

Degree in Informatics Engineering

Informatics Project

PropertEase - A Unified Property Listing Manager

Technical Report

Advisors:

Professor Dr. Osvaldo Pacheco

Professor Dr. Rui Costa

Daniel Ferreira

Students:

Bárbara Nóbrega Galiza – 105937

Diana Miranda - 107457

Miguel Figueiredo - 108287

João Dourado - 108636

Ricardo Quintaneiro - 110056

June 4, 2024

Abstract

Managing property listings across various online platforms can be a cumbersome and time-consuming task. Each platform has its own interface and requirements, compelling property owners to manually update details, availability and pricing separately for each listing service. This process is not only tedious but more prone to errors, such as inconsistencies in property descriptions, double bookings due to desynchronized calendars and outdated pricing information that can result in a property owner losing revenue or having unsatisfied customers.

PropertEase is a platform that simplifies property management by centralizing it into a single interface, automatically updating property details across all services and synchronizing reservations into a unified calendar.

One additional very important feature is the price recommendation, which enables the user to keep the price up to market trends through the use of a combined set of algorithms, such as Machine Learning prediction and Google Trends analysis for real-time demand fluctuations. By doing this, *PropertEase* offers a dynamic pricing recommendation and an automatic price setting functionality to users.

This report outlines *PropertEase*'s context, requirements, engineering design, features, validation and concluding remarks.

Contents

1	Introduction	5
2	State of the Art	7
3	Requirements Gathering	11
3.1	Functional Requirements	11
3.2	Non-Functional Requirements	12
3.3	Actors	14
3.4	Use Cases	15
3.4.1	Model	15
3.4.2	Description	17
4	System Architecture	21
4.1	Architecture	21
4.1.1	Presentation Layer	22
4.1.2	Business Layer	24
4.1.3	Backoffice Layer	27
4.1.4	Message Queue	29
4.2	Domain Model	29
4.3	Internal Lifecycle	32
4.4	Physical and Technological Model	39
4.5	Price recommendation algorithm	40
4.5.1	Property Features ML Model Recommendation	41
4.5.2	Google Trends Analysis Recommendation	42
4.5.3	Internal Price Variation Recommendation	44
4.5.4	Final Price Recommendation	44
4.5.5	Price Recommendation Lifecycle	45
5	Results	47
5.1	Price Recommendation	47
5.2	Use Cases Materialized	47
5.3	Quality Assurance	57
5.3.1	Contract Tests	57
5.4	Usability Testing	57

5.4.1	Sample	58
5.4.2	Method	58
5.4.3	Results	59
5.5	Data Correlations	66
6	Discussion	69
6.1	Usability Tests	69
6.2	Price Recommendation	70
7	Conclusion	71
7.1	Summary	71
7.2	Main Results	72
7.3	Future Work	73
8	Appendix A - Business Layer API Documentation	77
8.1	User Service API documentation	77
8.2	Property Service API documentation	84
8.3	Calendar Service API documentation	94
9	Appendix B - Usability Tests	115
9.1	Tasks	115
9.1.1	Tasks Responses	118
9.2	Post Task Questionnaire	123
9.2.1	Post Task Questionnaire Responses	128
10	Appendix C - Example Property Schema	135
11	Appendix D - Contract Test Source Code Example	137
12	Appendix E - Detailed Sequence Diagrams	139

1 Introduction

Tourism has always been one of the most important sectors in Portugal's economy and now it is no different, representing roughly 15.6% of the country's GDP in 2022, as seen in Figure 1 (Instituto Nacional de Estatística, 2022). Furthermore, in 2023, overnight stays grew 10.7% and guests increased by 13.3% compared to 2022 (Instituto Nacional de Estatística, 2024). Now, more than ever, with the prospect of overcoming the COVID-19 aftermath, there is a palpable sense of eagerness among travelers to reconnect with loved ones, indulge in long-awaited vacations and explore new destinations, which Portugal wants to be a part of.

The resurgence of tourism plays a critical role in driving demand for rental properties, as travelers increasingly seek unique and personalized accommodation experiences beyond traditional hotels. In this current rental landscape, managing property listing across multiple hosting platforms in intents to maximize revenue is common, but it doesn't come without its complexities. The task of synchronizing calendars, setting competitive prices, and efficiently managing listings poses a complex and time-consuming challenge for property owners, greatly affecting their ability to attract guests and maximize revenue. In order to address these issues and assist owners in saving time and maximizing revenue, *PropertEase* aims to simplify property management by automating these processes through an innovative solution. Additionally, the goal is not only to streamline listing management for owners but also to empower them to make informed and strategic decisions that optimize the performance of their properties through data collection and detailed analysis.

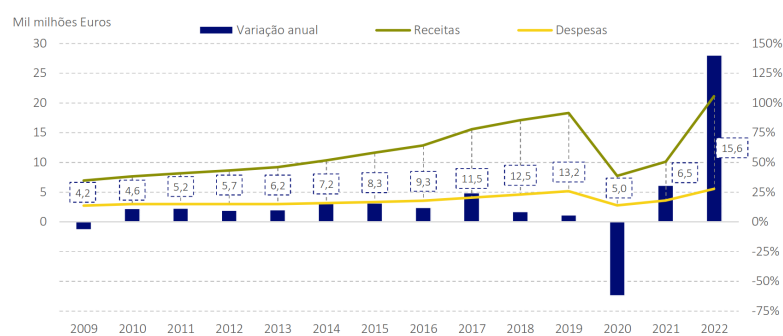


Figure 1: Portuguese Tourist Balance, Travel and Tourism, 2009-2022

In alignment with the European project ATT - Acelerar e Transformar o Turismo e Plano de Recuperação e Resiliência, *PropertEase* aims to gather data and gain insights into, for instance, the correlation between the quantity of websites showcasing a property and its occupancy rate.

This report describes the full engineering process behind the development of *PropertEase*, from design and architectural decisions to implementation, validation and some critical remarks of the developed work. The source code repositories are hosted in the *PI-PropertEase*¹ GitHub organization.

¹<https://github.com/PI-PropertEase>

2 State of the Art

After conducting a research into products currently available in the market similar to *PropertEase*, we elaborated the following overview of the various services and their relevant features according to our vision and intended use case:

- **Holidu**² is a company that promises to transform the vacation accommodation booking process, making it simpler and faster. They offer a product that can be used by guests to make their reservations, as well as providing the option for property owners who wish to centrally manage all their accommodation listings across various platforms to do it directly in their system. Their product features the functionality to synchronize calendars to prevent double bookings. Additionally, it conducts market analysis to advise owners on the optimal pricing for each property to maximize revenue, and enables price synchronization. Owners need to register to access these services, pay a one-time activation fee, and also pay commissions for each booking made in their accommodations through Holidu.
- **Icnea**³ is a product that offers property owners a Channel Manager to manage all their accommodation listings published on various booking platforms, aiming for centralized management of all their listings and reservations. It provides real-time calendar synchronization to avoid overbooking, allows users to synchronize rates and prices and applies differentiated sales strategies for each platform. It also enables the transmission of listing content (titles, descriptions, and photos) to the various platforms, facilitating the creation of listings. Besides having all these features, Icnea has a wider reach relative to what it allows you to do. It has a defined reservation and guest workflow that automates some of the processes related to the reservation such as: sending a reservation reminder and a thank you email, elaborating a weekly list of check-ins and checkouts, planning cleaning staff and managing client's comments. Moreover, it allows for the personalization of emails and its sending time, automated payments, and the gathering of statistics related to the growth of your business and to handle bureaucracy required by government. However, this product has a monthly cost for the user, which increases according to the number of properties.
- **AvaiBook**⁴ is the most complete among all studied products. Its Channel Manager allows calendar synchronization and custom tariff setting for each platform, thereby enabling

²<https://www.holidu.pt/partners>

³<https://icnea.com/>

⁴<https://www.avaibook.com/en/>

price synchronization. Additionally, it offers a Revenue Management System that provides property owners with essential information about their competitors, enabling them to adjust pricing strategies, cancellation policies, communications, etc., according to the market. Furthermore, it has an incorporated property management system that has features such as: centralized management of all descriptions, images, amenities and usage rules which will be automatically synchronized with the different portals through the Channel Manager. It also provides a price range for each date, allowing property owners to intelligently manage their revenues according to desired strategies. This product has a monthly cost and offers two types of packages: standard and pro. The cost varies not only according to the package type but also based on the number and type of properties the user owns.

- **Avantio**⁵ is a local accommodation management tool that offers a variety of useful features for property owners. With its Channel Manager, it's possible to synchronize content and calendars across multiple booking platforms, providing more efficient management. It also enables the user to create personalized websites or to integrate existing ones with its software. Additionally, it simplifies rate automation and the definition of occupancy rules, making the entire process more agile. Another characteristic of Avantio is its ability to gather and present data. It provides property owners with guest reservation statistics and trends, facilitating quick and effective data analysis. Furthermore, it allows for customization and exporting of reports for a deeper business analysis. All of this is available through a monthly subscription, ensuring continuous access to all its services.

This listing of property management systems and their characteristics is not exhaustive, but it is representative of the concerns that were taken into account when designing the solution. From what it is possible to learn from previous platforms, in any property management system there are essentially four types of features:

1. **Centralization of properties and automated bidirectional interaction with property listing websites:**

- (Holidu, Icnea, AvaiBook, Avantio) Single calendar that synchronizes every time someone makes a reservation on any property listing website. This reservation should propagate to every other website associated with *PropertEase*. This feature was common across all different property management platforms;
- (Icnea, Avaibook) Property info page that contains all information related to a prop-

⁵<https://www.avantio.com/>

erty. This information should appear in a listing in all related listing websites and, if changed, should be propagated accordingly to them. This information should include name, location, owner info, property photographs, occupancy rules and any other information to be showcased to the user in relation to the property. This way, progress is made towards the goal of increasing separation between the platform and its depending listing websites while interacting with them using a single system.

2. **Statistics and maximization of revenue:**

- (Holidu, AvaiBook) Market analysis to advise owners on the optimal pricing for each property based on its characteristics. This way, the property owner can know where he/she stands relatively to the market and, consequently, make more informed choices;
- (Icnea) Reservation statistics from different points of view such as by accommodation, region, owner or channel. With this information, the user can have a general perspective of the business' well being.

3. **Reservation and guest workflow** - workflow related to the reservation process, guest entry and exit and, possibly, property maintenance (it may include mail or message automation and calendar updates):

- (Icnea) Linking each customer to a property and embedding customer chat on the platform so that the owner can easily know from which property is each client and can attend to their needs.
- (Icnea) Sending a reservation reminder to the users when the reservation date is approaching. This way, the property owner's credibility and customer relation is increased while assuring the user that their reservation has been processed. Following the same reasoning, automating emails to send a thank you message to the customer welcoming to come back;
- (Icnea) Automated summaries of check-in and check out lists. In this manner, the property owner can always be aware of the property occupation and can take appropriate action if needed;
- (Icnea) Cleaning staff managing features.

3 Requirements Gathering

3.1 Functional Requirements

- Property related requirements:
 - FR1: the property owner must be able to see different available listing services and connect to them.
 - FR2: a property owner must be able to import their listings and reservations from a supported property listing website upon connecting to it.
 - FR3: a property owner must be able to view their current listings.
 - FR4: a property owner must be able to see the details of a property, such as price, location and amenities. They should also be able to see a URL that redirects them to their property for each listing service that property is in.
 - FR5: a property owner must be able to see, for each property, what reservations have been made.
 - FR6: a property owner must be able to get a price recommendation on a property according to houses with similar characteristics and to the external market analysis.
 - FR7: a property owner must be able to view a calendar with filled slots representing occupied properties and empty slots representing time slots without reservations. This calendar should represent the synchronized state of calendars in connected listing services.
 - FR8: a property owner must be able to choose a property and, in the property details page, set its price to a new value. This should update the price on all other websites the property is listed in.
 - FR9: a property owner must be able to decide, when setting a price, whether it's the amount of money they will receive (after commissions are applied) or the amount of money that will be paid by the customer. Since commissions are different depending on the listing service, the owner's property on each website will have the correct value based on his choice. Example: they want to receive 200€ - Booking.com will have a 204€ cost and AirBnB 206€ (as both websites will take their own commissions).
 - FR10: when a reservation is made in a certain website, it must be reflected in all

- other websites - calendars must be synchronized.
- FR11: a property owner must be able to set availability date manually. For example, when construction work happens for a month or cleaning happens for a day, the property owner must be able to lock out that particular property from receiving reservations for the time being.
 - FR12: property owner must be able to filter calendar by property, showing only reservations related to the selected property.
 - FR13: a property owner must be able to send emails to customers with a key code for them to enter the property.
 - FR14: a property owner must be able to see the current status of each one of their properties, to know whether they are currently free, occupied or have a check-in/checkout soon. They should also be able to filter this list by status.
 - FR15: a property owner must be able to choose whether to receive reservations 100% automatically or work under request where you are required to accept the reservation manually through a notification.
 - FR16: when editing the price of a property, the system must allow the possibility of considering other factors into the price calculation, such as cleaning cost and maintenance cost.
- Authentication related requirements:
 - FR17: in order to use any features, all users must be authenticated: both property owners and administrators.
 - Administration related requirements:
 - FR18: the administrator must be able to remove an user account.
 - FR19: the administrator must be able to remove user's properties.

3.2 Non-Functional Requirements

- Data security:
 - Ensure that user data, including property information, is protected against unauthorized access, manipulation or loss. The application must guarantee compliance

-
- with the GDPR (General Data Protection Regulation) ⁶ and implement encryption protocols for both stored data as well as for the network connections needed.
- The application must employ industry-standard cryptographic hashing algorithms to securely hash user passwords before storage. It also should enforce password complexity requirements (e.g. minimum length, special characters, etc.).
 - User and property data must be logically segregated to prevent leakage between different services.
 - Anonimization of users or properties between services should be conducted with the use of internal IDs.
 - Users inputs should be sanitized to avoid common security problems such as XSS, SQL injection or CSRF.
- Performance and Scalability:
 - The application must be capable to handle 100 concurrent users without compromising performance. It must be scalable in order to cope with future growth in the number of users and volume of data.
 - Page load times must not exceed 2 seconds under normal operating conditions.
 - Usability:
 - The user interface should be intuitive, easy to use and accessible to a wide range of users, including those with different levels of technology experience. The application should provide clear guidance and feedback for its users, making navigation and tasks simple and efficient. This should be validated with usability tests for the application with a task success rate of more than 68% (Sauro and Lewis, 2016) in total.
 - Reliability and Availability:
 - The application must be highly reliable and available at least 95% of the day, minimizing downtime and interruptions. It must be able to resiliently cope with failures and guarantee rapid recovery in the event of problems.
 - The Mean Time between Failures (MTBF) for the application should be 30 days and the Mean Time to Recover (MTTR) should be of 4 hours.

⁶<https://gdpr-info.eu/>

- **Compatibility:**
 - The website should be compatible with a variety of browsers and types of devices. The percentage of features or components that aren't seamless between all browsers/devices should be less than 10%.
- **Interoperability:**
 - The platform must be able to integrate well with real-estate platforms in order to perform its primary functions, such as calendar synchronization, property management, etc. The downtime of such integration when the platform is under normal operating conditions should be limited to the service provided by these external platforms.
- **Maintenance and Extensibility:**
 - The application must be easy to maintain and update, with clean and well documented code. It should allow new features and functionalities to be added efficiently. To achieve this, it should use a micro-services architecture or similar that focuses on modularity, ease of extensibility and update or replacement of certain services. For example, adding a new external listing service should be straightforward and not require changing code.

3.3 Actors

Table 1: System actors

Actor	Role
Property Owner	The main user of the application, manages one or more properties through the system
Administrator	Responsible for assuring the smooth operation of the system and the compliance with security and legal requirements

3.4 Use Cases

3.4.1 Model

Initially, on the requirements gathering phase, an initial Use Case diagram was made, which can be seen at Figure 2 below:

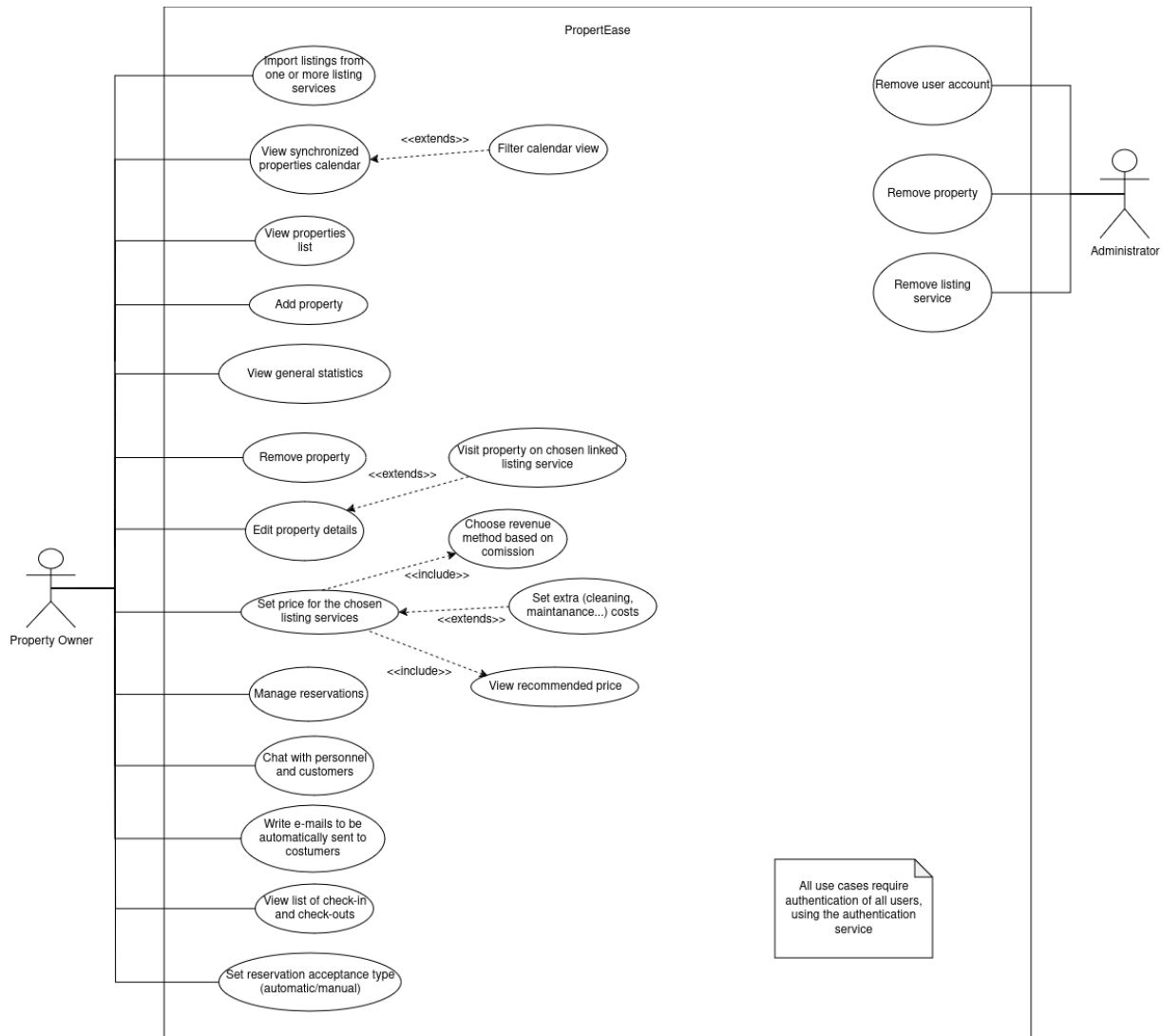


Figure 2: Initial Use Case diagram for *PropertEase*.

Within the next iterations, new ideas for features were given by students and the advisors and some limitations were found, such as the inability to use real listing service APIs, leaving us to simulate these. This lead to the creation of a new Use Case diagram, the **final version**, displayed in **Figure 3**. In the following diagram, the red dashed outline represent the use cases and actor that were low priority and would be interesting to have, but couldn't be done on time for the MVP.

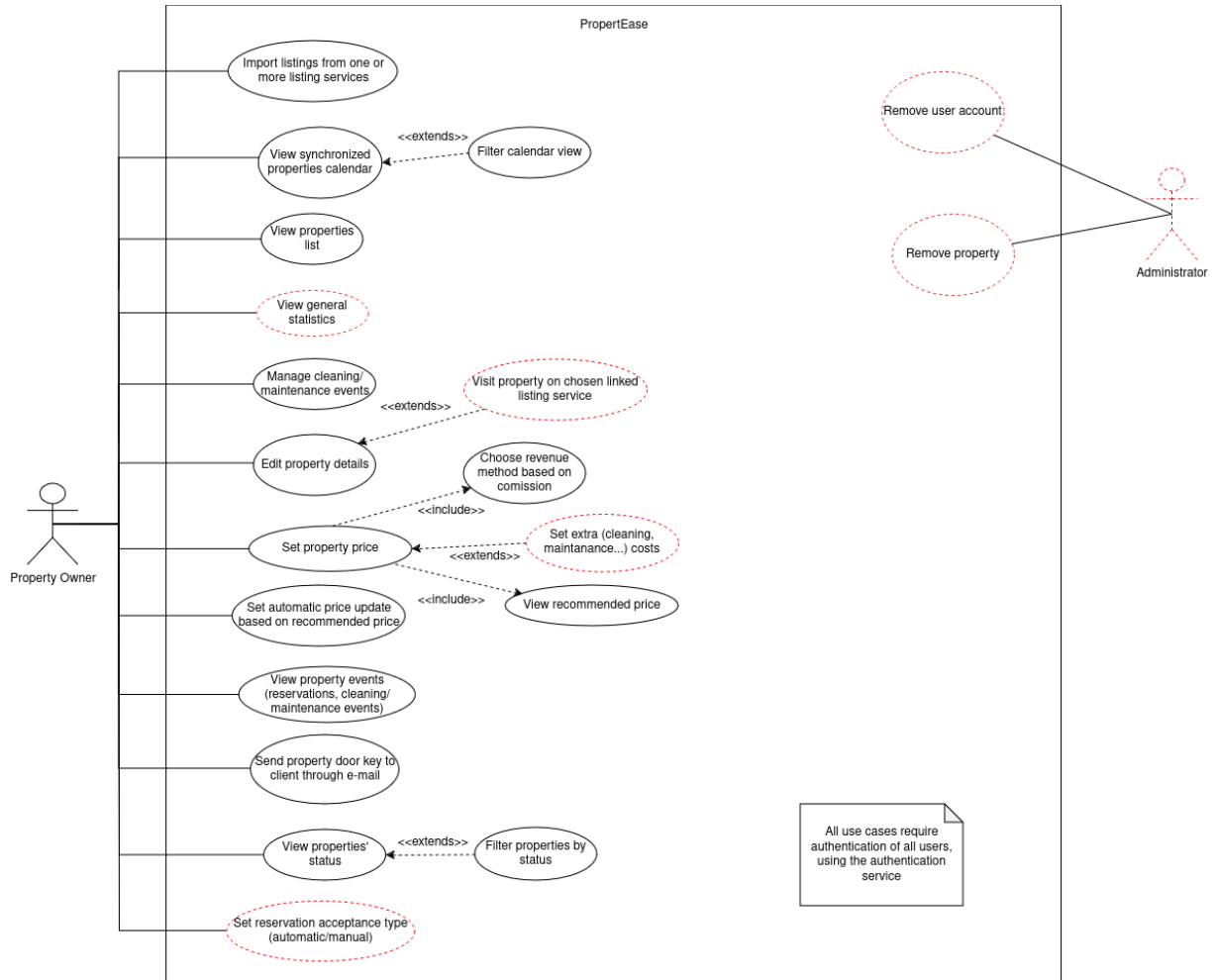


Figure 3: Final Use Case diagram for *PropertEase*.

