 173 code smells
- 0 vulnerabilities

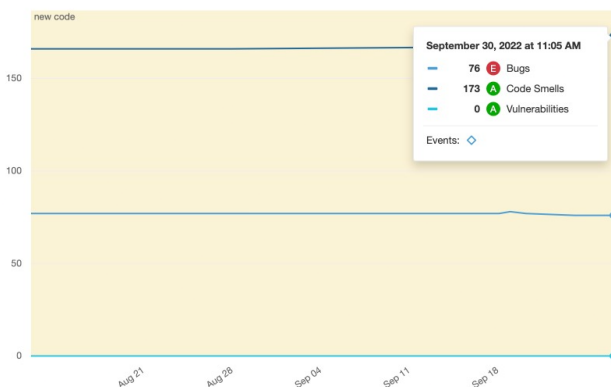


Fig. 13: Final status of the project

Thus, the number of bugs reduced in 1 and only 7 code smells were introduced during the development of the application for the generation of KPIS.

9 CONCLUSION

As assessed in the preceding section, the vast majority of the objectives have been achieved. The key features, such as the CRUD operations for sub-campaigns, the generation of KPIS and the ability to export all the data in Excel, have been successfully implemented at the end of the project.

As a comprehensive summary, the project has been a success as all the requirements requested by the client have been met and the application is currently running in the production environment with no reported errors. The development of KPIS and the exporting application has drastically reduced the time needed by the advertising company to monitor their statistics.

From a personal point of view, the development of the application for the generation of KPIS was a challenging but rewarding experience. Being a first-time developer using Django, I had to overcome the learning curve and become acquainted with the framework's syntax and features. Moreover, tackling the legacy code of the company's existing system was a challenge, as it necessitated me to meticulously study and comprehend the existing codebase in order to integrate my new application seamlessly.

Overall, the project was a success and the new application has proven to be a valuable asset for the company, providing real-time insights and metrics to aid them in making informed decisions. The project has also provided me with valuable experience in working with Django and dealing with legacy code, which will be advantageous skills in future development projects.

REFERENCES

- [1] What is a Key Performance Indicator? [Online]. Available: <https://www.kpi.org/KPI-Basics/>
- [2] Kevin Hobert. (2018, Jul 31). Writing Clean Code — A Paradigm for Scalable Codebase [Online]. Available: <https://medium.com/@kevinhobert29/writing-clean-proper-code-8f7dd80a0626>
- [3] SonarQube [Online]. Available: <https://www.sonarqube.org/>
- [4] Wikipedia. "Code coverage" [Online]. Available: https://en.wikipedia.org/wiki/Code_coverage
- [5] Wikipedia. "Code smell" [Online]. Available: https://en.wikipedia.org/wiki/Code_smell
- [6] (2021, Apr 15). How To Evaluate The Technical Debt With Sonarqube [Online]. Available: <https://www.bitegarden.com/how-to-evaluate-technical-debt-sonarqube>
- [7] Atlassian. What is Agile? [Online]. Available: <https://www.atlassian.com/agile>
- [8] Javier Ferrer. (2016, May 12). Introducción Arquitectura Hexagonal – DDD [Online]. Available: <https://codely.com/blog/screenshots/arquitectura-hexagonal-ddd>
- [9] Robert C. Martin. (2012, Aug 13). The Clean Architecture [Online]. Available: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>
- [10] ProductPlan. MoSCoW Prioritization [Online]. Available: <https://www.productplan.com/glossary/moscow-prioritization/>

- [11] Martin Fowler. Data Transfer Object [Online]. Available:
<https://martinfowler.com/eaCatalog/dataTransferObject.html>
- [12] Figma [Online]. Available:
<https://www.figma.com/design/>