3.4.2 Description

Table 2: Use cases' description

ID	Use Case	Description
1.1	Import listings from one or more listing services	A property owner can connect to external listings services and import all data (reservations and properties) related to their properties' listings from those services
1.2	View synchronized properties calendar	After having listings on our service, a property owner can view a calendar of the all the events associated with the properties that are synchronized across all external websites
1.2.1	Filter calendar view	A property owner can filter the events listed in the calendar by external listing service and by property
1.3	View properties list	A list of properties owned/managed by the property owner can be viewed
1.4	Manage cleaning/maintenance events	Property owners can manage events of cleaning or maintenance for each property. He/she can create, remove or update those events. These events set the property status as occupied during that period of time, as a reservation does
1.5	Edit property details	A property owner can change details of a property and those details are then synchronized across the connected services the property is in

ID	Use Case	Description
1.5.1	Visit property on chosen linked listing service	Allows the property owner to visit the property page on a listing website of choice to check if the details match
1.6	Set property price	A property owner can set the price of a property and the update should propagate to all listing services the property is in
1.6.1	Choose revenue method based on commission	A property owner can choose if he/she wants to include or exclude the commission of a external listing website when setting the price of a property
1.6.2	View recommended price	A property owner can view the recommended price of a property given to them by <i>PropertEase</i> 's price recommendation algorithm
1.6.3	Set extra (cleaning, maintenance...) costs	The ability to set extra costs to property prices, if these properties regularly have cleaning or maintenance occurrences, and consistently cost the property owner money
1.7	Set automatic price update based on recommended price	The option to allow property owners to set their properties to automatically update their price to the recommended price upon receiving price recommendation
1.8	View property events (reservations, cleaning/maintenance events)	A property owner can view a list of all the events (reservations, cleaning and maintenance) of one of the properties

ID	Use Case	Description
1.9	Send property door key to client through e-mail	After a reservation is done by a client, a property owner can send a property door key code through e-mail to that client
1.10	View properties' status	A property owner can view the status of all properties on the list of properties to see if it is free, occupied or has a check-in/checkout soon
1.10.1	Filter properties by status	The owner can filter properties on the properties list by status
1.11	View general statistics	Property owners can view general statistics of all the properties and events, such as revenue per property, revenue per listing service, rate of occupation along a time interval, etc...
1.12	Set reservation acceptance type (automatic/-manual)	A property owner can choose if he/she wants to automatically accept reservations or if he/she wants to do it manually
2.1	Remove user account	Allows the administrator to remove user's account on demand, in a case where its removal needs to be forced
2.2	Remove property	Allows the administrator to remove a property on demand, in a case where its removal needs to be forced

4 System Architecture

4.1 Architecture

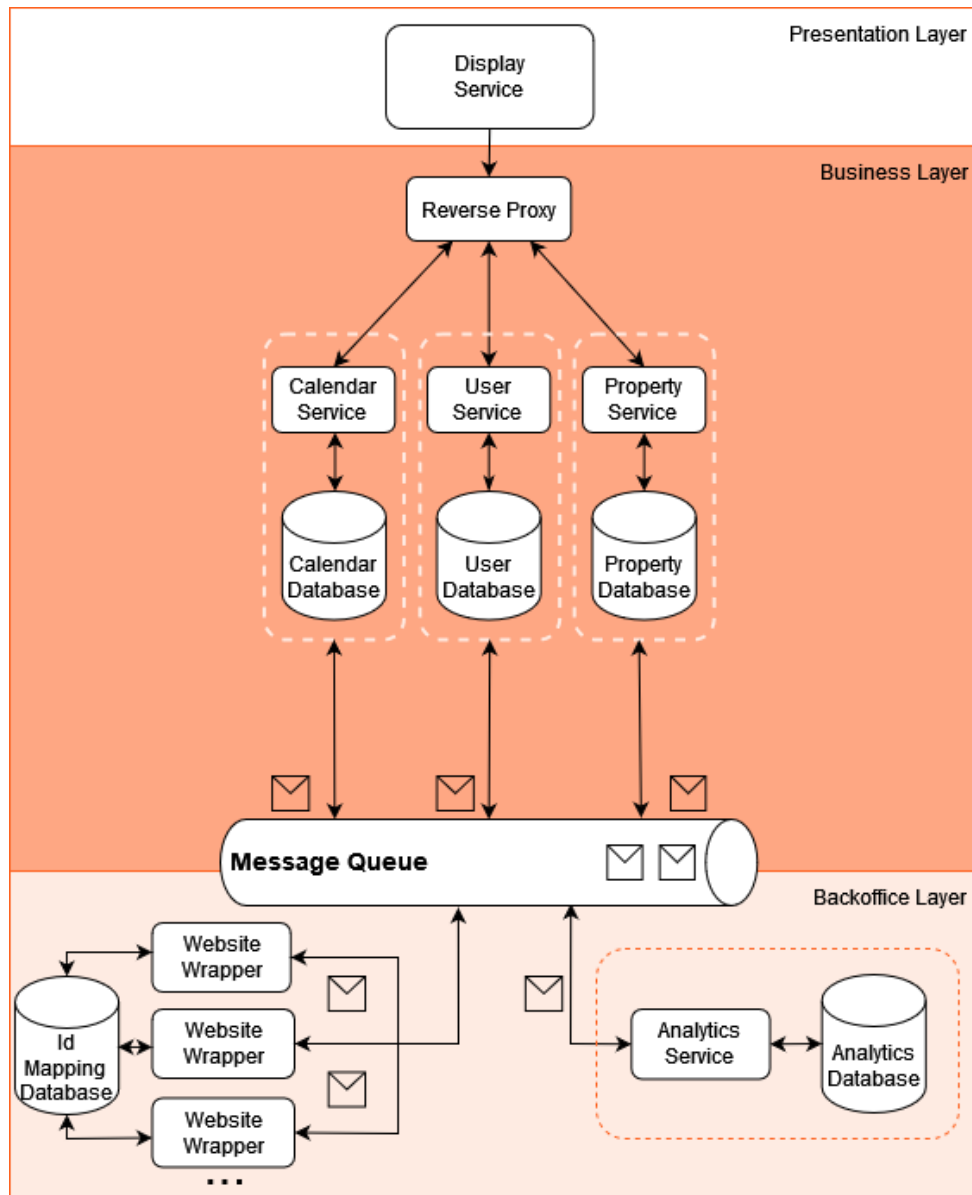


Figure 4: System architecture for *PropertEase*

The *PropertEase* system operates according to two different aspects: one related to the client-side and another one related to data aggregation and availability, thus requiring a segregation of responsibilities. Regarding the data aspect, it is built using metrics measured across all parts of the system, with a microservices architecture being ideal to measure them first at the source and afterwards as an aggregate. Each microservice is independent, has a single responsibility (separation of concerns) and should be deployed as a separate unit, allowing an easier and

better deployment and decoupling. This way, the change or stoppage of one of the microservices will not affect the proper functioning of the others. This is most important as the system is dependent on external services that provide the properties and reservations it aggregates and manages.

4.1.1 Presentation Layer

The presentation layer is a sole microservice layer. The **Display Service** is responsible for providing the user interface and handling interaction with the user. It is also responsible for forwarding the requests to the reverse proxy, which will in turn delegate it to the appropriate service.

It is implemented as a *React*⁷ web application. *React* is a very popular⁸ reactive JavaScript and TypeScript library for creating user interfaces. The React app was created using *Vite* over *Create-React-App (CRA)* as it is considered deprecated and contains many critical security vulnerabilities that could lead to security issues in *PropertEase*. *Vite* also includes a development server with *Hot Module Replacement (HMR)* which speeds up development significantly.

Typescript was chosen over JavaScript due to its static typing capabilities, which makes it easier to find bugs early on in development, such as accessing null or undefined values, increasing overall code quality. The TypeScript compiler also catches compiler-time bugs, as opposed to the dynamically typed and interpreted language JavaScript.

As for styling the user interface, *Tailwind CSS*⁹ was used. It replaces the typical verbose and error prone vanilla CSS styling with class-based styles that, in conjunction with *IDE* support¹⁰, helps to increase developer productivity.

For the state management in this web application, *React TanStack Query*¹¹ was used. It allows easy global state management across multiple pages and *React* components and has direct support for some specific features we defined as requirements in *PropertEase* to provide a better user experience, such as:

- Query caching: queries for data such as a particular user's properties and reservations are used in multiple *React* components, and *React TanStack Query* has built-in caching to prevent the need for re-fetching, which wastes network resources and adds unnecessary

⁷<https://react.dev/>

⁸<https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/>

⁹<https://tailwindcss.com/>

¹⁰<https://marketplace.visualstudio.com/items?itemName=bradlc.vscode-tailwindcss>

¹¹<https://tanstack.com/query/v3>

load to the API layer;

- Query invalidation: queries can be cached, but, as new data gets created and data changes, it is crucial to have a mechanism for invalidating cached data to prevent stale data;
- Periodic re-fetching: as new reservations come in, it is crucial to show them to the user as soon as possible. This is another mechanism to prevent stale cache data, as the built-in cache can be periodically invalidated to detect new data by re-fetching;
- Loading states: know when a query is in progress and display the page in a loading state, so the user can understand the page is not yet ready;
- Error states: know when a query failed in order to show error messages for direct feedback to the user.

For displaying calendars with property event data (reservations and cleaning/maintenance events) we used *FullCalendar*¹², as it fit our requirements. It supports multiple views of calendars, such as fully detailed timeline view, with the possibility to aggregate events based on some attribute (for *PropertEase*, we aggregated events by property) and a smaller calendar that displays events in a concise way, allowing for a view of events in a particular week.

In order to allow users to create management events, such as cleaning or maintenance events, we needed to allow users to pick a date and a time (simultaneously) in which those events occur. For that purpose, we used *Flatpickr*¹³, as the default HTML date-time input does not fully support all browsers (in particular, Mozilla Firefox¹⁴) and we don't want to hinder our users' experience if they use those browsers¹⁵.

Table 3 has a brief description of all pages available to access in Display Service.

¹²<https://fullcalendar.io/>

¹³<https://flatpickr.js.org/>

¹⁴<https://www.mozilla.org/en-US/firefox/new/>

¹⁵<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/datetime-local>

Table 3: Description of all available pages in Display Service

Path	Description
/	Home page. Has a small description of <i>PropertEase's</i> features. The default page you see when not logged in.
/dashboard	The main dashboard the user can access to see their reservations in a small calendar and their properties in a table with the property's current status
/properties	List of properties page. Contains information related to each property's status (free, occupied, check-in soon, checkout soon), the nearest reservation's arrival and departure times and price.
/property/:id	Shows property information for property with given <i>id</i> parameter. In particular, shows all of the property's details, and has tables for reservation, cleaning and maintenance events. This is also where users can send e-mails for opening the door to their properties to clients, and also create, update or delete cleaning and maintenance events.
/calendar	Has a full view of a filterable calendar. Allows for filtering by property and by listing platform.
/integrations	Page for users to connect to external listing services, and therefore import their properties and reservations.

4.1.2 Business Layer

As the name indicates, the business layer is responsible for the business logic of the system. It communicates with the upward layer through a **reverse proxy**, implemented using *nginx*¹⁶, which will redirect and delegate the requests to the appropriate microservice. Furthermore, this component allows for load balancing and contributes to enhance the scalability and security of the system, as no service will be directly exposed to the end user. The microservices, implemented using *FastAPI*¹⁷, which handle the business logic of the system are the following:

- **User Service** - This service is responsible for handling user related actions, such as creating an account and signing in, supported by *Firebase Authentication*¹⁸. As we will later show in the domain model section, the information saved in this microservice database

¹⁶<https://nginx.org/en/>

¹⁷<https://fastapi.tiangolo.com/>

¹⁸<https://firebase.google.com/docs/auth/>

not only contains regular user information but also to which external services the user is connected to and intends to import properties and receive reservations from. This being the case, this will be the microservice that periodically sends requests through the message queue to import new properties or reservations. However, that doesn't mean that the User Service will be the one receiving the requested data as will later be shown.

- **Property Service** - This service is responsible for storing and providing any data related to properties, including pricing data. As this service contains the property information, it will also:
 - Generate events each time the user updates property information, as these changes need to be propagated to the external listing websites where that property actually is;
 - Distinguish between different properties through their address, upon importing them. If a property for the same user with the same address already exists, then it shouldn't be overwritten with the newly imported one. In the best scenario, it would even make sense for there to be a conflict resolution algorithm that minimizes the amount of data lost in the aggregation. This would even make more sense when thinking about the diversity of the data present in different property listing websites. The main objective would be to gather all that data in the property service database and every time an external property listing website wants to showcase one of the properties, it would only use the information that it needs. However, keeping track of what properties are from what external listing websites and connecting those properties to the one saved on this service's database is beyond the scope of this service, as it will later be discussed;
 - Send requests to the analytics services in two different schedules: 1. send the data it needs for the price recommendation algorithm and 2. send data to compute statistics that are relevant to the ATT project.
- **Calendar Service** - This service is responsible for orchestrating reservations between the various external websites and for their confirmation or denial in an automatic way, keeping calendars synchronized across multiple listing services. However, *PropertEase* isn't only an aggregation system where a property owner can see all of his properties and reservations from multiple property listing websites. It actually helps them to manage their properties, by allowing them to create other events in their calendar that will automatically

be propagated to those websites, marking them as occupied for the respective time frame (i.e. period of time where the property is under maintenance and therefore cannot receive people). The creation of these events and the generation of the related events to be propagated to the external websites are controlled by this service as well.

Table 4 shows a non-exhaustive list of exposed endpoints and description of their behavior, related to business layer microservices. Full documentation in Swagger¹⁹ format can be found in Appendix A.

¹⁹<https://swagger.io/>

Table 4: Non-exhaustive business layer API endpoints by Microservice

Microservice	Operation	Endpoint	Description
Calendar Service	POST	/events/reservation/{reservation_id}/email_key	Receives a key code to be sent by e-mail to the client associated with the reservation with ID equal to 'reservation_id'.
Property Service	PUT	/property/{property_id}	Updates the property details for the property with the specified property ID. Also propagates changes to the external listing services counterparts.
Calendar Service	POST	/events/management/-cleaning	Creates a new cleaning event with the specified time frame and creates corresponding events in external listing services to close down the property during the cleaning time interval.
Calendar Service	GET	/events	Returns all events for a certain property, including reservations and cleaning/maintenance events, in a general <i>Event</i> schema.

4.1.3 Backoffice Layer

The backoffice layer is divided in two different types of microservices, the first one being website wrappers. Each **website wrapper** is a module that encapsulates each property listing website (namely their respective API's) and is responsible for interacting with them and

propagating changes to and from them. As it wasn't possible to get access to a real API for our use case, we simulated API's for three made-up listing services: Zooking, ClickAndGo and Earthstayin.

On one hand, if a user makes a reservation in any of the booking websites, then that reservation should be propagated to all other property listing websites and should appear in the property owner's calendar.

On the other hand, if the property owner decides to change the price of one of their property listings, that change should also be propagated to all of the property listing websites.

Therefore, each wrapper generates events that use data coming from them. These events can be triggered as a consequence of certain actions on external listing websites (e.g. client booking a reservation for existing property, property owner creating a new property) or can be triggered within the system after the component processes property/reservation import requests or property details update requests (this request results from the update of the details of a property in the Property Service). For this to happen, it's required the existence of a database that keeps track between the mapping of internal IDs of the properties and reservations on the microservices and external IDs of those same properties and reservations on the external API's. It's important to note that each website wrapper constitutes a different microservice and should work independently from the others.

The fact that all microservices share one database doesn't go against this as each wrapper only uses their designated tables. Nothing would prevent each website wrapper of having an independent database in the future. The fact of using only one is merely an option derived of just being an ID mapping database, of the number of external services being supported and to decrease the complexity of development.

The communication between the Business and Backoffice layer should be bidirectional and is insured through a message queue for asynchronous communication, creating here an abstraction layer. Therefore, if a new property listing service is to be supported, it would just be necessary to create a new website wrapper that would start to generate and process events seamlessly relative to the Business Layer.

In addition, the Analytics Service is responsible for regularly calculating the recommended price for each property and sending properties data to the Elasticsearch database in order to ease the retrieval of aggregation analytics based on them. Both actions are triggered by scheduled asynchronous requests from the Property Service sent to the message queue. It employs the

*ELK*²⁰ stack, which simplifies the visualization of data for analysis purposes.

4.1.4 Message Queue

The message queue, implemented using *RabbitMQ*²¹, is be used to pass messages asynchronously between the Business and Backoffice layers. In the context of *PropertEase*, messages indicate events such as property details changes (pricing changes or other details changes), users connecting to listing services, scheduled reservation and property importing, which require communication among multiple components of the architecture.

Messaging is also used to periodically request the Analytics Service for price recommendations, given the properties existing in the Property Service. It is also used to send property data (without personally identifiable information, for security reasons) to the Analytics Service to be used for data analytics using *Kibana*²², which allows for easy data visualization through queries using the *KQL* (*Kibana Query Language*).

4.2 Domain Model

As the previous section detailed, our architecture follows the microservices pattern, and for that reason, there are multiple database to handle incoming data. This section depicts the conceptual modeling of all data *PropertEase* handles.

Figure 5 represents data modeling for the User Service, which holds all information related to users.

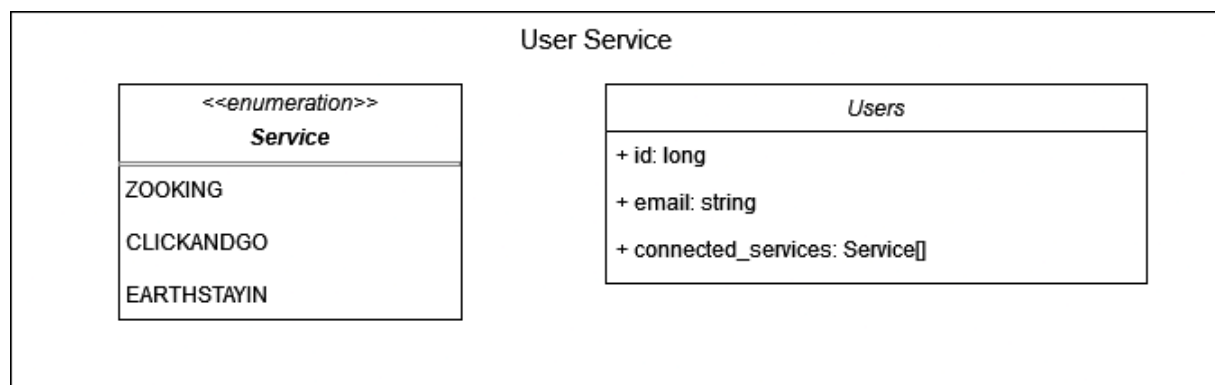


Figure 5: User Service Data Model

Figure 6 represents data modeling for the Property Service, which holds all data related

²⁰<https://www.elastic.co/pt/elastic-stack>

²¹<https://www.rabbitmq.com/>

²²<https://www.elastic.co/kibana>

to properties. Since property data coming from different listing services is expectedly different among them, we decided for a NoSQL database, particularly document-oriented using *MongoDB*²³.

As we handle data coming from multiple external APIs, their schema differ. For that reason, we modeled *Property* data as a combination of the most common attributes across popular listing services. We analyzed the schema of three different listing services - *Airbnb*²⁴, *Booking.com*²⁵ and *Vrbo*²⁶ - and ended up with the model found at Figure 6.

An example schema for a property was included in Appendix C.

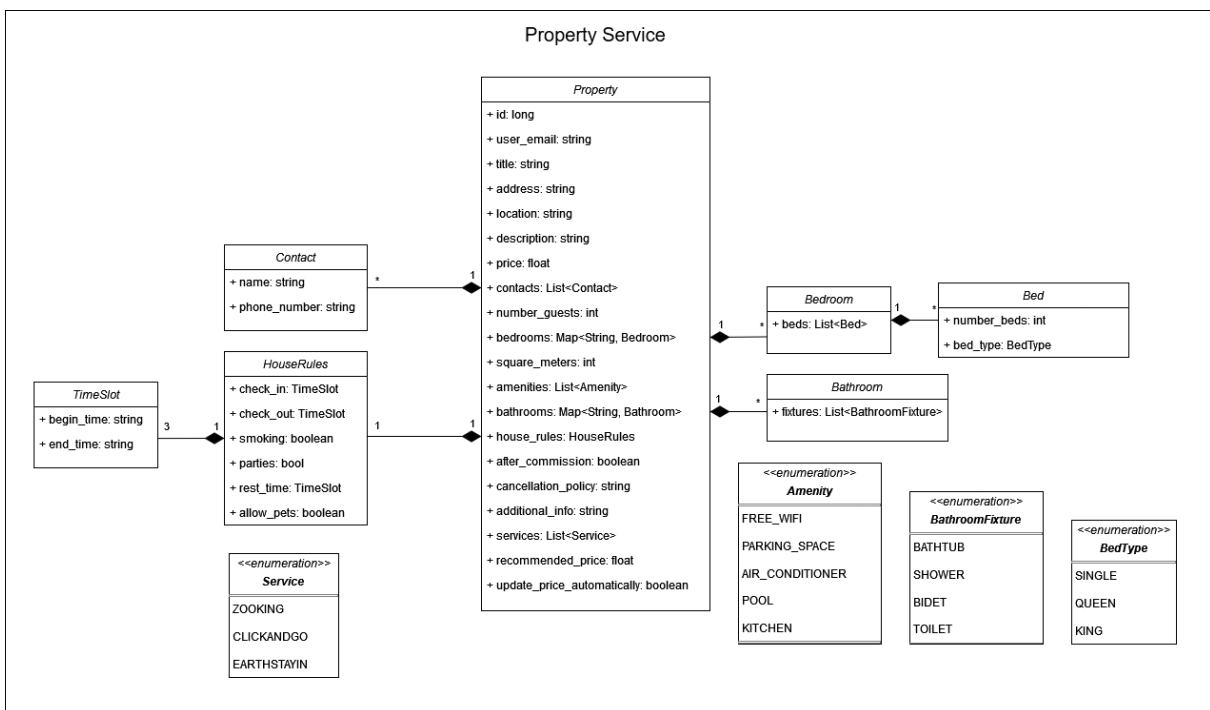


Figure 6: Property Service Data Model

Figure 7 represents data modeling for the Calendar Service, which holds all event (reservation, cleaning and maintenance) information. The database holding this data uses a relational database, particularly *PostgreSQL*²⁷.

As there are two different sources of events - interval events (created by the user - maintenance and cleaning) and external events (obtained from external API - reservations) - we used

²³<https://www.mongodb.com/>

²⁴<https://www.airbnb.com/>

²⁵<https://www.booking.com/>

²⁶<https://www.vrbo.com/>

²⁷<https://www.postgresql.org/>

inheritance, which increases extensibility greatly. If there is a need to add a new type of event in the future, we can simply extend one of the existing classes and work from there.

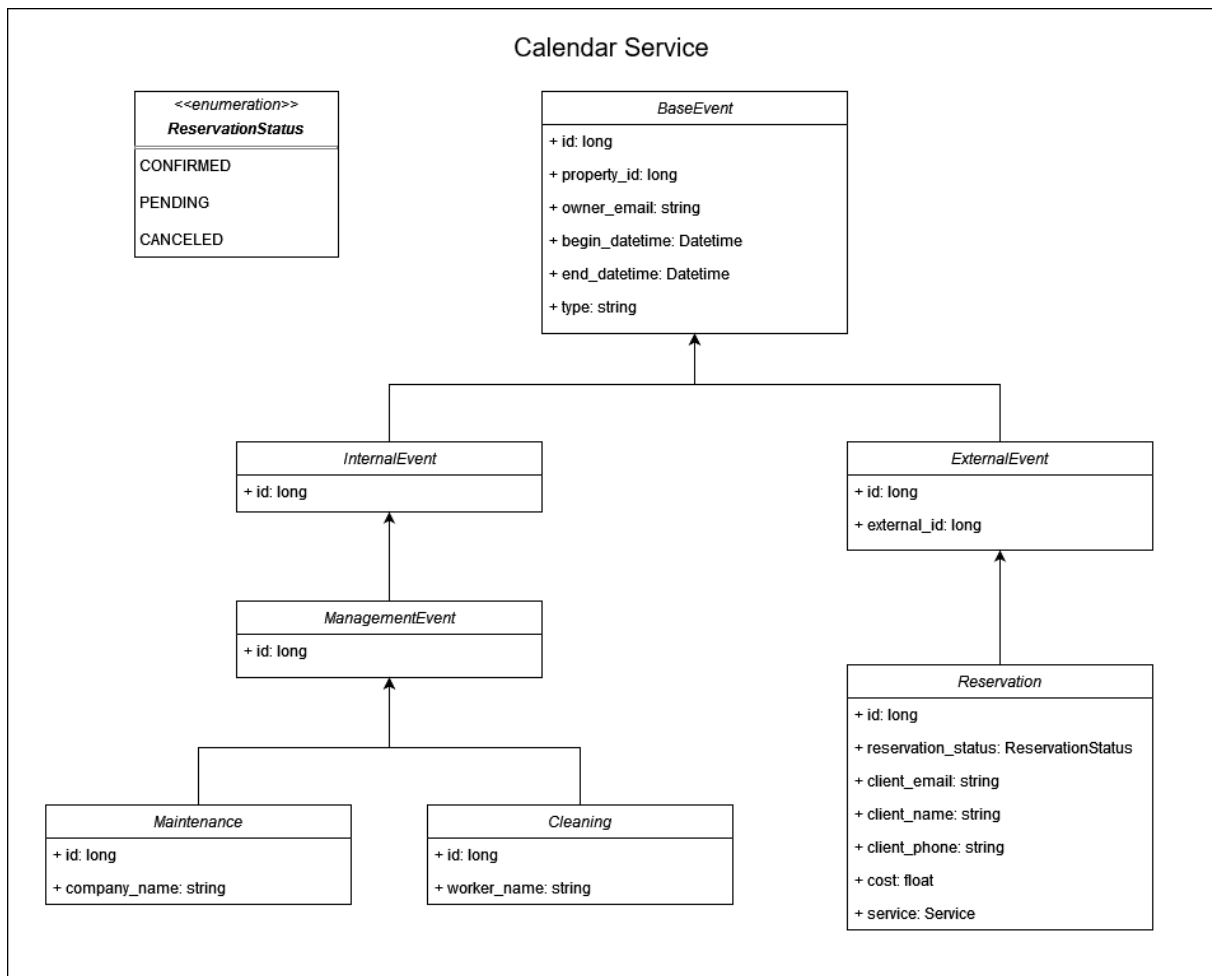


Figure 7: Calendar Service Data Model

Figure 8 represents data modeling for the Website Wrappers. These require a mapping between external and internal property and reservation identifier, so that they can efficiently detect changes occurring in the outside, and propagate changes that the user made. As this database is only used for mapping IDs from external services to internal services, it is supposed to be simple, and for that reason, we used *SQLite*²⁸, which prides itself for being a small, fast and self-contained implementation of SQL.

²⁸<https://sqlite.org/>

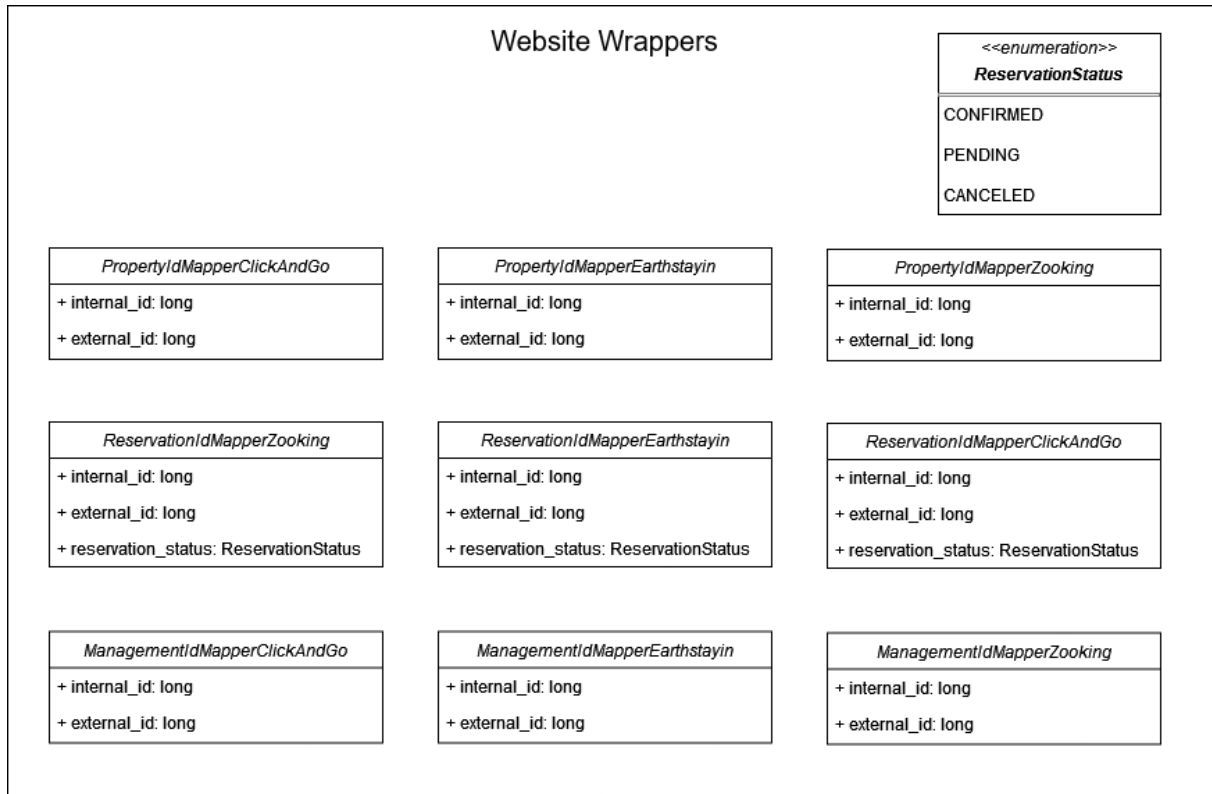


Figure 8: Website Wrappers Data Model

4.3 Internal Lifecycle

PropertEase main goal is to reduce manual work by property owners regarding property management. Therefore, *PropertEase* is responsible for the automation of these processes, namely the edition of property details, the synchronization of new properties and the synchronization of new reservations (including possible cancellation of a reservation). This section will dive further into the discussion of the architectural workflows behind these features.

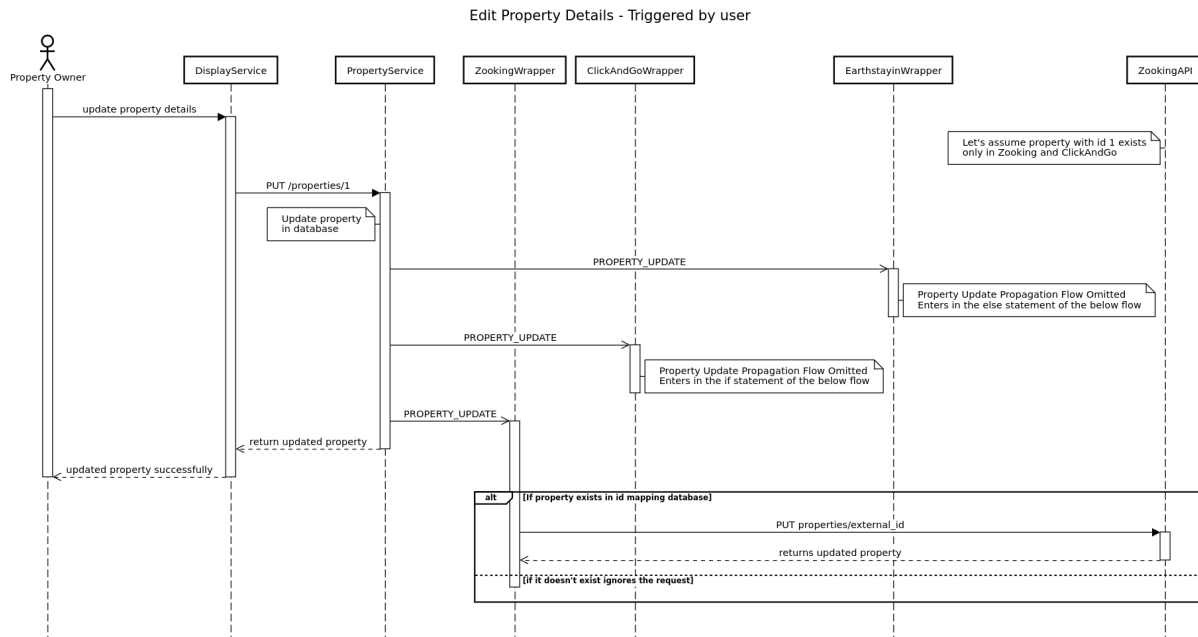


Figure 9: UML Sequence Diagram of edition of property details by a property owner

Figure 9 presents the full sequence diagram in order for a property to be updated not only on the Property Service, but also on all external services the property owner is connected to. However, this process is completely transparent to the user, as the propagation of the update to the wrappers, and consequently to external services, happens asynchronously.

1. The property owner updates property details through the user interface provided by the Display Service;
2. The Display Service calls a PUT request in order to update the matching property on the Property Service;
3. The Property Service processes the update and updates the corresponding property in the MongoDB database;
4. The Property Service broadcasts through the message queue a PROPERTY_UPDATE message to all wrappers;
5. The Property Service returns the updated property. It is worth highlighting that the property updated in the Property Service database was returned as soon as the messages were sent to the message queue. The service doesn't wait for the processing of the tasks related to such messages as all the propagation behavior happens asynchronously.
6. Therefore, this process is completely transparent to the user that will immediately see the

updated property on his dashboard.

7. Then each of the website wrappers will process the requests when they proceed to consume the `PROPERTY_UPDATE` message. Assuming that the edited property is registered on Zooking and ClickAndGo services, each of the wrappers would process the request in the following way:

- **Earthstayin Wrapper** - The wrapper will query for the property internal ID on its ID mapping database. Since the property isn't registered, the return value of the query came out empty. Thus, the wrapper will not proceed with the update request.
- **ClickAndGo Wrapper** - The wrapper will query for the property internal ID on its ID mapping database. Since the property is already registered, the return value of the query wouldn't be empty. The only thing left to do is consuming the appropriate API endpoint with the corresponding external ID in order to update the property in the external service. There is no need to update anything on the wrapper's database as it doesn't contain any property sensitive information;
- **Zooking Wrapper** - Since the property is also registered in this service, the wrapper will proceed in the same way as the ClickAndGo wrapper differing only on the conversion schema for the property and the API and corresponding endpoint.

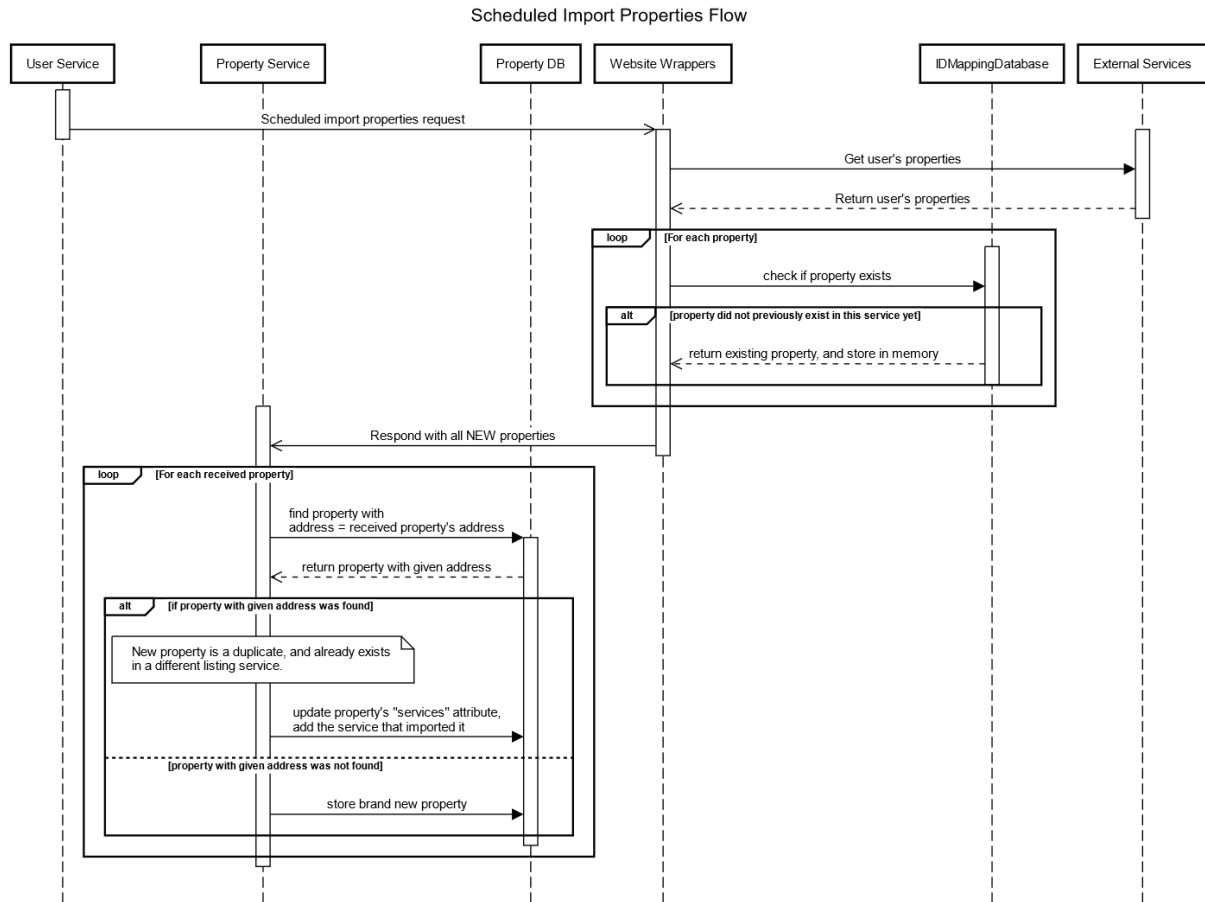


Figure 10: Simplified UML Sequence Diagram of scheduled flow to import properties from external services

Figure 10 presents a simplified sequence diagram that illustrates the scheduling of property imports across multiple listing services. The detailed version of this diagram can be found in Appendix E in Figure 38.

1. The User Service periodically sends a request to import properties for a list of users with their services;
2. Each website wrapper fetches all user's properties from the given external service API;
3. Then, the website wrapper checks if each of the returned properties exists in the corresponding ID mapping database;
4. Each property that doesn't exist is then persisted in the ID mapping database;
5. All new properties are returned asynchronously to the Property Service;
6. The Property Service checks for each of the received properties if there is an existing

property with the same address, which was the defined way to detect duplicated properties.

7. If a duplicated property is detected, the service from which it was imported gets added to the list of services for that property. The details of the property persisted in the database remain unchanged. However, for that to happen, a property with the same address for that user had to previously have been propagated from other external listing service.
8. If the imported property is a new property, then the property with all of its details is persisted into the database.

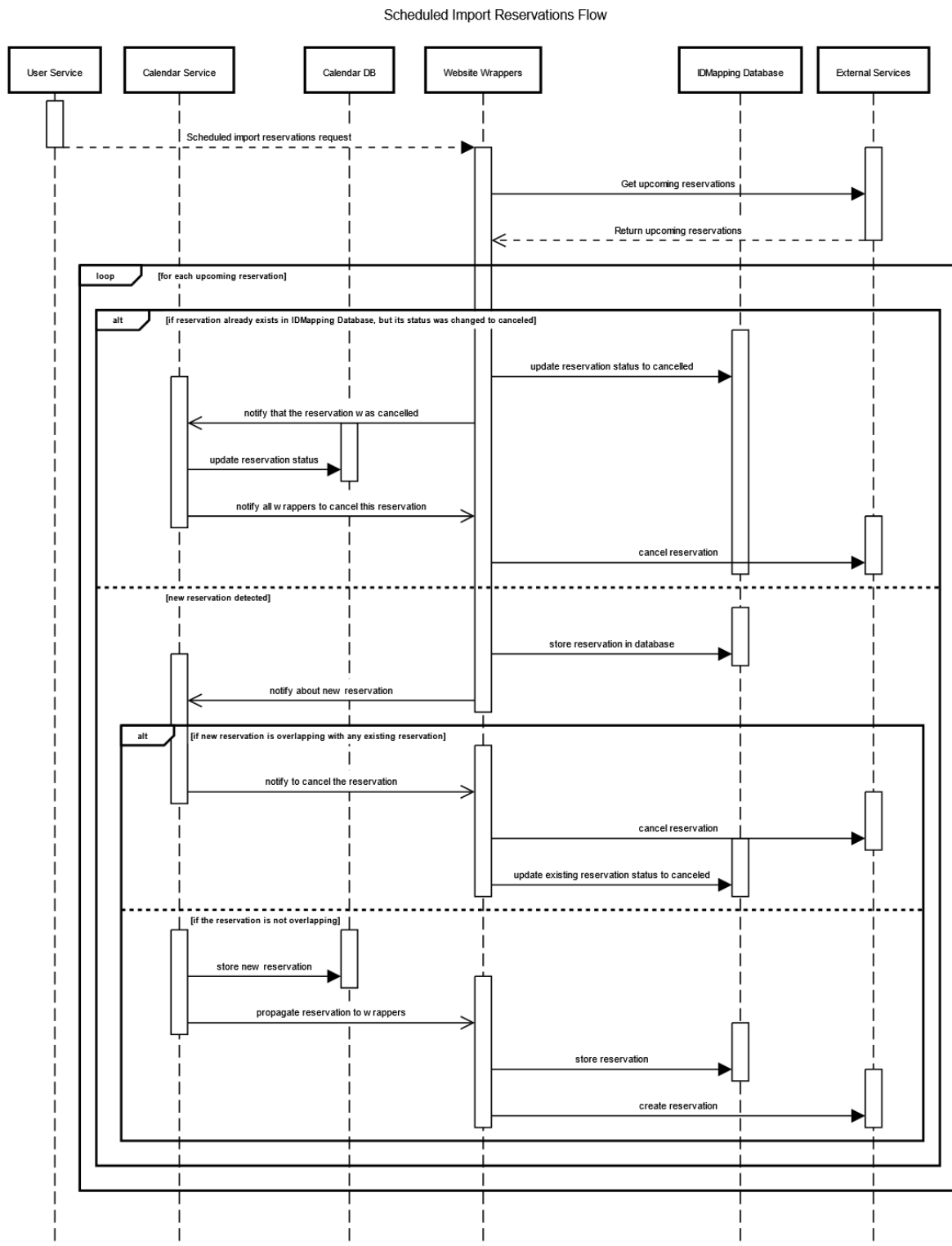


Figure 11: Simplified UML Sequence Diagram of scheduled flow to import reservations from external services

Figure 11 presents a simplified sequence diagram that illustrates the scheduling of property reservations across multiple listing services. The detailed version of this diagram can be found

in Appendix E in Figure 39.

1. The User Service periodically sends a request to import reservations for a list of users with their services;
2. Each website wrapper fetches all user's incoming reservations from the given external service API;
3. Then, there are two conditional branches according to the existence of the reservation in the website wrappers ID mapping database:
 - If the reservation already exists, but its status was changed to canceled:
 - (a) Update reservation status stored in mapping database to CANCELED;
 - (b) Notify that the reservation was canceled to the Calendar Service. Then, the change of status of the reservation will be propagated to the Calendar Service's database;
 - (c) Then, the Calendar Service will notify all wrappers to cancel the reservation.
 - (d) Each of the website wrappers will propagate the changes to the external services. Bear in mind this only happens for the website wrappers which have the corresponding property (and consequently the reservation) in their mapping ID table.
 - If it's a new reservation:
 - (a) The website wrapper will store the reservation in the ID mapping database - including the internal and external ID of the reservation and the respective reservation status;
 - (b) Then, notify the Calendar Service about the new reservation;
 - (c) If there is an overlapping event with the new reservation, then notify the website wrapper the reservation came from so that it can cancel it. When the wrapper receives this request, it should call an endpoint to cancel the reservation and update the reservation status on the id mapping database;
 - (d) If there isn't an overlapping event, then store the new reservation and propagate it to all wrappers.

- (e) Each of the wrappers can now store the reservation in their ID mapping table and consume the endpoint of their respective API to create/propagate the reservation.

4.4 Physical and Technological Model

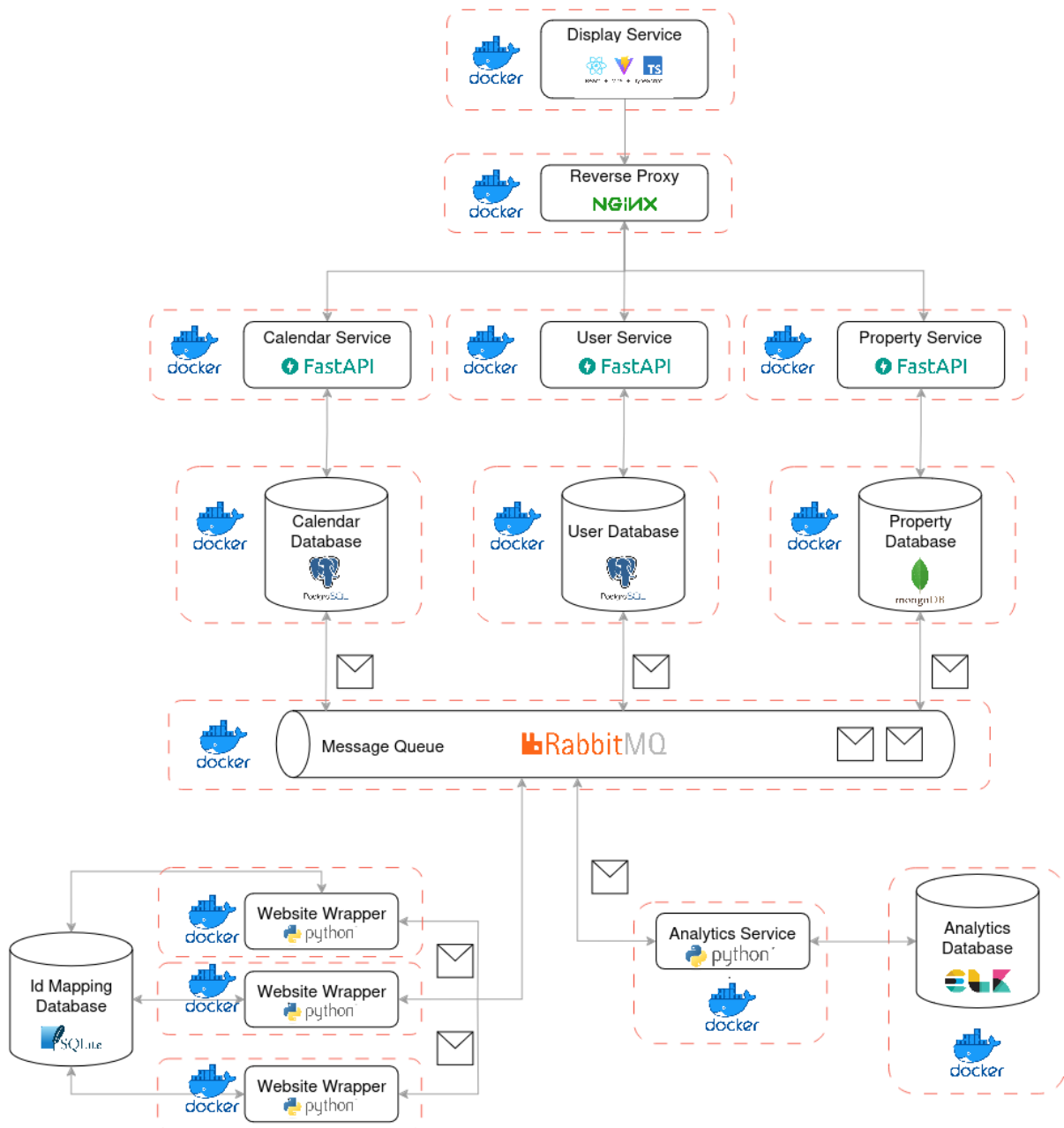


Figure 12: Deployment Diagram

Figure 12 presents the physical architecture of the system, detailing each component's technology and providing a description on how the components will interact between each other.

In terms of deployment, each component will operate in its own Docker container for

better operational efficiency, isolation, and security. Communication between the components will be restricted by networks, as follows:

- The Display Service communicates only on the 'frontend' network, through which it communicates with the Reverse Proxy. Only these two components run on this network, ensuring that the display service cannot communicate with any other components.
- The Reverse Proxy, besides running on the 'frontend' network, also runs on the 'backend' network. This allows it to communicate with the Calendar, User and Property Service.
- Each service in the business layer has its own network to communicate with its respective database. Additionally, all of them run on the 'message broker' network to communicate with the message queue.
- The Message Queue runs on three different networks to communicate separately with the Website Wrappers, the Analytics Service, and the Calendar, User, and Property services.
- All website wrappers runs on a single network to communicate with the message queue.
- The Analytics Service runs on two networks: one for communication with the message queue and another for communication with its database.

This network separation is important for the optimization of performance and security. It reduces congestion and latency, provides better control over access, and supports scalable and resilient system operations. This way, proper communication and maintenance can be ensured to sustain robust and secure architecture.

4.5 Price recommendation algorithm

For the price recommendation, three different algorithms are used: the first one (Figure 13.1) uses a collection of the property features (not including the price) to feed a machine learning model, produces the median of the predicted property prices and recommends a price based on the difference on features from each property to the median property; the second (Figure 13.2) uses the location of each property to fetch current trends tied to that location on Google Trends and reflects it's variation on the recommended price; and the third (Figure 13.3) calculates average internal (i.e. only properties linked to *PropertEase*) price variation and recommends the detected variation to properties which had no variation whatsoever.

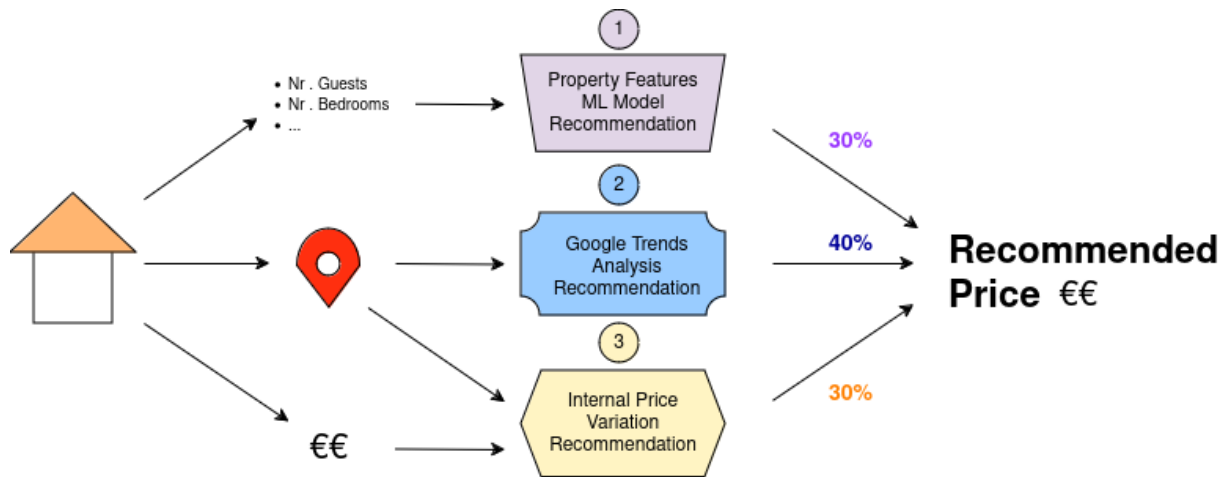


Figure 13: Simplified diagram of the price recommendation process

4.5.1 Property Features ML Model Recommendation

The goal with this component is to obtain the median property price and recommend the other prices in comparison with the median. A Random Forest Regression model is used for predicting each property's price, with the input features being: number of bathrooms, number of bedrooms, number of guests (allowed per stay), number of beds, number of amenities, latitude and longitude. The option to use this model, these features and how to obtain them was inspired by a Jupyter Notebook "Helping people in Lisbon to predict Airbnb prices" (Freitas, 2022) published at Kaggle, which was found to be the most adequate for the task after systematic research. The same Notebook also inspired the usage of the dataset for training, which is the 2021 Lisbon Airbnb data (InsideAirbnb, 2024).