APPENDIX

A.1 Gantt diagram

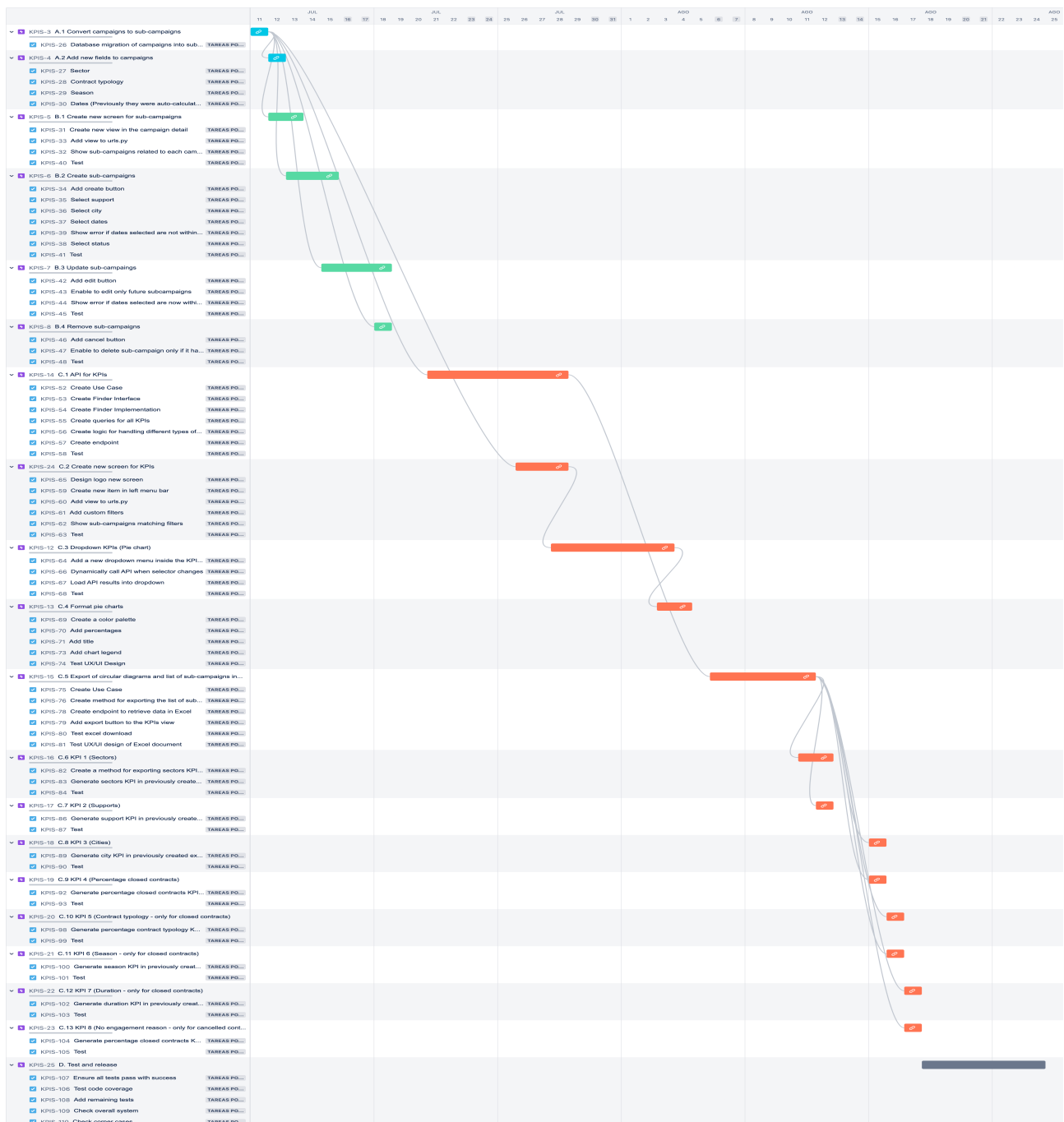


Fig. 14: Complete Gantt diagram followed during the execution of the project

A.2 Demo Excel

ESTUDIANT	GRAU	FACULTAT	ANY DE NAIXEMENT	POBLACIÓ	BECARI
Adalberto Sola Gallart	SINYERIA DE TELECOMUNICACIÓ	ESCOLA D'ENGINYERIA	2000	VALENCIA	SI
Primitiva Pinto	FILOLOGIA ANGLESA	ESCOLA DE HUMANITATS	2002	BARCELONA	NO
Óscar Sabas Rueda Robles	QUIMICA	ESCOLA DE CIENCIES	2001	SABADELL	NO
Sergio Pardo Benavides	ADE	ESCOLA D'ECONOMIA	1999	SABADELL	NO
Rosario Adolfo Arranz Cuesta	HISTORIA	ESCOLA DE CIENCIES SOCIALS	1998	TERRASA	NO
Luis Amando Zamora Zapata	ARTS ESCENIQUES	ESCOLA D'ARTS	1980	TERRASA	SI
Àurea Alegria Herrera	HISTORIA	ESCOLA DE CIENCIES SOCIALS	1999	EL PRAT	NO
Ruben Ochoa Diez	GEOGRAFIA	ESCOLA DE CIENCIES SOCIALS	2000	BADALONA	SI
Cirino Gutiérrez Tamarit	FISICA	ESCOLA DE CIENCIES	2000	SANT CUGAT	SI
Edmundo Noé Zabaleta Seguí	MATEMATIQUES	ESCOLA DE CIENCIES	2002	MANRESA	SI