To obtain the median property price, each property is sent individually (its features) to a Random Forest Regression model and then the median property in terms of price is obtained from the list of predictions. Next, each property is compared to the median property on each feature, and according to defined weights ²⁹, the price recommended will rise or decrease based on the result of the total weighted difference between features. The formulas 1 and 2 describe better this calculation: in 1, for each property p with n features with the values $p_0, p_1 \dots p_n$, and for the median property m with the features values $m_0, m_1 \dots m_n$, the total difference d is calculated using the feature weights w_i . Then, the total difference d is used in 2 to calculate the recommended price for each property, where $(1 + d)$ represents the percentage difference applied upon m , which is the median property price.

²⁹The weights were defined based on own observation and judgment

Table 5: Weights for feature comparison

Number of bathrooms	0.03
Number of bedrooms	0.05
Number of beds	0.08
Number of guests	0.07
Number of amenities	0.02

$$d = \sum_{i=0}^n (p_i - m_i)w_i \quad (1)$$

$$r = m(1 + d) \quad (2)$$

4.5.2 Google Trends Analysis Recommendation

In order to obtain external demand regarding each location, a Google Trends analysis component was incorporated. Google Trends has constantly been used in different areas to predict interest in certain subjects, being tourism one of the areas with a increasing number of papers published (Dinis et al., 2019). For the scope of this project, it is used to recommend property prices based on the rise or decrease of the Google Trends search value given the property location. That way, when a big event is announced, for example, and people start searching on google for details about it or for accommodations, general information about the city, tour tips and/or more, that demand is perceived by the algorithm and a rise on price is suggested. Additionally, when the event is over, the value of searches decreases, causing the algorithm to adjust the recommended price down to around the original normal price.

In order to obtain that variation, a request is made daily in the end of the day to SerpAPI, a Google scrapper³⁰, which then returns a list of the relative number of searches (0-100) made for each day since one month ago. Next, the percentage difference between today and yesterday is computed, then used on an logarithmic equation (3 for positive and 4 for negative differences) to define the corresponding price variation pv . The reason for using a logarithm is to prevent high variations of causing a direct, unrealistic, price variation. Also, it was defined a minimum of 10% increase or decrease to actually apply the formula, given small variations aren't useful to be reflected in this scenario. In those cases where variation is less then 10%, the price recommended

³⁰<https://serpapi.com/>

by the algorithm is the property own current price.

$$pv = \log_{10} \frac{x + 0.3}{4} + 1 \quad (3)$$

$$pv = (-1) \log_{10} \frac{x + 0.3}{4} - 1 \quad (4)$$

After calculating price variation, the recommended price is calculated as shown in 5:

$$r = p(1 + pv) \quad (5)$$

The reason for choosing 0.3 and 4 was to obtain a curve that results on reasonable values and has 0.10 as root, while the translation of plus/minus one was done to make the function positive/negative. Figure 14 shows equations 3 and 4 as functions. Both functions are limited to a maximum of one in absolute value because variations higher than that are forced back to one/minus one, since in cases of, for example, 99% variation, the function would output 2.39, which wouldn't be appropriate for price fluctuation.

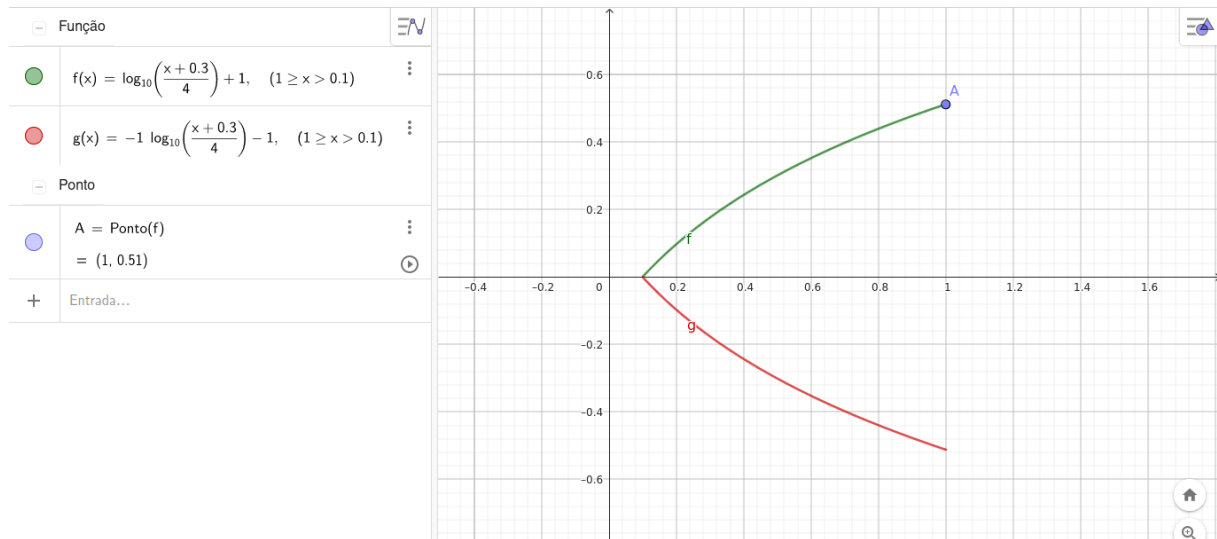


Figure 14: Logarithmic functions used for price recommendation Made with Geogebra <https://www.geogebra.org/classic?lang=pt-PT>

Moreover, in order to avoid possible pitfalls, two different checks were implemented. The first is a "bad trend" check, done before sending a request for the search values. This checker sends a request to the API to retrieve the related topics to the property location and verifies if

any word that would indicate a negative event, such as "disaster", "flood", "war" or "tragedy", is present in the list returned. If that's the case, the trend percentage difference is not applied, and the price recommended will be the property own actual price. This is done in order to not recommend a price increase in cases where tourism wouldn't be actually benefited.

As for the second check, it is done after the recommended price is calculated, and it verifies if it is lower than the Property Features ML Model component recommendation. In that case, the price recommended will be also the property own current price, since the trends recommended price would be too low and unrealistic, and using it would have a substantial negative impact on the final recommended price.

4.5.3 Internal Price Variation Recommendation

This component is responsible for detecting price variations within the *PropertEase* properties for each location and reflect them on properties in the same location that didn't follow. The goal with this approach is to be able to discover a trend that wasn't perceived by the Google Trends analysis. In that sense, if a big event or any other factor causes property owners to collectively increase or decrease the prices, this component will reflect that variation on the recommended price.

The algorithm for this component is simple: for each location, the mean of the old prices (updated every time the price recommendation is requested) and of the current prices are calculated. Then, if the absolute value of the percentage difference between them is greater than or equal to 10%, this difference will be reflected on each property that had no changes from the old price to the current price.

4.5.4 Final Price Recommendation

After obtaining the recommended prices from each component, the final price recommendation is calculated as their weighted average. The weights, as shown in Figure 13, were defined as 30% for the Property Features ML Model component, 40% for the Google Trends Analysis component and 30% for the Internal Price Variation component. That way, each component can balance the result from the others while still leaving a margin for the Google Trends Analysis to have a higher impact, which is presumably the most accurate and representative one.

4.5.5 Price Recommendation Lifecycle

The price recommendation lifecycle can be visualized in Figure 15. It starts with the trigger of the scheduled function at Property Service, that runs daily at 10:30 p.m. (22:30) UTC. This function sends a message to the message queue requesting the price recommendation from the Analytics Service, with a list of all system properties. After receiving it, the Analytics Service main module calls the price recommendation functions from each component (in the diagram, "model recommendation" corresponds to the Property Features ML Model component, "internal price recommendation" corresponds to the Internal Price Variation component and "trends recommendation" corresponds to Google Trends Analysis component). Finally, the final price recommendation is calculated with a weighted average, as previously explained, and sent to the Property Service through the message queue.

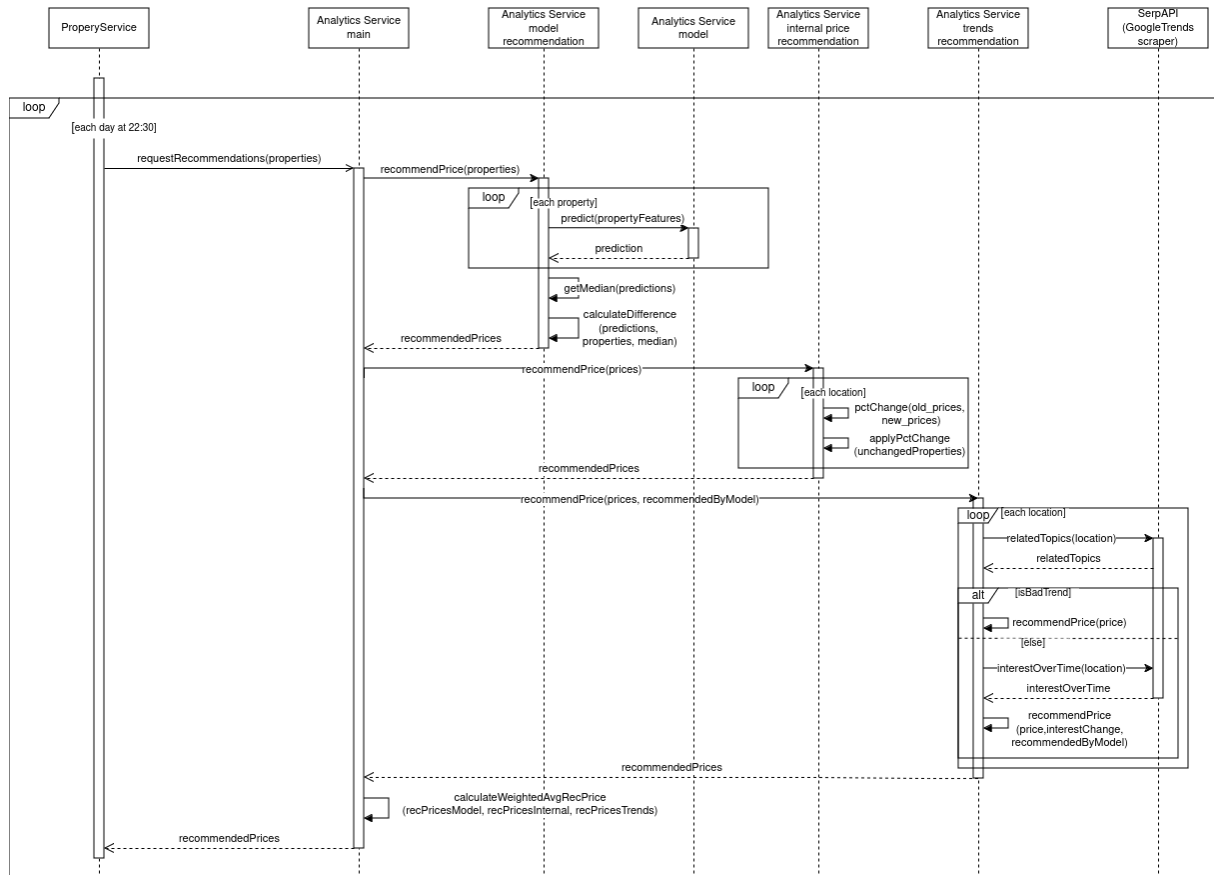


Figure 15: Price recommendation sequence diagram

5 Results

5.1 Price Recommendation

Since the usage of real property prices was not viable and simulated properties had to be used, it isn't possible to obtain general results from the price recommendation feature. The only aspect that can be measured is the model prediction, and to do that, metrics such as R2, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) were used. In Table 6, the results were obtained from training and testing with only 2021 Airbnb Lisbon data from the InsideAirbnb website (InsideAirbnb, 2024). The division for the train and test subsets was 0.75 and 0.25, respectively.

Table 6: Random Forest model results for Lisbon testing

RMSE	20.77
MAE	13.95
R2	69.35%

5.2 Use Cases Materialized

This section describes the implemented use cases and their materialization into tangible pages, as part of our user interface, represented by images with numbered tags. By mentioning 'Figure 10.1', the reader should be looking at the tag with number 1 in Figure 10. The title for each of the following sub-sections includes the name and identifier of the use case detailed in it. A list of these use cases, along with their identifier (ID) and description can be found in Section 3.4.2.

Import listings from one or more listing services - ID: 1.1

This is one of the core use cases of our system, and is a requirement to effectively use the rest of the system, as it imports the user's property listings from external listing services into *PropertEase*.

This use case can be achieved in the page represented in Figure 16. Users click on the **Connect** button, and a full import properties and reservations workflow will be triggered in the background.

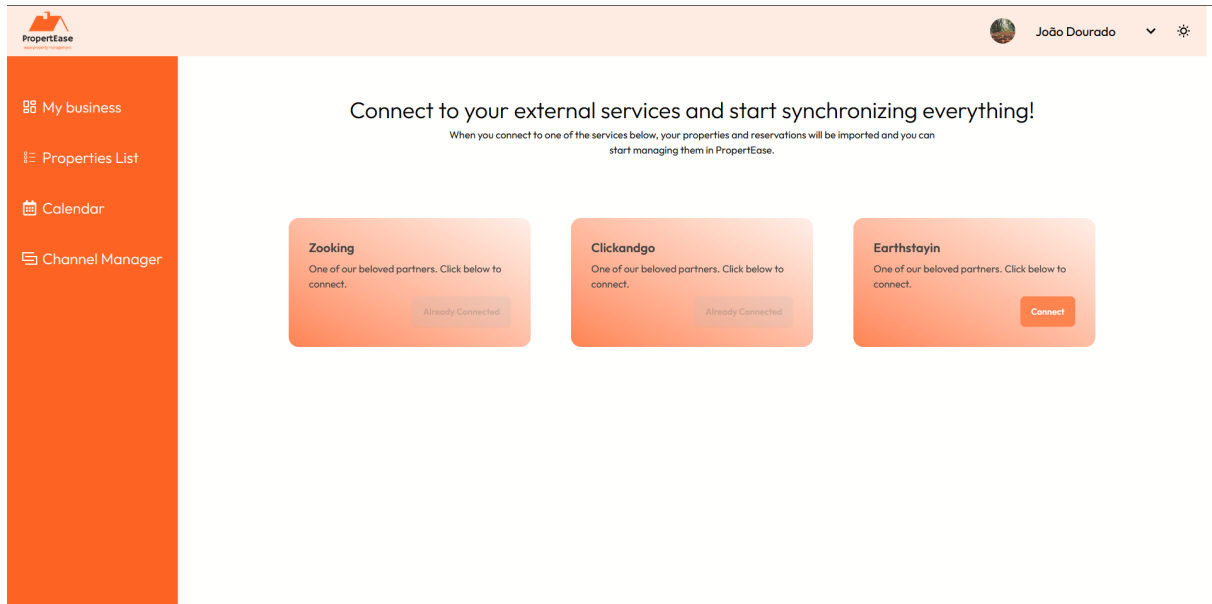
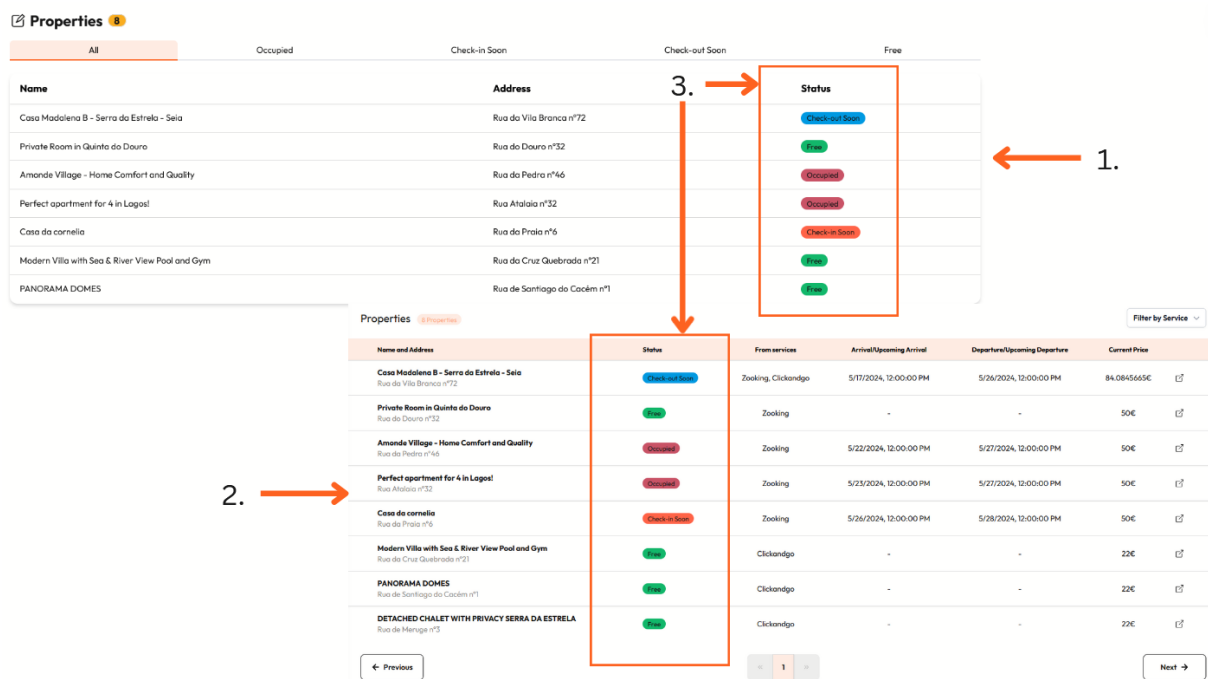


Figure 16: Channel Manager page where users can connect to external listing services and import their listings

View properties list - ID: 1.3 / View properties' status - ID: 1.10 / Filter properties by status - ID: 1.10.1

Both the **View properties list** and **View properties' status** use cases are materialized in two different locations in our system:

1. In the *Dashboard* page, which has a small table with the property list, with the goal of quickly check the status of any given property. This is represented in Figure 17.1 and 17.3.
2. The *Properties* page, whose difference compared to the *Dashboard* is to show more information about the property, such as its name, address, current price, and current status. This is represented in 17.2 and 17.3.



Properties 8

All Occupied Check-in Soon Check-out Soon Free

Name	Address	Status
Casa Madalena B - Serra do Estrela - Seia	Rua da Vila Branca nº72	Check-out Soon
Private Room in Quinto do Douro	Rua do Douro nº32	Free
Amonde Village - Home Comfort and Quality	Rua da Pedra nº46	Occupied
Perfect apartment for 4 in Lagos!	Rua Atalaia nº32	Occupied
Casa da cornelia	Rua do Praia nº6	Check-in Soon
Modern Villa with Sea & River View Pool and Gym	Rua de Cruz Quebrada nº21	Free
PANORAMA DOMES	Rua de Santiago da Cadeim nº1	Free

1. →

3. →

2. →

Filter by Service

Name and Address	Status	From services	Arrival/Upcoming Arrival	Departure/Upcoming Departure	Current Price
Casa Madalena B - Serra do Estrela - Seia Rua da Vila Branca nº72	Check-out Soon	Zooking, Clickandgo	5/17/2024, 12:00:00 PM	5/26/2024, 12:00:00 PM	84.0845665€
Private Room in Quinto do Douro Rua do Douro nº32	Free	Zooking	-	-	50€
Amonde Village - Home Comfort and Quality Rua da Pedra nº46	Occupied	Zooking	5/22/2024, 12:00:00 PM	5/27/2024, 12:00:00 PM	50€
Perfect apartment for 4 in Lagos! Rua Atalaia nº32	Occupied	Zooking	5/23/2024, 12:00:00 PM	5/27/2024, 12:00:00 PM	50€
Casa da cornelia Rua do Praia nº6	Check-in Soon	Zooking	5/26/2024, 12:00:00 PM	5/28/2024, 12:00:00 PM	50€
Modern Villa with Sea & River View Pool and Gym Rua de Cruz Quebrada nº21	Free	Clickandgo	-	-	22€
PANORAMA DOMES Rua de Santiago da Cadeim nº1	Free	Clickandgo	-	-	22€
DETACHED CHALET WITH PRIVACY SERRA DA ESTRELA Rua de Menige nº3	Free	Clickandgo	-	-	22€

← Previous 1 Next →

Figure 17: Dashboard and Properties page, showing a list of properties, their status

View events calendar - ID: 1.2 / Filter events by service and/or property - ID: 1.2.1

The **View synchronized properties calendar** and **Filter calendar view** use cases occur in our system as follows:

1. In the *Dashboard* page which includes a condensed timeline view to quickly visualize all events within a weekly time frame, as per Figure 18.1.
2. In the dedicated *Calendar* page which provides a detailed timeline and filtering options, allowing users to apply various filters to customize their view according to specific criteria (platform and/or property). This is represented in Figure 18.2 and 18.3.

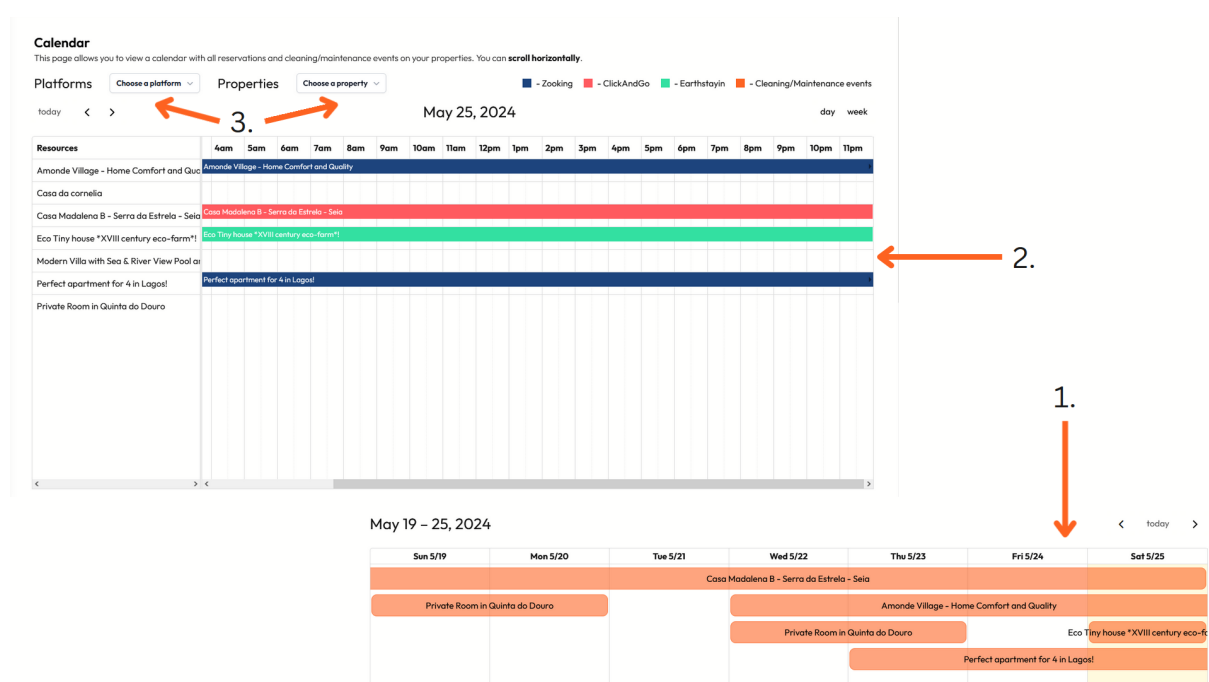


Figure 18: *Calendar* page, showing a timeline view of events allowing them to be filtered by platform or property, and *Dashboard*'s weekly calendar view

Set property price - ID: 1.6 / Choose revenue method based on commission - ID: 1.6.1 / View recommended price - ID: 1.6.2 / Set automatic price update based on recommended price - ID: 1.7

Figure 19 shows 4 use cases about price related operations:

- In 19.1, a button can be seen that, when clicked, opens a modal to update the property price, materializing the use case *Set property price*.
- In 19.2, there's a small text box highlighted by a tint of green, showing the price recommendation for this property, calculated by our dynamic pricing algorithm, materializing the use case *View recommended price*.
- In 19.3, there's a checkbox that dictates whether the price update will consider commission or not. If it is checked, the price update will be as such that the user receives whatever value he set - for example, if user sets 90€ and checks the checkbox, price updates might propagate as 92.70€ for an external service with 3% commission, so that the user will receive 90€ after commission is taken. This materializes the use case *Choose revenue method based on commission*.
- In 19.4, there's a checkbox with a label and a small tool-tip related to updating the price automatically according to the recommended price for this property, as given by our dynamic pricing algorithm. This materializes the use case *Set automatic price update based on recommended price*.

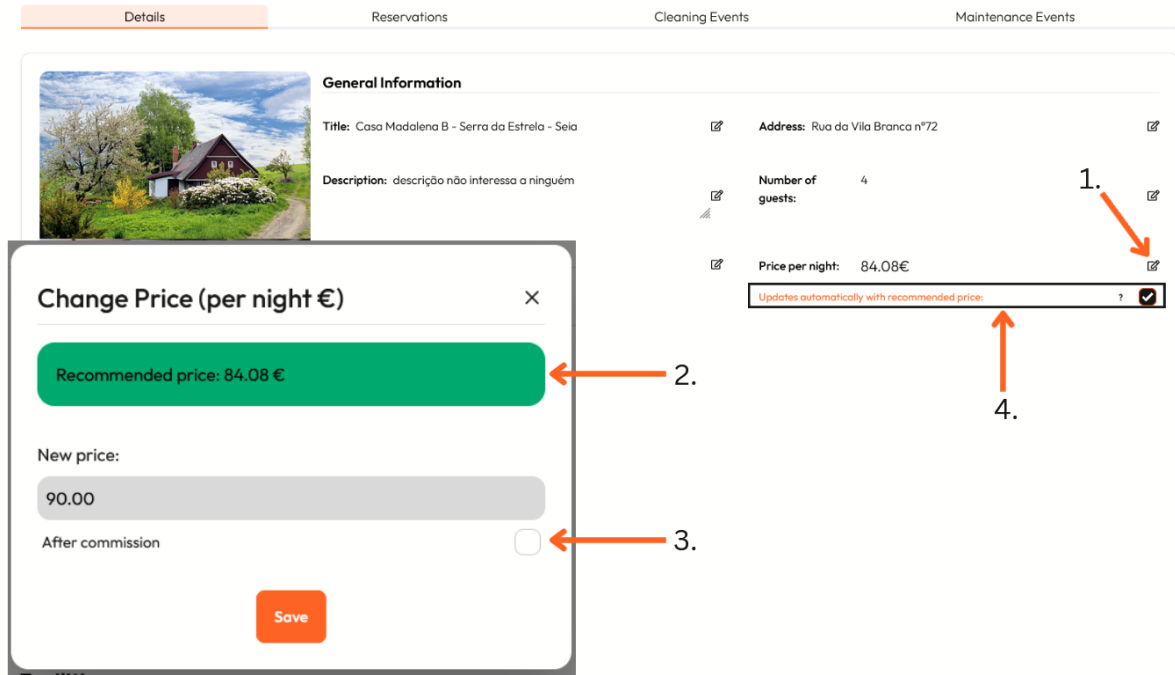


Figure 19: View of *Property Details* page, including the *Edit price* modal

Edit property details - ID: 1.5

Figure 20 shows a view of the *Property Details*, with blue squares around operations that edit multiple fields of the property, as defined by the *Edit property details* use case. Whenever these buttons are clicked, they open a modal to update the field they refer to.

For example, when the button referenced by Figure 20.1 is clicked, the modal represented in Figure 21 is opened, allowing users to add a new bathroom including what fixtures that bathroom has, editing the *Bathrooms* field in the process.

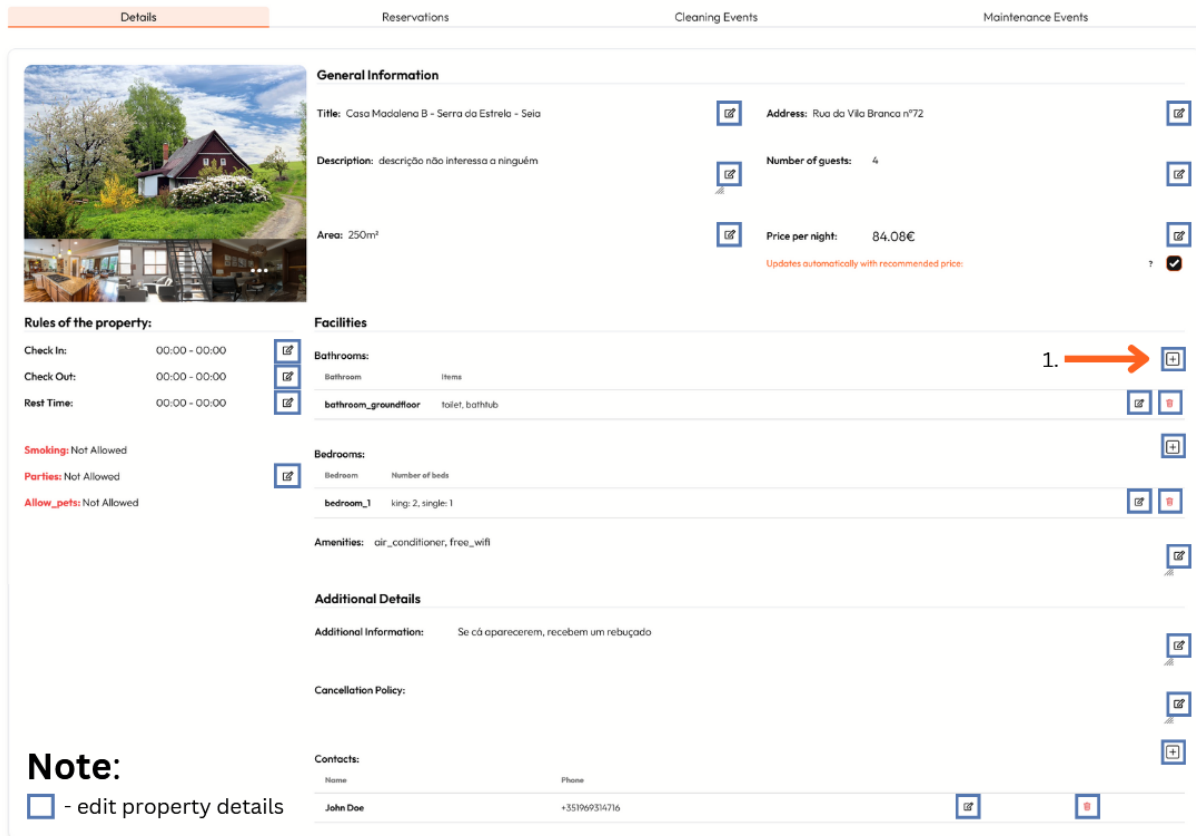


Figure 20: View of *Property Details* page, showing property details

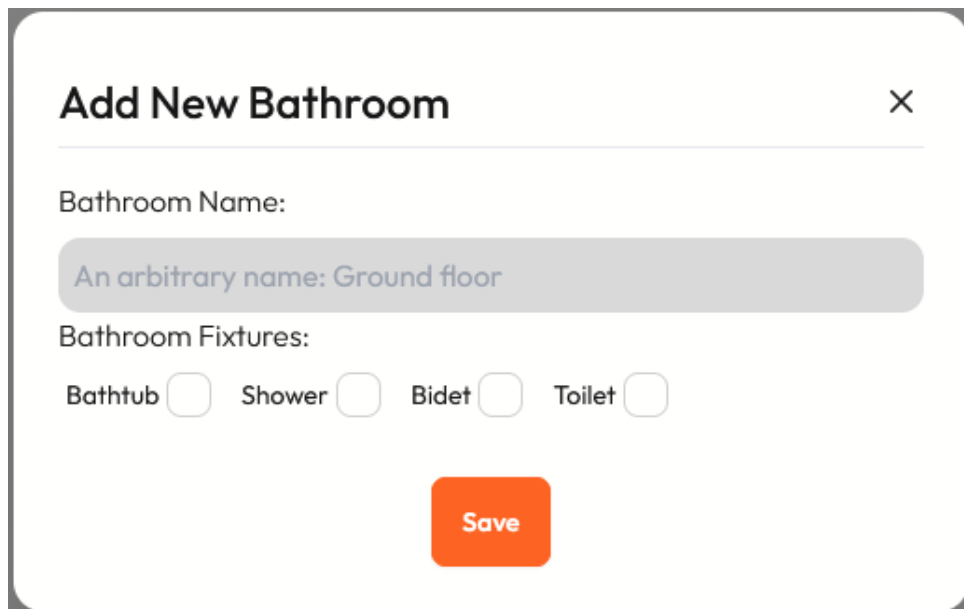
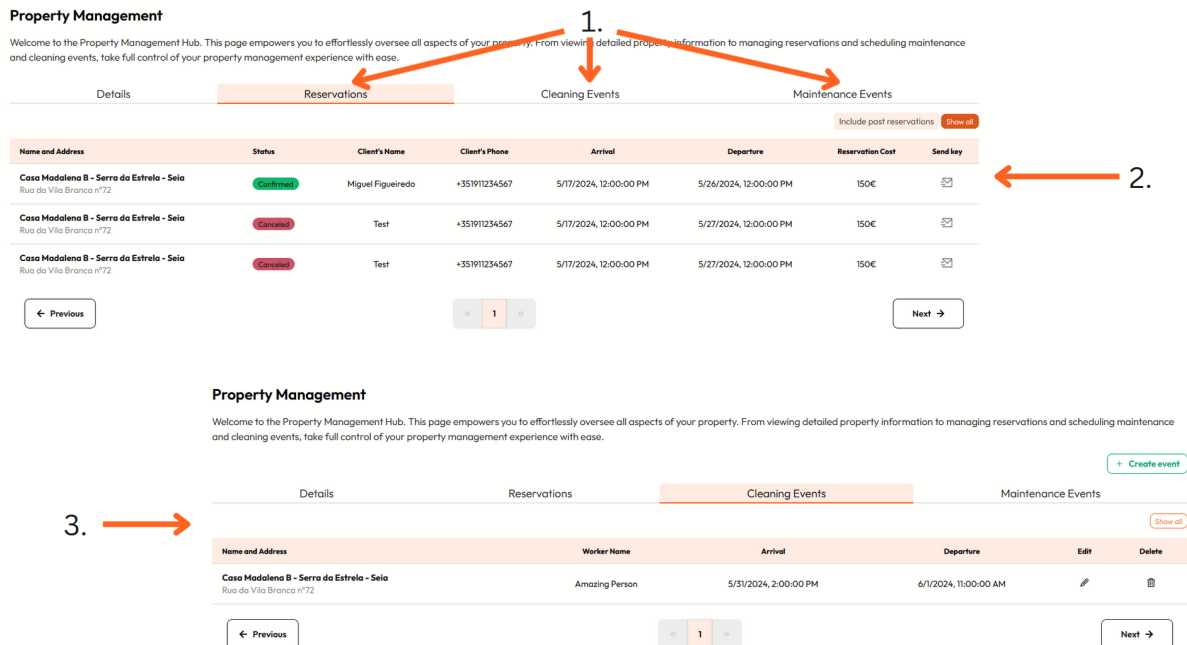


Figure 21: Modal for adding a new bathroom to the selected property, editing its *Bathrooms* field

View property events (reservations, cleaning/maintenance events) - ID: 1.8

In the *Property Details* page it is provided a detailed view of property-specific events such as reservations, cleaning and maintenance activities within their own tabbed sections (see Figure 22.1. Reservations are depicted in Figure 22.2, while cleaning events are shown in Figure 22.3.



Property Management
 Welcome to the Property Management Hub. This page empowers you to effortlessly oversee all aspects of your property. From viewing detailed property information to managing reservations and scheduling maintenance and cleaning events, take full control of your property management experience with ease.

1. →

2. →

3. →

Name and Address	Status	Client's Name	Client's Phone	Arrival	Departure	Reservation Cost	Send key
Casa Madalena B - Serra da Estrela - Seia Rua da Vila Branca nº72	Confirmed	Miguel Figueiredo	+351911234567	5/17/2024, 12:00:00 PM	5/26/2024, 12:00:00 PM	150€	✉
Casa Madalena B - Serra da Estrela - Seia Rua da Vila Branca nº72	Cancelled	Test	+351911234567	5/17/2024, 12:00:00 PM	5/27/2024, 12:00:00 PM	150€	✉
Casa Madalena B - Serra da Estrela - Seia Rua da Vila Branca nº72	Cancelled	Test	+351911234567	5/17/2024, 12:00:00 PM	5/27/2024, 12:00:00 PM	150€	✉

Name and Address	Worker Name	Arrival	Departure	Edit	Delete
Casa Madalena B - Serra da Estrela - Seia Rua da Vila Branca nº72	Amazing Person	5/31/2024, 2:00:00 PM	6/1/2024, 11:00:00 AM	✎	🗑️

Figure 22: Reservation and cleaning tables from a *Property Details* page

Send property door key to client through e-mail - ID: 1.9

Figure 23 shows the reservation table, and Figure 23.1 is highlighting a button, that when clicked, opens a modal with a form where users can input a *key code* to be sent to the reservation client e-mail, as a means to open the door to this property. Clicking 23.2 triggers the request and subsequent e-mail. A button to randomize key codes was also included - it generates a 6 characters long numeric string, but is completely optional, as the user can input whatever they want into the input field. All this materializes the "Send property door key to client through e-mail" use case.

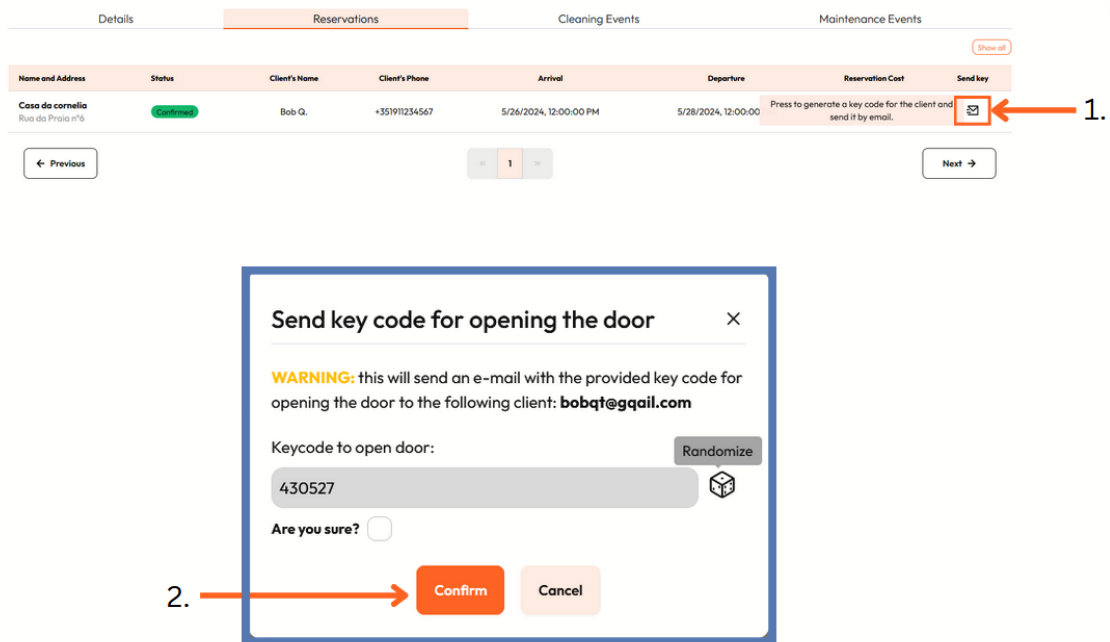


Figure 23: Reservation table highlighting the *Send key code to client* button, and its corresponding modal

Manage cleaning/maintenance events - ID: 1.4

Figure 24 shows both the *Cleaning* and *Maintenance* events table, including events related to both tables. Clicking the green button highlighted at 24.1 will open the modal shown in Figure 25, which allows users to create a brand new event of the requested type, based on the selected tab (Cleaning Events/Maintenance Events). Creating either events triggers the workflow of closing off the relevant time interval in the external services this property is found in, in order to synchronize calendars. Clicking the buttons at 25.2 and 25.3 also allows editing and deleting, respectively, these events.

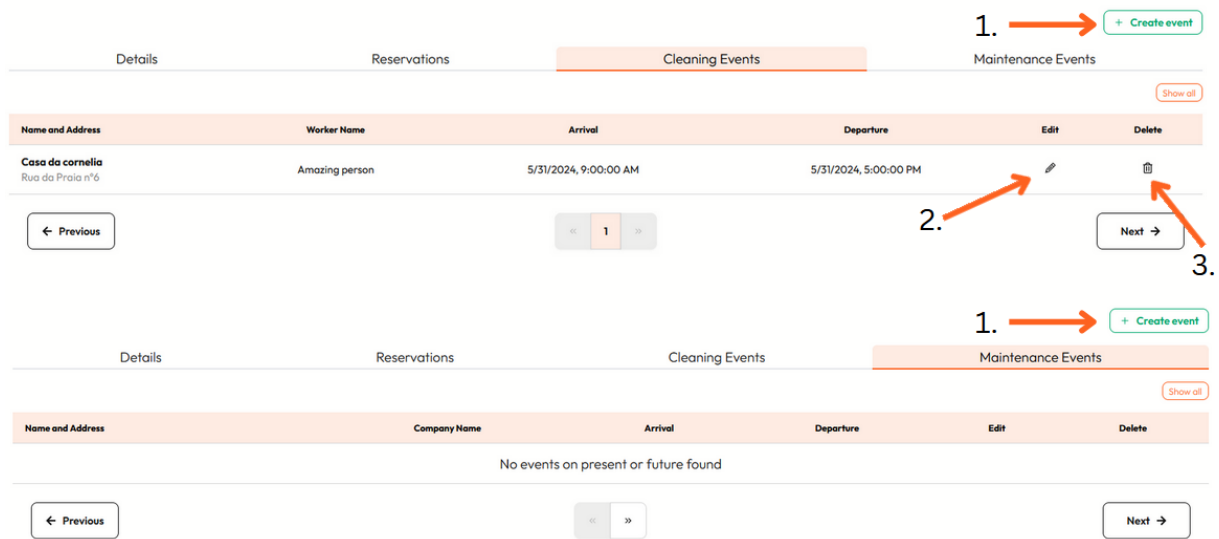


Figure 24: Cleaning/Maintenance events tables, showing their kind of events respectively, allowing the user to also create new ones and edit or delete existing ones

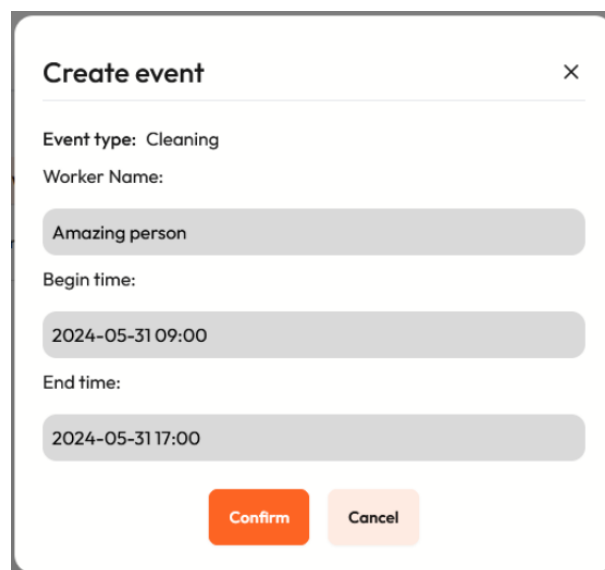


Figure 25: Modal for creating new cleaning/maintenance events

5.3 Quality Assurance

For assuring the quality of *PropertEase*, we conducted tests on two fronts: usability tests for our user interface, and contract tests on our website wrappers. Usability tests were conducted late on the project, when most use cases were developed and the system was at an MVP level. The contract tests are supposed to serve as a safety net, used to detect API schema changes in our supported external services, which might break wrappers if changed.

5.3.1 Contract Tests

The one place coupled to the supported external services API schemas in our architecture is the website wrappers, since we directly interact with them whenever any changes in *PropertEase* need to be propagated, or when we need to detect new changes from external services and register them in *PropertEase*.

In order to guarantee that breaking schema changes are detected and acted upon quickly, we developed contract tests with *pytest*³¹ for each one of the supported external APIs. These tests fetch data from the external APIs using the *requests*³² library, such as properties and reservations, and check the returned data schema against an expected schema. Using *pydantic*³³, a python library for data validation, our tests validate against a pre-defined schema (which is always the most recent API return schema), and expect that this schema is exactly met: if any extra fields are returned, tests fail, as this means that new data might have been added to the API schemas; if any field changed name, tests fail; if any field was removed, tests fail.

A source code example can be found in Appendix D, which displays contract tests developed for the (simulated) *Zooking* external listing service API.

5.4 Usability Testing

After having a stable MVP with the core use-cases implemented, it was time to conduct usability tests, to gather some feedback regarding user experience when using our system. We had the opportunity to carry out these tests with fellow University of Aveiro students, in the Human-Computer Interaction course.

³¹<https://docs.pytest.org/en/8.2.x/>

³²<https://docs.python-requests.org/en/latest/index.html>

³³<https://docs.pydantic.dev/latest/>

5.4.1 Sample

With the help of Professors Dr. Paulo Dias and Dr. Samuel Silva, teaching Human-Computer Interaction, we conducted 16 usability tests. 14 of the participants were students from ages 19-22 and two were teachers. 4 students had "some" experience with property management systems, while all other participants did not have any prior experience with this kind of system. Forms and raw data obtained from results can be found in Appendix B.

5.4.2 Method

Participants were asked to use our system to complete 6 different tasks, which were based on common use cases of our system. During the test execution, two forms were filled: an observers script form, with relevant information obtained while watching participants complete the different tasks; and a task form, where users answered the problems presented in the tasks (for example, how many properties were imported from Zooking when connecting to that external service) and gave them a difficulty rating from 1 to 5, 1 being very difficult and 5 being very easy. The task form filled by users during task execution and their responses can be found in Appendix B.

We started each test by giving a small introduction of our system, to contextualize the different tasks and make sure the participant knew what the system concepts were and an introduction in what "centralized property management" meant.

The first task was intended to see if users knew what to do from a completely fresh system point of view, starting by the "import properties from external services" use case. They were requested to import their properties from the Zooking external service and answer how many were imported, which had them go to the Channel Manager page, click on the connect now button related to Zooking, and then check the Properties page which had a list with the number of properties.

The second task had users connect to another external service, ClickAndGo, and check if those properties had any reservations on May 14th (the day the tests were conducted). This task had users once again go to the Channel Manager page, and connect to ClickAndGo. After that, they would either check the dashboard calendar for existing reservations, or the dedicated Calendar page to see if any reservations were ongoing at that moment.

The third task had users check which amenities a specific property had. The goal was to check whether finding the property details, such as price, name and amenities, was intuitive.

The fourth task had the objective of finding the price of a specific property, and have users check a checkbox to make the property automatically update price with the recommended price. We wanted to assess whether activating automatic price updates based on our machine learning algorithm was intuitive, as it was one of our core use cases.

The fifth task tested the "Create cleaning/maintenance events for a property during a specific time frame", and we wanted to see whether doing this seemingly simple task was intuitive or not.

The sixth and final task had users send a key code to open the door to a specific property to a client, that had made a reservation.

After completing all tasks and finishing the usability test, participants were prompted to fill a Post Task Questionnaire in a form, where users gave subjective assessments of usability based on the System Usability Scale. The Post Task Questionnaire can also be found in Appendix B.

5.4.3 Results

From the obtained data, we not only got direct feedback from participants regarding user experience related to some of the tasks, but we also had a measure of the system usability, using the System Usability Scale. Raw data can be found in Appendix B.

The most common feedback from our participants was the following:

- The list of properties in the Properties page should show the service(s) from which a property was imported
- The reservations table in the Property Details page requires scrolling, and its position is not obvious; many people mistakenly scrolled down and found it.
- In the reservation table, hide reservations from the past by default and only show present/future reservations, while keeping the possibility of showing past reservations
- Property details page has too much information, and the details of the actual property, which rarely change, were the first thing users saw, instead of current and future reservations
- Having no visual queue or feedback in the "Updates automatically with best price" checkbox in the Property Details page, makes it hard to know if when you check the checkbox anything changed

- Property details page should have a calendar for that given property, so that users can have a visual representation of upcoming/on-going reservations, instead of having to read dates in the reservation table

The chart in Figure 26 shows the average score given to each question from participants when answering the Post Task Questionnaire, which followed the System Usability Scale.

The obtained SUS score was of 78.75.