Primitiva Prudencia Pardo Vives	ENGINYERIA INFORMATICA	ESCOLA D'ENGINYERIA	2000	BILBAO	SI
Paulino Santamaría Gual	ECONOMIA	ESCOLA DE CIENCIES SOCIALS	2002	BARCELONA	NO
Tania Alejandra España Bernat	ENGINYERIA QUIMICA	ESCOLA D'ENGINYERIA	2001	SABADELL	NO
Amador Horacio Ferrando Escolano	ENGINYERIA INFORMATICA	ESCOLA D'ENGINYERIA	1999	SABADELL	NO
Sigfrido Salinas Campos	HISTORIA	ESCOLA DE CIENCIES SOCIALS	1998	TERRASA	NO
Tatiana Alcántara Goicoechea	ENGINYERIA DE DADES	ESCOLA D'ENGINYERIA	1980	TERRASA	SI
Fabián Río Cuevas	MATEMATIQUES	ESCOLA DE CIENCIES	1999	EL PRAT	NO
Julia Benítez-Suárez	SINYERIA DE TELECOMUNICACIÓ	ESCOLA D'ENGINYERIA	2000	BADALONA	SI
Emilio Torrens Cordero	ECONOMIA	ESCOLA DE CIENCIES SOCIALS	2000	SANT CUGAT	SI
Teodoro Paniagua Bosch	ARTS ESCENIQUES	ESCOLA D'ARTS	2002	MANRESA	SI

Fig. 15: Expected view of the list of sub-campaigns within the exported excel

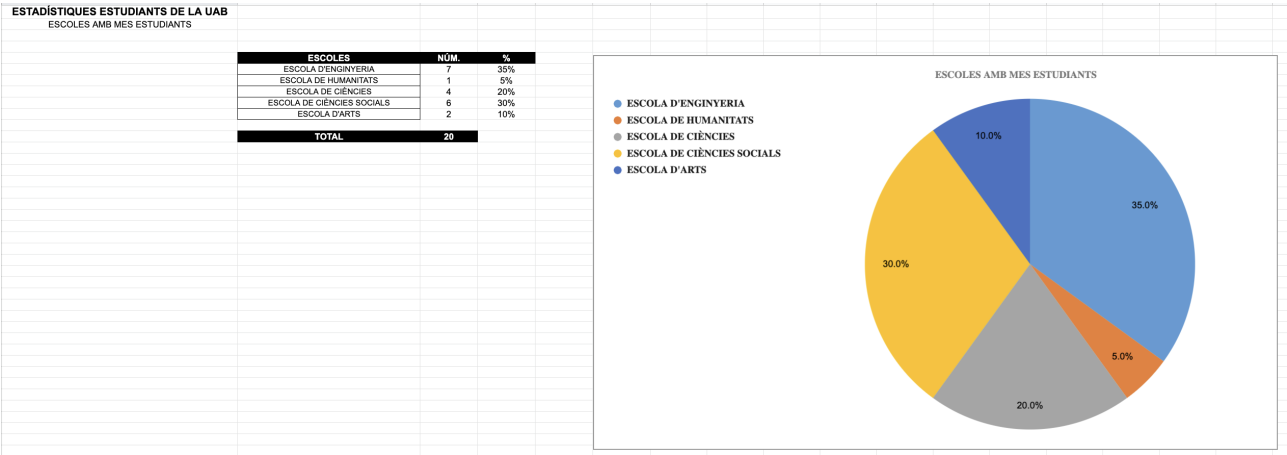


Fig. 16: Expected view of a KPI page within the exported excel