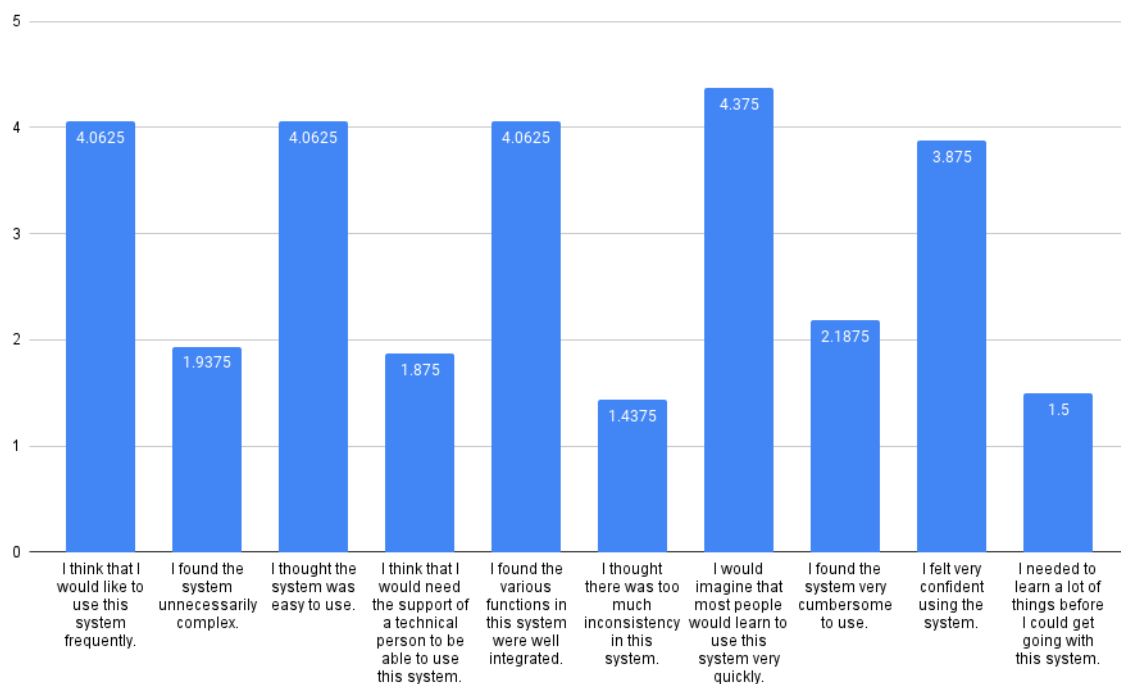


Figure 26: Average per System Usability Scale question

From the given feedback, we implemented some changes to the user interface to improve user experience. The following section shows changes that were implemented, and an image with the before and after of the changes.

Figure 27 shows changes made in the property table in Properties page, related to the feedback given about including the service(s) from which a property was imported in this table as a column. In 27.1, a column shows the services from different services from which the property was imported, and to which changes will be propagated. We also decided to include 27.2, which allows users to filter that table by the available External Services, and only show properties imported from a specific service.

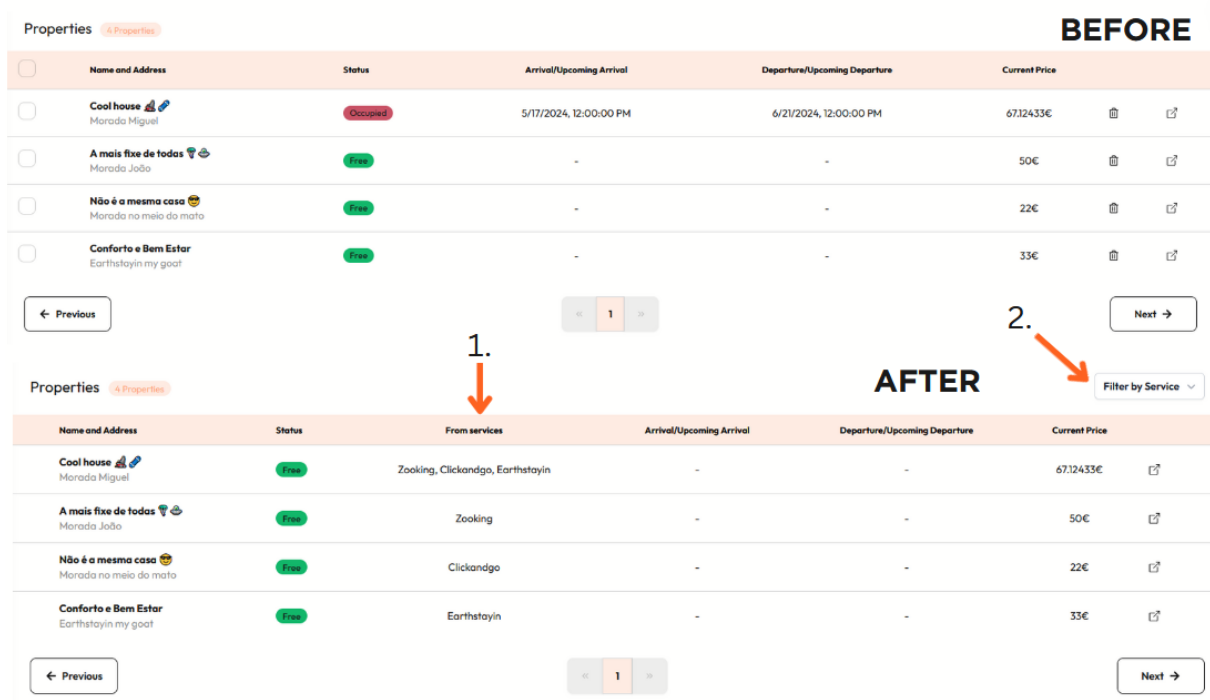


Figure 27: Changes in the property table in Properties page

Figure 28 shows changes made in the Property Details page, related to the feedback about how difficult it was to find the reservations table and the fact that clicking the checkbox to update the price automatically based on the recommendation from our dynamic pricing algorithm did not have any visual feedback.

In 28.1, a small description of what can be done in this page was added in the top of the page. In 28.2 and 28.4, we added a tool-tip to describe the function of the checkbox, and an alert as feedback when changes are made to it, respectively. In 28.3, we changed the previous layout, which simply had the reservations and cleaning/maintenance events tables under property details, into this layout with tabs.

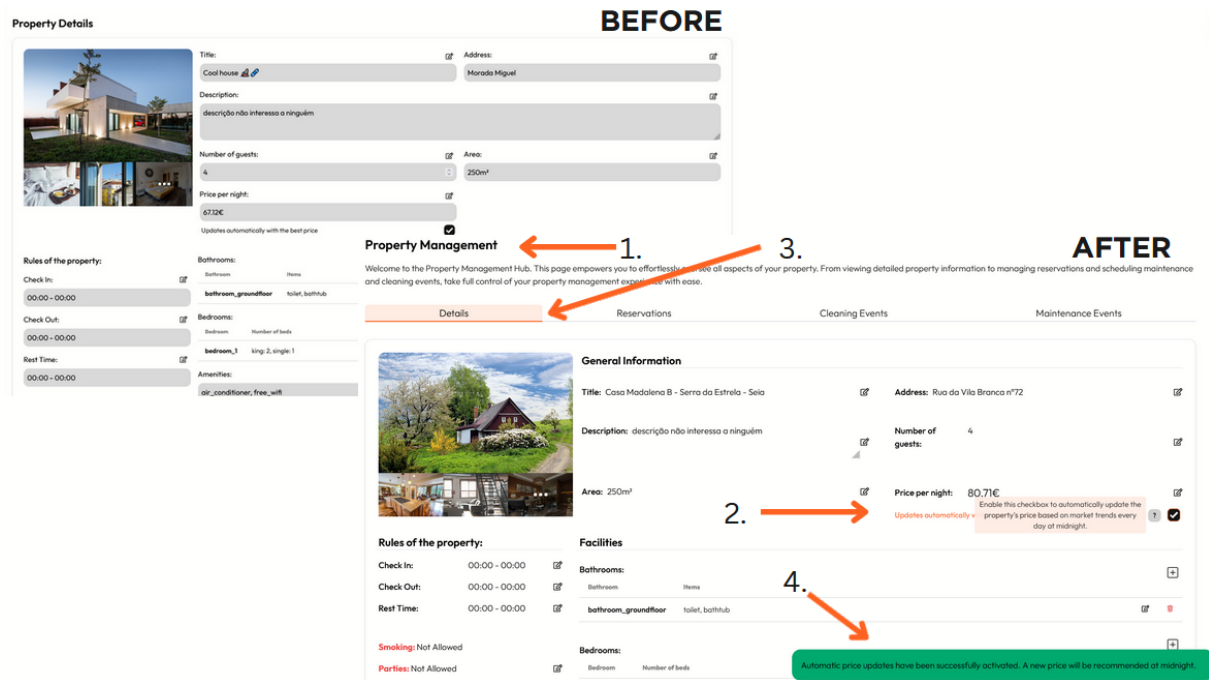









Figure 28: Changes in the Property Details page

Figure 29 shows changes applied to the reservation table in the Property Details page, related to feedback about how seeing old reservations made the table have a lot of useless data, and finding the useful data (current and future reservations) was difficult.

In 29.1 we added a button with the label "Show all"/"Show less", that when clicked, shows all reservations, or only current/future ones. It also has a tool-tip so users can more easily understand its behavior.

+ Create event

Reservation Events			Cleaning Events		Maintenance Events		
<input type="checkbox"/>	Name and Address	Status	Client's Name	Client's Phone	Arrival	Departure	Reservation Cost
<input type="checkbox"/>	Cool house  Morada Miguel	Confirmed	Test	+35191234567	5/1/2024, 12:00:00 PM	5/4/2024, 12:00:00 PM	150€
<input type="checkbox"/>	Cool house  Morada Miguel	Confirmed	Test	+35191234567	5/4/2024, 4:00:00 PM	5/5/2024, 6:00:00 PM	150€
<input type="checkbox"/>	A mais fixe de todas  Morada João	Confirmed	Olá	+35191234567	4/29/2024, 12:00:00 PM	4/30/2024, 12:00:00 PM	150€
<input type="checkbox"/>	A mais fixe de todas  Morada João	Confirmed	Cahaha	+35191234567	5/2/2024, 12:00:00 PM	5/6/2024, 12:00:00 PM	150€
<input type="checkbox"/>	Cool house  Morada Miguel	Cancelled	Test	+35191234567	5/1/2024, 12:00:00 PM	5/4/2024, 12:00:00 PM	150€
<input type="checkbox"/>	Cool house  Morada Miguel	Cancelled	Test	+35191234567	5/4/2024, 4:00:00 PM	5/5/2024, 6:00:00 PM	150€
<input type="checkbox"/>	Não é a mesma casa  Morada no meio do mar	Confirmed	bebe	+35191234567	4/20/2024, 4:00:00 PM	4/26/2024, 6:00:00 PM	150€

AFTER

Welcome to the Property Management Hub. This page empowers you to effortlessly oversee all aspects of your property. From viewing detailed property information to managing reservations and scheduling maintenance and cleaning events, take full control of your property management experience with ease.

+ Create event











Details		Reservations	Cleaning Events		Maintenance Events			
		Only present and future reservations Show less						
<input type="checkbox"/>	Name and Address	Status	Client's Name	Client's Phone	Arrival	Departure	Reservation Cost	Send key
<input type="checkbox"/>	Cool house  Morada Miguel	Confirmed	Test	+35191234567	5/1/2024, 12:00:00 PM	5/4/2024, 12:00:00 PM	150€	
<input type="checkbox"/>	Cool house  Morada Miguel	Confirmed	Test	+35191234567	5/4/2024, 4:00:00 PM	5/5/2024, 6:00:00 PM	150€	
<input type="checkbox"/>	Cool house  Morada Miguel	Cancelled	Test	+35191234567	5/1/2024, 12:00:00 PM	5/4/2024, 12:00:00 PM	150€	
<input type="checkbox"/>	Cool house  Morada Miguel	Cancelled	Test	+35191234567	5/4/2024, 4:00:00 PM	5/5/2024, 6:00:00 PM	150€	
<input type="checkbox"/>	Cool house  Morada Miguel	Confirmed	Figueiredo	+35191234567	3/20/2033, 12:00:00 PM	3/21/2033, 12:00:00 PM	150€	

Figure 29: Changes in the reservations table in the Property Details page

Figure 30 shows changes in the Properties page when the user still has no imported properties. During usability tests, we noticed some users started out by exploring the system, and when stumbling upon this page with an empty table, many were kind of confused.

For that reason, in 30.1, we added a button that redirects users to the "Channel Manager" page (previously "Integrations" page), where users can connect to external services and have their properties imported.

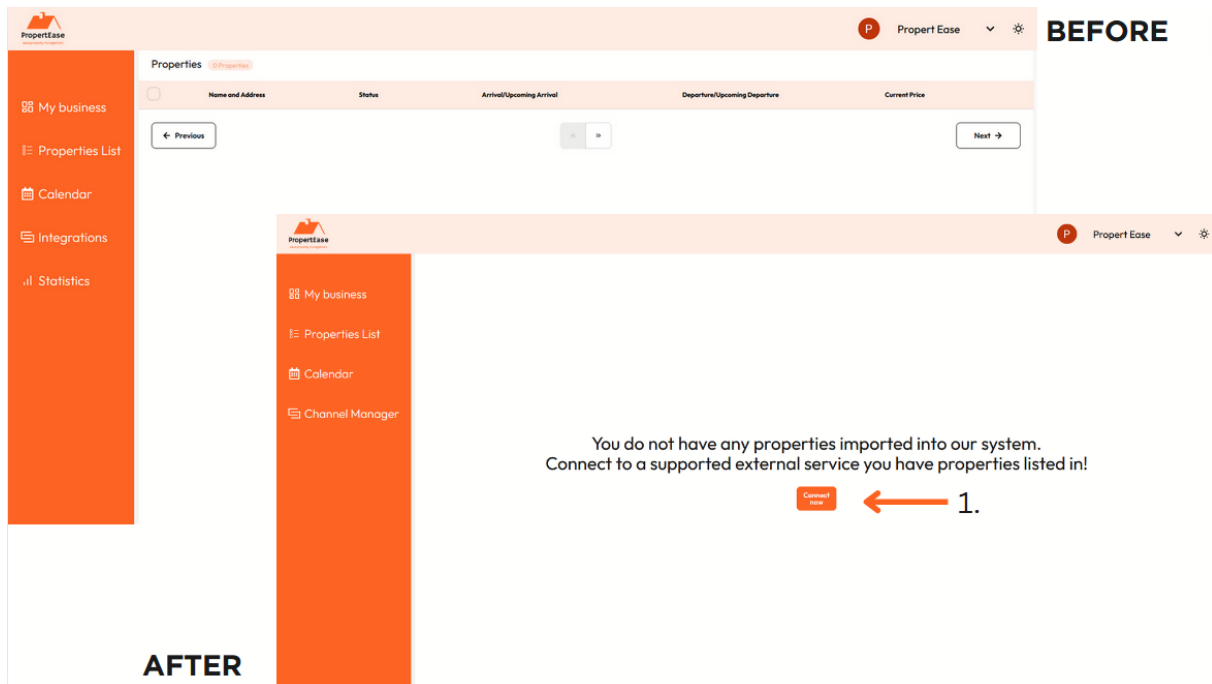


Figure 30: Properties page changes

Figure 31 shows changes in the "Create Event" modal, used as a form for the user to input data regarding a Cleaning/Maintenance event they might be trying to create.

As requested by user feedback, when the user is in the "Cleaning Events" tab, clicking the "Create Event" button should open this modal to create a cleaning event, instead of having the user pick the type of event they want to create an event for.

In 31.1, the event type is automatically selected to the current selected tab. If the user is in the "Maintenance Events" tab, the modal will automatically select the "Maintenance" as the event type being created. The same thing applies to cleaning events.

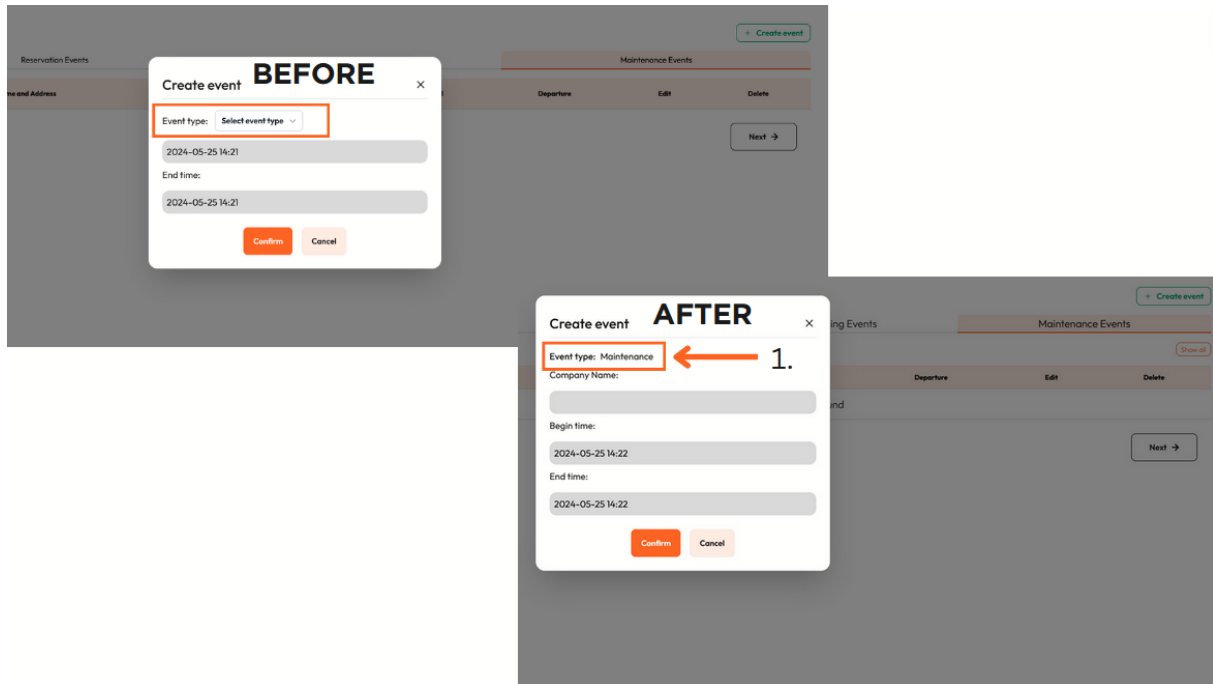


Figure 31: Create event modal changes in the Property Details page

Figure 32 shows the changes applied to the left-side drawer in our user interface. Some users pointed out that *Integrations* was not intuitive, and a different name for this tab would be better, related to management. A user pointed out *Channel Manager* as a suggestion, which is typically the name used by existing property management systems to refer to this tab. In 32.1 changes were applied when it comes to naming of the *Integrations* tab.

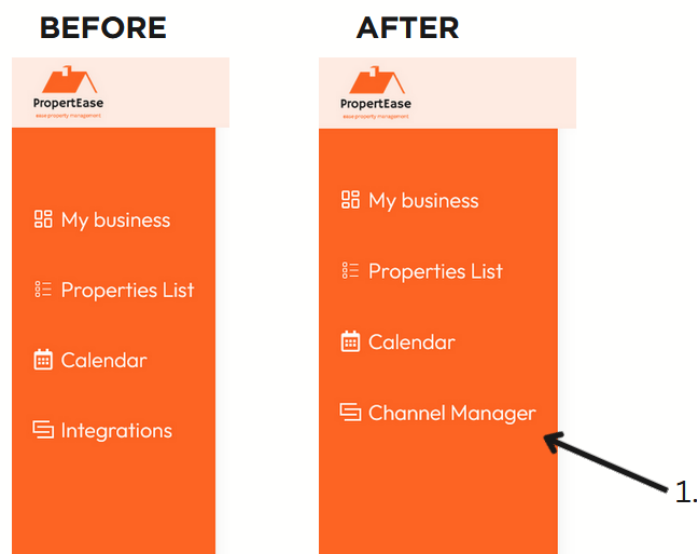


Figure 32: Changes in the left-side drawer, regarding naming of the Integrations tab

5.5 Data Correlations

The use of the Elasticsearch-Logstash-Kibana (ELK) stack enabled detailed data analysis within *PropertEase*, facilitating effective correlation establishment. Figure 33 displays the Kibana dashboard, however, the property data used to create it was simulated. Therefore, although the recommended prices received are based on algorithm-suggested values, it's important to note these simulated data may not fully reflect real market values, thereby impacting the algorithm's price accuracy. Nevertheless, this demonstrates that *PropertEase* is well-prepared to perform such analyses with ELK once deployed in a production environment.

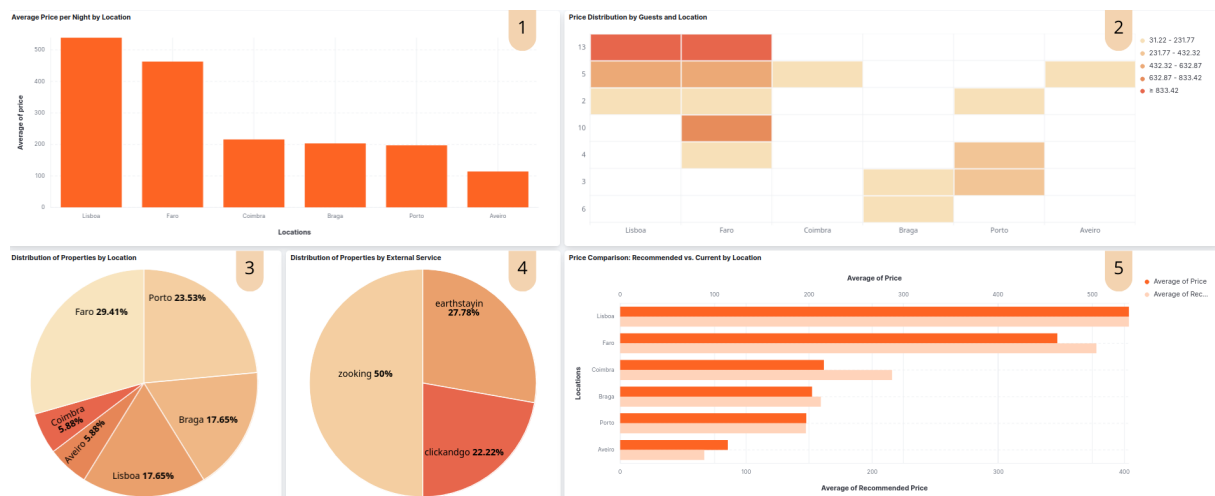


Figure 33: Kibana Dashboard (1. - Average Price per Night by Location; 2. - Price Distribution by Guests and Location; 3. - Distribution of Properties by Location; 4. - Distribution of Properties by External Service; 5. - Price Comparison: Recommended vs. Current by Location)

The bar chart showing the average price per night by location, displayed in Figure 34, allows for the analysis of current price trends for properties on *PropertEase* based on location. However, many variables can influence the price per night of a property. To complement this view, a heatmap, shown in Figure 35, was also created to show the distribution of average prices by location, taking into account the number of guests each property accommodates. These two graphs provide a comprehensive view of how prices are distributed and which locations have higher prices.

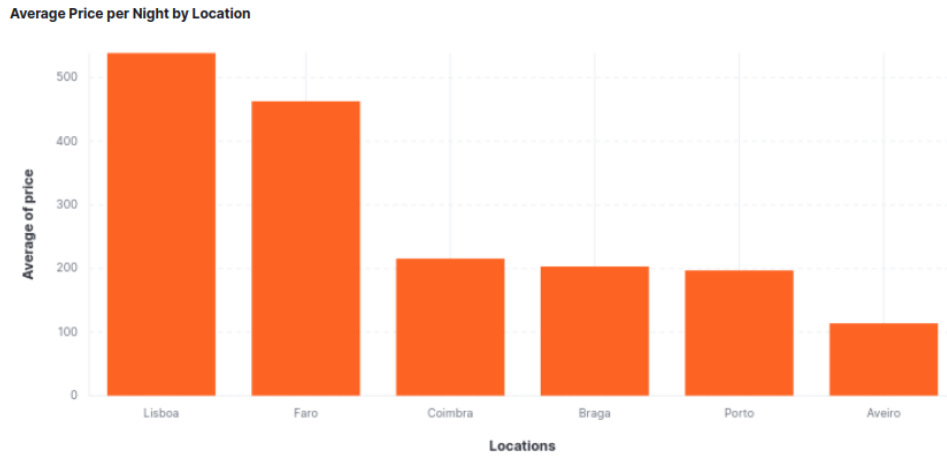


Figure 34: Bar Chart with Average Price per Night by Location

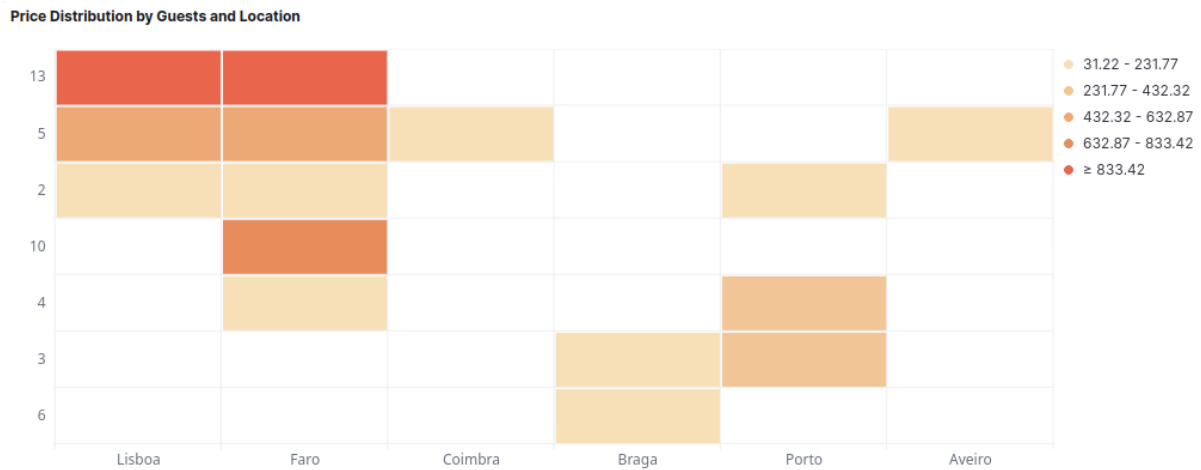


Figure 35: Heat Map with Price Distribution by Guests and Location

The pie charts presented in Figure 36 show the percentage distribution of properties by location (left chart) and by the booking services from which they are imported (right chart). These charts are essential for understanding which regions *PropertEase* is most utilized in and which booking services are most popular among *PropertEase* users. Such information is valuable for guiding strategic marketing decisions.

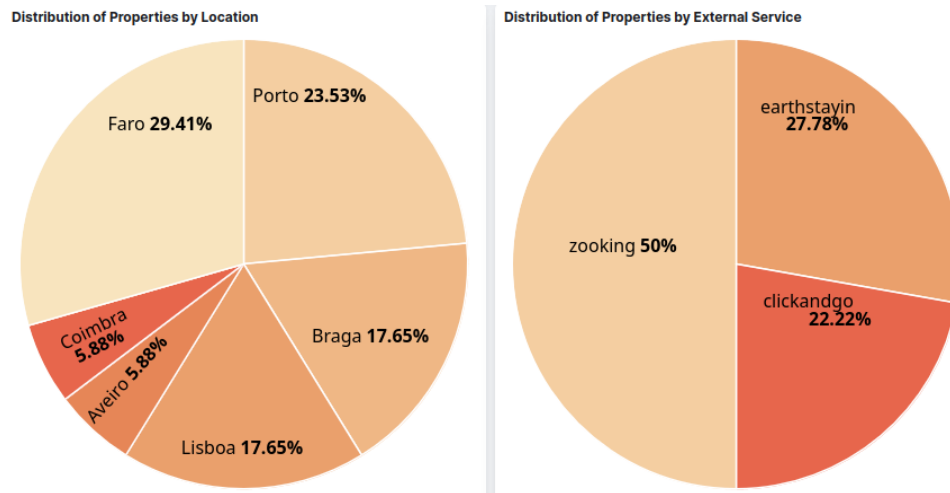


Figure 36: Pie Charts with Distribution of Properties by Location and External Services

Finally, in Figure 37, a horizontal bar chart is presented comparing the average nightly price that properties charge guests with the average price recommended by the implemented machine learning algorithm, by location. This chart simplifies the analysis to determine whether *PropertEase* users generally opt for the prices suggested by the algorithm or prefer to maintain their own prices. Additionally, it helps conclude whether users tend to set prices above or below the algorithm's recommendations.

In this way, conclusions can be drawn regarding users' perception of the price recommendation algorithm and whether adjustments are needed to improve the accuracy of the recommendations.



Figure 37: Bar Graph with Price Comparison

6 Discussion

6.1 Usability Tests

The usability tests done successfully gave an idea of how usable the system was, with much feedback to improve user experience based on the *System Usability Scale*. (Brooke (1995))

The obtained *System Usability Scale (SUS)* score, of 78.75 (as mentioned in the Results section) is above the average 68 (Sauro and Lewis (2016)), which meant most of the system was usable, but this is highly dependent on the tasks presented to users, which did not fully cover some more specific use-cases of the system, such as:

- When updating price of a property, choose revenue method based on commission
- View recommended price
- Filter calendar view by a specific property or platform
- View property status (e.g. free, occupied, ...) and filter properties by status

That effectively means some use cases were left untested, which leaves uncertainty about the quality of user experience for those. Still, most of the use-cases the system supports were tested, including core use-cases:

- Import listings from one or more listing services
- View synchronized properties calendar
- View properties list
- Manage cleaning/maintenance events
- Set automatic price update based on recommended price
- View property events (reservations, cleaning/maintenance events)
- Send property door key to client through e-mail

Feedback obtained was extremely important for shaping a greater user experience by implementing requested changes. In the future, a second round of tests, including new changes obtained from given feedback and extra tasks for untested use-cases, would be instrumental to making sure the system provides a great user experience for users that might not have any experience using property management systems like *PropertEase*.

6.2 Price Recommendation

Regarding the measurable results, the values obtained were acceptable. However, they were based only on testing with Lisbon properties, so they might vary when testing with other locations. In fact, more data from different cities would be needed to train and test the model so that more conclusive results could be obtained. This limitation becomes even more important due to the usage of latitude and longitude as features.

As for the unmeasured results, they impose some uncertainty about the quality of the price recommendation. For example, it isn't possible to ensure the logarithmic equation used is the optimal one, or that it really reflects trends on price in a optimal way. The same happens for the weights used in feature comparison (Figure 5) and the weights used on the weighted mean to calculate the final recommended price, since both were based on observation and our own assumptions.

Another limitation is the search queries used on the Google Trends analysis. Issues such as two cities with the same name or with name included in other's names such as "Porto" and "Porto Alegre", or the inability to ensure all "bad trends" will be caught by the algorithm check, may jeopardize the price recommendation by this component.

Given all these limitations, it's important to note however, that the weights used for the final recommendation guarantee a good balance between the components, which attenuates the individual problems each one has. Additionally, while some are tied to the nature of the process and harder to fix (e.g. Google Trends), some of those limitations such as the data shortage could easily be fixed the moment the data becomes available. One way to do that would be, once *PropertEase* has a good portion of properties from different locations, to use those properties to train the model. That way, the real quality of the model could be measured and improved if needed.

7 Conclusion

7.1 Summary

This report outlined the full process of developing *PropertEase*. It starts out by a detailed analysis of existing solutions, in order to assess what our system should support to be competitive and to have an edge in the property management system business. This detailed analysis is found in Chapter 2. From that analysis, we concluded that most competitors supported core functionalities of property management, such as managing property prices and reservations from a unified view, but none of them supported a built-in price recommendation algorithm that aims to maximize a property owner's revenue while minimizing manual tedious work.

This investigation led to Chapter 3, where we detailed the requirements we needed to fill, to provide a streamlined property management experience to property owners. We outlined functional requirements, non-functional requirements, system actors and a detailed list of use-cases *PropertEase* needs to fulfill to provide added value to our clients.

In Chapter 4, we thoroughly explained our system architecture, which follows the microservices architectural pattern, aimed at modularity and decoupling of the different components of *PropertEase*. That is exceptionally important considering the need to support different listing services over time, ideally without requiring architectural changes. We also comprehensively described our price recommendation algorithm, including the different sub-components that lead to the final price recommendation, such as the machine learning model and the property features used, the Google Trends analysis of property locations and the internal price variation.

Following the full architecture specification, the next logical step was implementation of *PropertEase* using the previously detailed technologies - FastAPI for the RESTful API our microservices provide, RabbitMQ as a message broker for asynchronous communication, ELK stack in the analytics service for data analysis using Kibana and React Typescript for our user interface. We also obtained and trained our model and implemented each component specified in the architecture. The results from this implementation are disclosed in Chapter 5 - it includes an analysis of our machine learning model validation process, the implemented use cases, some measures for quality assurance and a detailed analysis of the usability tests performed by users to outline possible problems with our user interface and improvements that were made based on the feedback given.

Chapter 6 includes a critical view of the results obtained - the effectiveness of our usability

tests and its results, and limitations our price recommendation algorithm has, based on the methodology and data used.

7.2 Main Results

PropertEase has successfully achieved its main goal of creating a centralized property management platform, having successfully implement the following features and benefits:

- **Importing of properties from different listing services;**
- **Synchronizing calendars from all connected listing services.** This means that when a booking is made in one of the listing services, *PropertEase* will forward that information to all the other connected services and update the calendars. The same applies when a property owner creates an event in *PropertEase*, such as cleaning or maintenance, which makes a property unavailable for some time.
- Implementation of a **dynamic pricing adjustment algorithm.** This is a machine learning powered algorithm that, based on some details of the property, prices of others properties in the market and Google Trends for the property location, recommends the best price for a property. This will help property owners to easily and quickly maintain an optimal price in line with the market trends.
- Implementation of tools to allow **editing property details** in *PropertEase*, ensuring these changes are reflected across all the listing services.
- Implementation of the feature that allows the property owner to **send the door key via email to the guest of a reservation.** The property owner can quickly view bookings for each property and check upcoming check-ins, streamlining processes such as key delivery directly through *PropertEase*.
- **Management of events, such as cleaning and maintenance.** From *PropertEase*, owners can manage some events that may affect the property's availability. When creating one of those events, this information is sent back to the listing services where the property is listed and closes the calendar for the corresponding dates to avoid bookings during those periods.
- Development of a **user-friendly and intuitive interface** for users, based on conducted usability tests, aimed to ensure pleasant usage of *PropertEase*.
- **Data analysis** using Elasticsearch-Logstash-Kibana (ELK) provides the visualization of

various data correlations, for example, in the perception and use of the dynamic pricing adjustment algorithm by the property owners. It also allows the determination of trends in use, definition of marketing strategies and the identification of improvements that can be adopted.

7.3 Future Work

In the future, remaining unimplemented use cases would be fully implemented, these include:

- Administrator Use Cases:
 1. **Remove user account:** allowing the administrator to remove a particular user from the system, when explicitly requested
 2. **Remove property:** allowing the administrator to remove a particular property from a certain user, when explicitly requested
- Property Owner Use Cases:
 1. **View general statistics:** allowing the property owner to view statistics about his properties and reservations, related to revenue or occupation
 2. **View property on chosen linked listing service:** allowing the user to view a URL that redirects him to his property on a given external listing service
 3. **Set extra (cleaning, maintenance...) costs:** allowing the property owner to add extra costs to his property which would alter the final cost per night when setting the price of a property
 4. **Set reservation acceptance type (automatic/manual):** allowing the property owner to decide whether to allow *PropertEase* to automatically accept reservations when they are made (and consequently propagate them to all relevant listing services) or to manually accept reservations based on their judgment

Apart from these unimplemented features, we would lead our efforts into optimizing scalability. Scalability was one of our priorities when designing our architecture, but, in particular, the *Website Wrappers* could be optimized, by re-designing them to have multiple instances of wrappers for each listing service. This way, we could distribute the load of scheduled events, which are not frequent (but consume a lot of computational and network resources), across multiple replicas, increasing performance and availability.

Furthermore, regarding the detection of duplicated properties in the Property Service there is one aspect which could be improved upon. In the Property Service, it would be convenient to have a mechanism to merge property details when importing a duplicated property, rather than keeping just the details from the firstly imported property that originally came from another service.

Another important future step would be to validate user interface changes applied after this first round of usability tests, by having a second round of usability tests. This second round should include more tasks and try to evaluate all use cases the user can perform, to ensure that the end user experience is great for all parts of the system.

An important note is that the current way of sending data to Elasticsearch, documents are split by property, and whenever new data is sent, the previous data gets replaced. So if one wants to use Elasticsearch in order to check price recommendations over time to identify potential peaks from Google Trends, the method of sending data to Elasticsearch needs to be changed: instead of using the property ID as the document ID, create a new ID for each submission and include a timestamp as an attribute so that all data is stored, allowing for historic analysis of price recommendations.

Moreover, it would be interesting to perform tests to validate the non-functional requirements of performance, scalability, reliability and availability that were defined, and to enforce complete GDPR compliance given the ATT project context.

Finally, as mentioned in Chapter 6, the algorithm could be improved by using data from different cities, which could be achieved by using *PropertEase's* properties. Additionally, without the limitation of using simulated properties, it would be possible to measure the results and improve each component as needed.

Glossary

Channel Manager a manager that handles the connection of a property to a variety of property listing websites/channels . 7

References

- J. Brooke. Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189, 11 1995.
- G. Dinis, Z. Breda, C. Costa, and O. Pacheco. Google trends in tourism and hospitality research: a systematic literature review. *Journal of Hospitality and Tourism Technology*, ahead-of-print, 08 2019. doi: 10.1108/JHTT-08-2018-0086.
- M. L. O. Freitas. Helping people in lisbon to predict airbnb prices, 2022. URL <https://www.kaggle.com/code/maurylukas/helping-people-in-lisbon-to-predict-airbnb-prices>. Last accessed 5 March 2024.
- InsideAirbnb. Inside airbnb: Get the data, 2024. URL <https://insideairbnb.com/get-the-data>. Last accessed 5 March 2024.
- I. Instituto Nacional de Estatística. Estatísticas do turismo - 2022, 2022.
- I. Instituto Nacional de Estatística. Resultados preliminares 2023, 2024. URL https://www.ine.pt/xportal/xmain?xpid=INE&xpgid=ine_destaques&DESTAQUESdest_boui=593971813&DESTAQUESmodo=2. Last accessed 1 March 2024.
- J. Sauro and J. R. Lewis. *Quantifying the user experience: Practical statistics for user research*. Morgan Kaufmann, 2016.

8 Appendix A - Business Layer API Documentation

8.1 User Service API documentation

UserService 1.0.0 OAS 3.1

</api/UserService/openapi.json>

The User Service exposes many endpoints for handling user account creation/signing in, and connecting users to external listing services. It also triggers workflows related to importing external data, such as scheduled messages for importing new properties and new reservations.

Servers

▾

Authorize

healthcheck ^

GET /health Perform a Health Check ^

Parameters Try it out

No parameters

Responses

Code	Description	Links
200	Return HTTP Status Code 200 (OK)	No links

Media type

▾

Controls Accept header.

Example Value | Schema

```
"string"
```


Client



GET

/users Get user account information



Get user account information, given the bearer token received from Firebase authentication.

Parameters

Try it out

No parameters

Responses

Code	Description	Links
------	-------------	-------

200	Successful Response	No links
-----	---------------------	----------

Media type

application/json

Controls Accept header.

Example Value | Schema

```
{
  "email": "user@example.com",
  "id": 0,
  "connected_services": [
    {
      "title": "zooking"
    }
  ]
}
```



404	User not found	No links
-----	----------------	----------

Media type

application/json

Example Value |

```
{
  "detail": "User not found"
}
```

POST /users Create a new user account  



Create a new user account, given the bearer token received from Firebase authentication.

Parameters Try it out

No parameters

Responses

Code	Description	Links
201	Successful Response	No links
Media type: <input type="text" value="application/json"/>		
Controls Accept header.		
Example Value Schema		
<pre>{ "email": "user@example.com", "id": 0, "connected_services": [] }</pre>		
400	Email already registered	No links
Media type: <input type="text" value="application/json"/>		
Example Value		
<pre>{ "detail": "Email already registered" }</pre>		

GET /services List all available listing services  

Get a list of all available listing services that can be connected to.

Try it out

Parameters

No parameters

Responses

Code	Description	Links
200	List all available listing services	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

```
[
  {
    "title": "zooking"
  },
  {
    "title": "clickandgo"
  },
  {
    "title": "earthstayin"
  }
]
```

POST

/services Connect to an external listing service



Connect to an external listing service, given the service title.

Parameters

Try it out

No parameters

Request body **required**

application/json

Example Value | Schema

```
{  
  "title": "zooking"  
}
```

Responses

Code	Description	Links
201	Successful Response Media type <input type="text" value="application/json"/> ▼ <small>Controls Accept header.</small> Example Value Schema	No links
	<pre>{ "email": "user@example.com", "id": 0, "connected_services": [{ "title": "zooking" }] }</pre>	
400	User is already connected to that service Media type <input type="text" value="application/json"/> ▼ Example Value	No links
	<pre>{ "detail": "User is already connected to that service" }</pre>	
404	User not found Media type <input type="text" value="application/json"/> ▼ Example Value	No links
	<pre>{ "detail": "User not found" }</pre>	
422	Validation Error	No links

Code	Description	Links
	<p>Media type</p> <p><input type="text" value="application/json"/></p> <p>Example Value Schema</p> <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>	

Schemas ^

- AvailableService** > Expand all **string**
- HTTPValidationError** > Expand all **object**
- Service** > Expand all **object**
- User** > Expand all **object**
- ValidationError** > Expand all **object**

8.2 Property Service API documentation

PropertyService 1.0.0 OAS 3.1

</api/PropertyService/openapi.json>

The Property Service exposes many endpoints for manipulating data related to properties, such as updating properties as obtaining data related to them, such as available amenities, bed types and bathroom fixtures. All endpoints require authorization, verified by the Authorization bearer token.

Servers

▾

Authorize

healthcheck ^

GET /health Perform a Health Check ^

Parameters Try it out

No parameters

Responses

Code	Description	Links
200	Return HTTP Status Code 200 (OK)	No links

Media type

▾

Controls Accept header.

Example Value | Schema

```
"string"
```

properties



GET

/properties List all properties for a specific user.



Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Return a list of all properties for a user, based on his authorization token.	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

```
[
  {
    "_id": 0,
    "user_email": "user@example.com",
    "title": "string",
    "address": "string",
    "location": "string",
    "description": "string",
    "price": 0,
    "number_guests": 0,
    "square_meters": 0,
    "bedrooms": {
      "bedroom1": {
        "beds": [
          {
            "number_beds": 2,
            "type": "single"
          }
        ]
      }
    },
    "bathrooms": {
      "bathroom1": {
        "fixtures": [
          "bathtub",
          "shower"
        ]
      }
    }
  }
]
```




GET

/properties/{prop_id} Get a specific property for a specific user.

Parameters

Try it out

Name Description

prop_id * required
integer
(path)

Responses

Code Description Links

200 Return a specific property for a user, based on his authorization token. *No links*



Media type

Controls Accept header.

Example Value | Schema

```
{
  "_id": 0,
  "user_email": "user@example.com",
  "title": "string",
  "address": "string",
  "location": "string",
  "description": "string",
  "price": 0,
  "number_guests": 0,
  "square_meters": 0,
  "bedrooms": {
    "bedroom1": {
      "beds": [
        {
          "number_beds": 2,
          "type": "single"
        }
      ]
    }
  },
  "bathrooms": {
    "bathroom1": {
      "fixtures": [
        "bathtub",
        "shower"
      ]
    }
  }
},
```

Code	Description	Links
404	Property not found for given user. Media type <input type="text" value="application/json"/> Example Value <pre>{ "detail": "Property 0 not found for user user@example.com." }</pre>	No links
422	Validation Error Media type <input type="text" value="application/json"/> Example Value Schema <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>	No links

PUT /properties/{prop_id} Update a specific property for a specific user.  

Parameters Try it out

Name	Description
prop_id * required integer (path)	<input type="text" value="prop_id"/>

Request body required

Example Value | Schema

```
{  
  "title": "string",  
  "address": "string",
```

```
"location": "string",
"description": "string",
"price": 0,
"number_guests": 0,
"square_meters": 0,
"bedrooms": {
  "bedroom1": {
    "beds": [
      {
        "number_beds": 2,
        "type": "single"
      }
    ]
  }
},
"bathrooms": {
  "bathroom1": {
    "fixtures": [
      "bathtub",
      "shower"
    ]
  }
},
"amenities": [
  "free_wifi"
]
```

Responses

Code	Description	Links
200	Return the updated property for a user, based on his authorization token.	No links



Media type

Controls Accept header.

Example Value | Schema

```
{
  "_id": 0,
  "user_email": "user@example.com",
  "title": "string",
  "address": "string",
  "location": "string",
  "description": "string",
  "price": 0,
  "number_guests": 0,
  "square_meters": 0,
  "bedrooms": {
    "bedroom1": {
      "beds": [
        {
          "number_beds": 2,
          "type": "single"
        }
      ]
    }
  },
}
```

Code	Description	Links
404	<pre>"bathrooms": { "bathroom1": { "fixtures": ["bathtub", "shower"] } }</pre> <p>Property not found for given user.</p> <p>Media type</p> <p><input type="text" value="application/json"/></p> <p>Example Value</p> <pre>{ "detail": "Property 0 not found for user user@example.com." }</pre>	No links
422	<p>Validation Error</p> <p>Media type</p> <p><input type="text" value="application/json"/></p> <p>Example Value Schema</p> <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>	No links

GET /amenities List all available amenities.  

Parameters Try it out

No parameters

Responses



Code	Description	Links
200	Return a list of all available amenities.	No links

Media type

Controls Accept header.

Example Value | Schema

```
[  
  "free_wifi",  
  "parking_space",  
  "air_conditioner",  
  "pool",  
  "kitchen"  
]
```

GET /bathroom_fixtures List all available bathroom fixtures.  

Parameters [Try it out](#)

No parameters

Responses



Code	Description	Links
200	Return a list of all available bathroom fixtures.	No links

Media type

Controls Accept header.

Example Value | Schema

```
[  
  "bathtub",  
  "shower",  
  "bidet",  
  "toilet"  
]
```

GET /bed_types List all available bed types.  

Parameters Try it out

No parameters

Responses

Code	Description	Links
200	Return a list of all available bed types.	No links

Media type
application/json ▾
Controls Accept header.

Example Value | Schema

```
[  
  "single",  
  "queen",  
  "king"  
]
```

Schemas ^

- Amenity** > Expand all **string**
- Bathroom** > Expand all **object**
- BathroomFixture** > Expand all **string**
- Bed** > Expand all **object**
- BedType** > Expand all **string**
- Bedroom-Input** > Expand all **object**

Bedroom-Output > Expand all **object**

Contact > Expand all **object**

HTTPValidationError > Expand all **object**

HouseRules > Expand all **object**

Property > Expand all **object**

Service > Expand all **string**

TimeSlot > Expand all **object**

UpdateProperty > Expand all **object**

ValidationError > Expand all **object**

8.3 Calendar Service API documentation

CalendarService 1.0.0 OAS 3.1

</api/CalendarService/openapi.json>

The Calendar Service exposes many endpoints for manipulating data displayed in the calendar, including reservations and cleaning/maintenance (management) events. It also allows users to handle key management for their properties. All events endpoints require authorization, verified by the Authorization bearer token.

Servers

▾

[Authorize](#)

healthcheck ^

GET /health Perform a Health Check ^

Parameters

[Try it out](#)

No parameters

Responses

Code	Description	Links
200	Return HTTP Status Code 200 (OK)	No links

Media type

▾



Controls Accept header.

Example Value | Schema

```
"string"
```

events



GET /events/management/types List available management event types.  


Parameters Try it out

No parameters

Responses

Code	Description	Links
200	Return a list of all available management event types.	No links



Media type




Controls Accept header.

Example Value | Schema

```
[  
  "maintenance",  
  "cleaning"  
]
```



GET /events List all events of a specific user, including reservations and cleaning/maintenance events.  

Parameters Try it out

Name	Description
reservation_status * required string (query)	Available values : confirmed, pending, canceled <input type="text" value="confirmed"/> 

Responses

Code	Description	Links
200	Successful Response	No links
Media type <input type="text" value="application/json"/>		
Controls Accept header.		
Example Value Schema		
<pre>[{ "id": 0, "property_id": 0, "owner_email": "user@example.com", "begin_datetime": "2024-06-01T13:37:50.829Z", "end_datetime": "2024-06-01T13:37:50.829Z", "type": "cleaning", "service": "zooking" }]</pre>		
422	Validation Error	No links
Media type <input type="text" value="application/json"/>		
Example Value Schema		
<pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>		

GET /events/management/maintenance List of maintenance events related to a user and his properties  

Returns list of maintenance events related to all, or a specific property, of a specific user based on his authorization bearer token.

Parameters Try it out

Name	Description
------	-------------

property_id integer (query)

Responses

Code	Description	Links
------	-------------	-------

200	Successful Response	No links
-----	---------------------	----------

Media type

Controls Accept header.

Example Value | Schema



```
[
  {
    "id": 0,
    "property_id": 0,
    "owner_email": "user@example.com",
    "begin_datetime": "2024-06-01T13:37:50.833Z",
    "end_datetime": "2024-06-01T13:37:50.833Z",
    "type": "maintenance",
    "company_name": "string"
  }
]
```

422	Validation Error	No links
-----	------------------	----------

Media type

Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

POST /events/management/maintenance Create new maintenance event  

Creates a new maintenance event for the specified property and the given timeframe

Parameters Try it out

No parameters

Request body required application/json



Example Value | Schema

```
{
  "company_name": "Maintenance Lda.",
  "property_id": "1",
  "begin_datetime": "2024-05-30T10:37:34",
  "end_datetime": "2024-05-31T10:37:34"
}
```

Responses

Code	Description	Links
201	Successful Response	No links
	Media type application/json Controls Accept header.	
	Example Value Schema	
	<pre>{ "id": 0, "property_id": 0, "owner_email": "user@example.com", "begin_datetime": "2024-06-01T13:37:50.837Z", "end_datetime": "2024-06-01T13:37:50.837Z", "type": "maintenance", "company_name": "string" }</pre>	
404	Specified property for creating event does not exist.	No links
	Media type application/json	

Code	Description	Links
	<p>Example Value</p> <pre>{ "detail": "There are no registered properties for email user@example.com which you can create events for." }</pre>	
409	<p>There are overlapping events with the event to be created.</p> <p>Media type</p> <p>application/json ▾</p> <p>Example Value</p> <pre>{ "detail": "There are overlapping events with the event with begin_datetime 2024-05-30T10:37:34 and end_datetime 2024-05-31T10:37:34." }</pre>	No links
422	<p>Validation Error</p> <p>Media type</p> <p>application/json ▾</p> <p>Example Value Schema</p> <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>	No links

GET /events/management/cleaning List of cleaning events related to a user and his properties  

Returns list of cleaning events related to all, or a specific property, of a specific user based on his authorization bearer token.

Parameters Try it out

Name	Description
------	-------------

property_id integer (query)

Responses

Code	Description	Links
------	-------------	-------

200	Successful Response	No links
-----	---------------------	----------

Media type

Controls Accept header.

Example Value | Schema



```
[
  {
    "id": 0,
    "property_id": 0,
    "owner_email": "user@example.com",
    "begin_datetime": "2024-06-01T13:37:50.841Z",
    "end_datetime": "2024-06-01T13:37:50.841Z",
    "type": "cleaning",
    "worker_name": "string"
  }
]
```

422	Validation Error	No links
-----	------------------	----------

Media type

Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

POST /events/management/cleaning Create new cleaning event  

Creates a new cleaning event for the specified property and the given timeframe

Parameters Try it out

No parameters

Request body required application/json



Example Value | Schema

```
{
  "worker_name": "Great person",
  "property_id": "1",
  "begin_datetime": "2024-05-30T10:37:34",
  "end_datetime": "2024-05-31T10:37:34"
}
```

Responses

Code	Description	Links
201	Successful Response	No links
	Media type application/json Controls Accept header.	
	Example Value Schema	
	<pre>{ "id": 0, "property_id": 0, "owner_email": "user@example.com", "begin_datetime": "2024-06-01T13:37:50.846Z", "end_datetime": "2024-06-01T13:37:50.846Z", "type": "cleaning", "worker_name": "string" }</pre>	
404	Specified property for creating event does not exist.	No links
	Media type application/json	

Code	Description	Links
	<p>Example Value</p> <pre>{ "detail": "There are no registered properties for email user@example.com which you can create events for." }</pre>	
409	<p>There are overlapping events with the event to be created.</p> <p>Media type</p> <p>application/json ▾</p> <p>Example Value</p> <pre>{ "detail": "There are overlapping events with the event with begin_datetime 2024-05-30T10:37:34 and end_datetime 2024-05-31T10:37:34." }</pre>	No links
422	<p>Validation Error</p> <p>Media type</p> <p>application/json ▾</p> <p>Example Value Schema</p> <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>	No links

GET /events/reservation List of reservations related to a user and his properties.  

Returns list of reservations related to all, or a specific property, of a specific user based on his authorization bearer token.

Parameters **Try it out**

Name **Description**

property_id
integer
(query)

Responses

Code **Description** **Links**

200 Successful Response *No links*

Media type

Controls Accept header.

Example Value | Schema

```
[
  {
    "id": 0,
    "property_id": 0,
    "owner_email": "user@example.com",
    "begin_datetime": "2024-06-01T13:37:50.853Z",
    "end_datetime": "2024-06-01T13:37:50.853Z",
    "type": "reservation",
    "service": "zooking",
    "reservation_status": "confirmed",
    "client_email": "user@example.com",
    "client_name": "string",
    "client_phone": "strings",
    "cost": 0
  }
]
```



422 Validation Error *No links*

Media type

Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

Code	Description	Links
	<pre>] }</pre>	

PUT /events/management/maintenance/{event_id} Update maintenance event  

Updates the maintenance event with the given id and the specified parameters. If the begin_datetime or end_datetime parameters are updated, the system will check for overlapping events.

Parameters

[Try it out](#)

Name	Description
event_id * required integer (path)	<input type="text" value="event_id"/>

Request body required

application/json 

Example Value | Schema

```
{
  "begin_datetime": "2024-03-21T11:00:00",
  "end_datetime": "2024-03-21T12:00:00",
  "company_name": "John Doe's Company"
}
```

Responses

Code	Description	Links
200	Successful Response	No links



Media type
 

Controls Accept header.

Example Value | Schema

```
{
  "id": 0,
```

Code	Description	Links
	<pre>"property_id": 0, "owner_email": "user@example.com", "begin_datetime": "2024-06-01T13:37:50.858Z", "end_datetime": "2024-06-01T13:37:50.858Z", "type": "maintenance", "company_name": "string" }</pre>	
404	<p>Specified event for updating does not exist.</p> <p>Media type <input type="text" value="application/json"/></p> <p>Example Value </p> <pre>{ "detail": "Event of type maintenance with id 0 not found for email u ser@example.com." }</pre>	No links
409	<p>There are overlapping events with the given time interval for the event to be updated.</p> <p>Media type <input type="text" value="application/json"/></p> <p>Example Value </p> <pre>{ "detail": "There are overlapping events with the event with begin_da tetime 2024-03-21T11:00:00 and end_datetime 2024-03-21T12:00:00." }</pre>	No links
422	<p>Validation Error</p> <p>Media type <input type="text" value="application/json"/></p> <p>Example Value Schema</p> <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>	No links

DELETE	/events/management/maintenance /{event_id}	Delete maintenance event	 
---------------	---	--------------------------	---

Deletes the maintenance event with the given id.

Parameters

Try it out

Name	Description
------	-------------

event_id * required

integer
(path)

event_id

Responses

Code	Description	Links
------	-------------	-------

204	Event deleted successfully.	No links
-----	-----------------------------	----------

Media type

application/json

Controls Accept header.

Example Value

```
{}
```

404	Specified event for deletion does not exist.	No links
-----	--	----------

Media type

application/json

Example Value



```
{  "detail": "Event of type maintenance with id 0 for email user@example.com not found."}
```

422	Validation Error	No links
-----	------------------	----------

Media type

application/json

Code	Description	Links
	Example Value Schema	
	<pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>	

PUT /events/management/cleaning/{event_id} Update cleaning event  

Updates the cleaning event with the given id and the specified parameters. If the begin_datetime or end_datetime parameters are updated, the system will check for overlapping events.

Parameters

[Try it out](#)

Name	Description
event_id * required integer (path)	<input type="text" value="event_id"/>

Request body required

application/json 

[Example Value](#) | [Schema](#)

```
{
  "begin_datetime": "2024-03-21T11:00:00",
  "end_datetime": "2024-03-21T12:00:00",
  "worker_name": "John Doe"
}
```

Responses

Code	Description	Links
200	<p>Successful Response</p> <p>Media type</p> <p><input type="text" value="application/json"/></p> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>{ "id": 0, "property_id": 0, "owner_email": "user@example.com", "begin_datetime": "2024-06-01T13:37:50.867Z", "end_datetime": "2024-06-01T13:37:50.867Z", "type": "cleaning", "worker_name": "string" }</pre>	No links
404	<p>Specified event for updating does not exist.</p> <p>Media type</p> <p><input type="text" value="application/json"/></p> <p>Example Value </p> <pre>{ "detail": "Event of type cleaning with id 0 not found for email user @example.com." }</pre>	No links
409	<p>There are overlapping events with the given interval for the event to be updated.</p> <p>Media type</p> <p><input type="text" value="application/json"/></p> <p>Example Value </p> <pre>{ "detail": "There are overlapping events with the event with begin_datetime 2024-03-21T11:00:00 and end_datetime 2024-03-21T12:00:00." }</pre>	No links
422	<p>Validation Error</p> <p>Media type</p> <p><input type="text" value="application/json"/></p> <p>Example Value Schema</p> <pre>{ "detail": [{</pre>	No links

Code	Description	Links
	<pre> "loc": ["string", 0], "msg": "string", "type": "string" }] } </pre>	

DELETE /events/management/cleaning/{event_id} Delete cleaning event 🔒 ⤴

Deletes the cleaning event with the given id.

Parameters Try it out

Name	Description
event_id * required integer (path)	<input type="text" value="event_id"/>

Responses

Code	Description	Links
204	Event deleted successfully. Media type <input type="text" value="application/json"/> <small>Controls Accept header.</small> Example Value <pre>{}</pre>	No links
404	Specified event for deletion does not exist. Media type	No links

Code	Description	Links
	<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;"> application/json </div> <p>Example Value </p> <pre style="background-color: #2e3436; color: #eeeeec; padding: 10px;">{ "detail": "Event of type cleaning with id 0 for email user@example.com not found." }</pre>	
422	<p>Validation Error</p> <p>Media type</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;"> application/json </div> <p>Example Value Schema</p> <pre style="background-color: #2e3436; color: #eeeeec; padding: 10px;">{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>	No links

POST
/events/reservation/{reservation_id}/email_key
Send email with key to reservation client

Sends an email to the client of the reservation with the given id. The email contains the key to the property.

Parameters

Name	Description
reservation_id * required integer (path)	<div style="border: 1px solid #ccc; padding: 5px; min-height: 20px;">reservation_id</div>

Try it out

Request body required application/json

Example Value | Schema

```
{  
  "key": "string"  
}
```

Responses

Code	Description	Links
------	-------------	-------

204	Successful Response.	No links
-----	----------------------	----------

Media type

Controls Accept header.

Example Value |

```
{}
```

404	Reservation with the given id for the email does not exist.	No links
-----	---	----------

Media type

Example Value |

```
{  
  "detail": "Reservation with id 0 for the email user@example.com not found."  
}
```

422	Validation Error	No links
-----	------------------	----------

Media type

Example Value | Schema

```
{  
  "detail": [  
    {  
      "loc": [  
        "string",  
        0  
      ],  
      "msg": "string",  
      "type": "string"  
    }  
  ]  
}
```

Code	Description	Links
	<pre>] }</pre>	


Schemas ^

- Base** > Expand all **object**
- CleaningWithId** > Expand all **object**
- HTTPValidationError** > Expand all **object**
- KeyInput** > Expand all **object**
- MaintenanceWithId** > Expand all **object**
- ReservationStatus** > Expand all **string**
- ReservationWithId** > Expand all **object**
- Service** > Expand all **string**
- UniformEventWithId** > Expand all **object**
- ValidationError** > Expand all **object**

9 Appendix B - Usability Tests

9.1 Tasks

Tasks



You are agreeing that you have read and accepted the conditions outlined in the General Data Protection Regulation.

Yes

No

Task 1
Import the properties from Zooking and check how many were imported.

A sua resposta _____

Task 2 Difficulty Level

1 - Very difficult

2

3

4

5 - Very easy

Task 2
Import properties from Clickandgo and verify if the properties in that service have reservations on May 14th.

A sua resposta _____

Task 2 Difficulty Level

1 - Very difficult

2

3

4

5 - Very easy

Task 3

Which amenities does the house "A mais fixe de todas 🏠🛀🚿" have?

A sua resposta _____

Task 3 Difficulty Level

- 1 - Very difficult
- 2
- 3
- 4
- 5 - Very easy

Task 4

Which is the price per night to stay in "Cool house 🏠🔧"? After finding that select the option to update it automatically with the recommended price.

A sua resposta _____

Task 4 Difficulty Level

- 1 - Very difficult
- 2
- 3
- 4
- 5 - Very easy

Task 5

Create a Cleaning Event and after that check how many Cleaning and Maintenance Events exists on the "Cool house 🏠🔧".

A sua resposta _____

Task 5 Difficulty Level

- 1 - Very difficult
- 2
- 3
- 4
- 5 - Very easy

Task 6

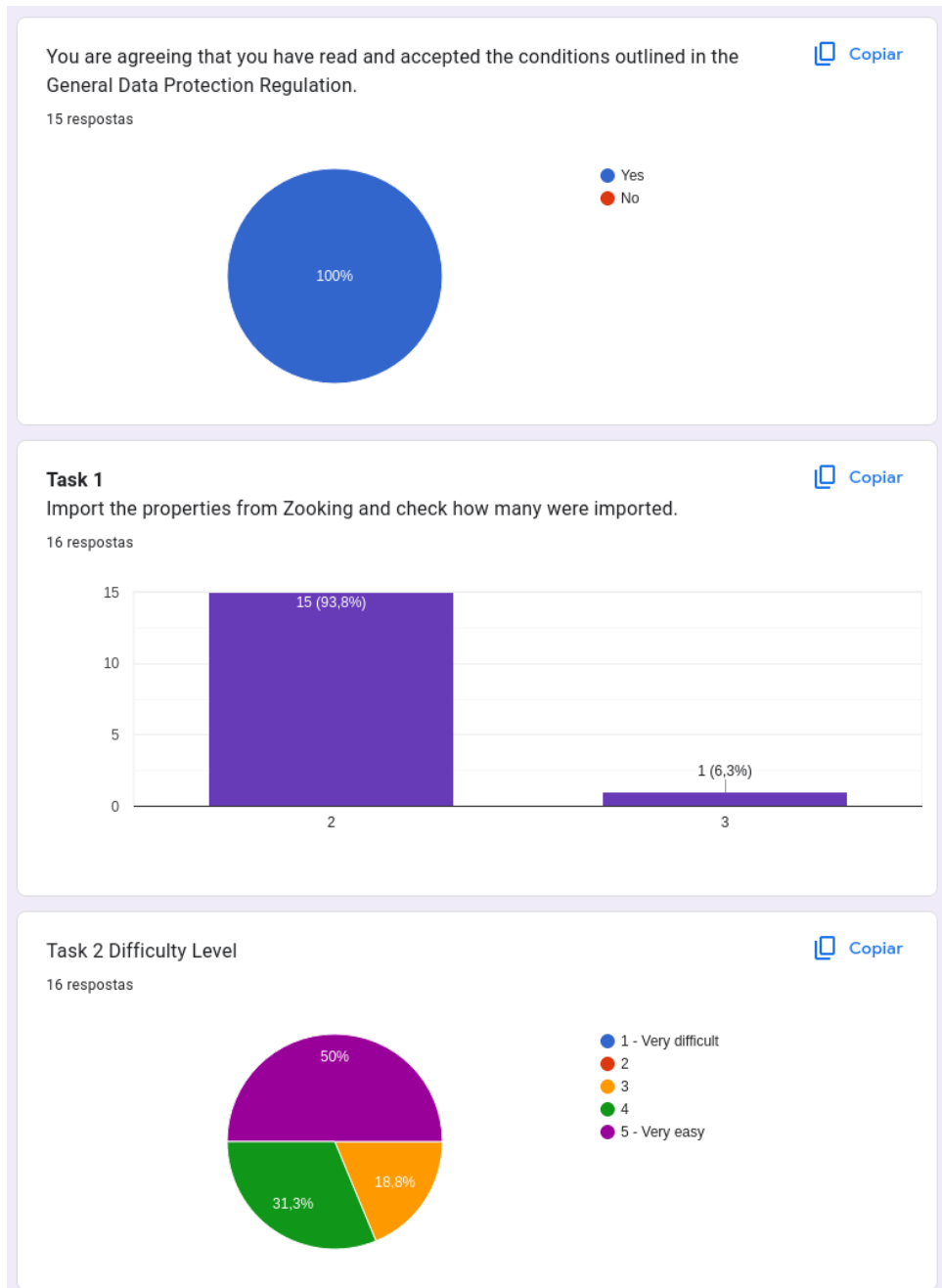
Send a key code to the client at the first reservation that exists in "Cool house 🏠". Check if the send was successful and identify the name of the client who made this reservation.

A sua resposta _____

Task 6 Difficulty Level

- 1 - Very difficult
- 2
- 3
- 4
- 5 - Very easy

9.1.1 Tasks Responses

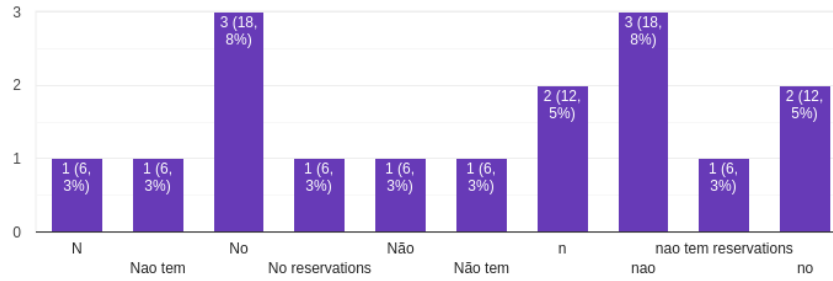


Task 2

 Copiar

Import properties from Clickandgo and verify if the properties in that service have reservations on May 14th.

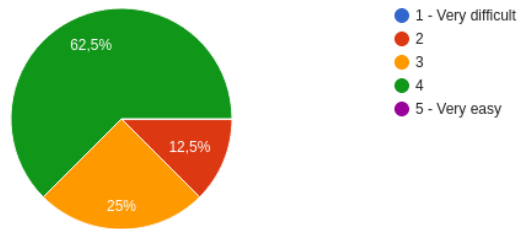
16 respostas



Task 2 Difficulty Level

 Copiar

16 respostas

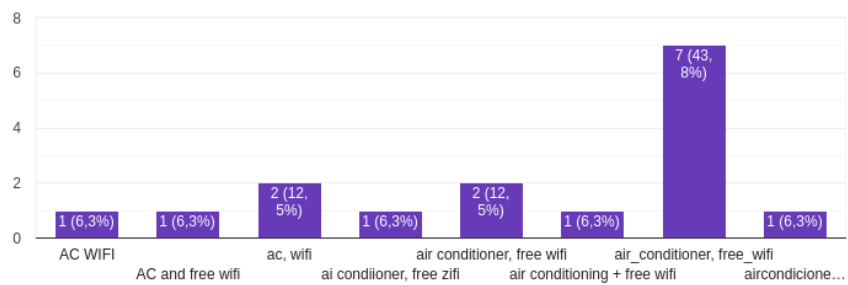


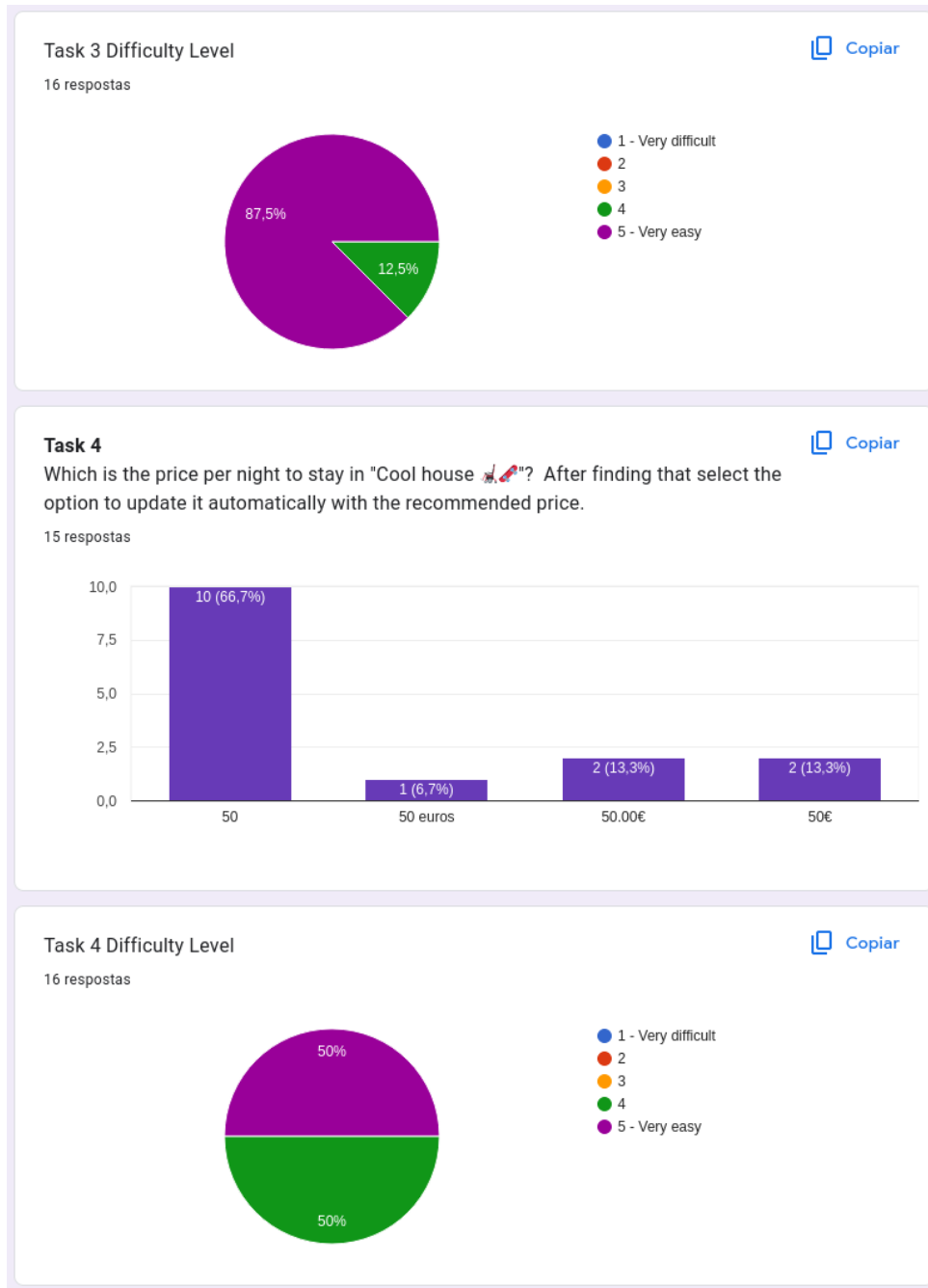
Task 3

 Copiar

Which amenities does the house "A mais fixe de todas 🏠🧺" have?

16 respostas



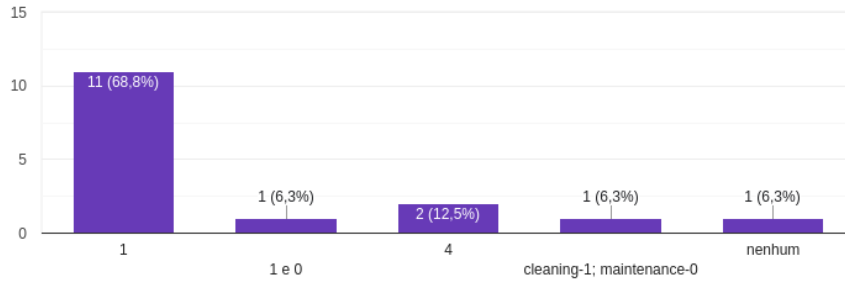


Task 5

 Copiar

Create a Cleaning Event and after that check how many Cleaning and Maintenance Events exists on the "Cool house 🏠🔧".

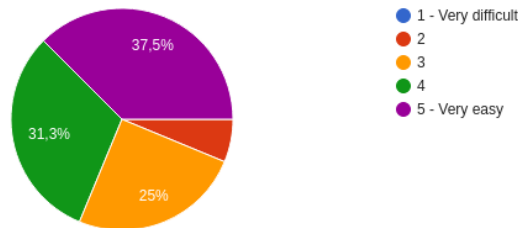
16 respostas



Task 5 Difficulty Level

 Copiar

16 respostas

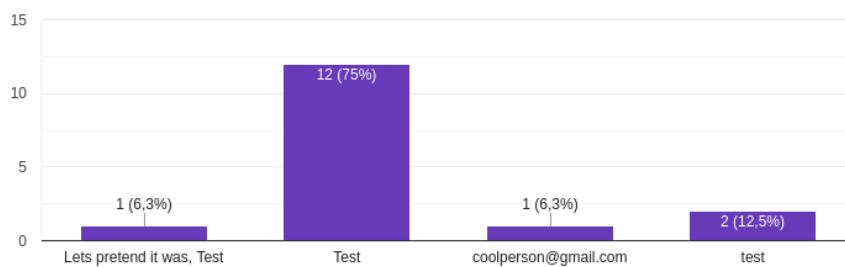


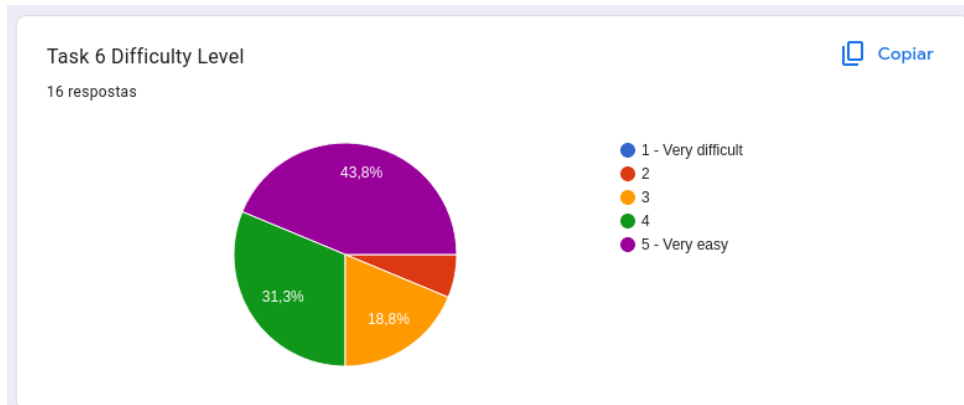
Task 6

 Copiar

Send a key code to the client at the first reservation that exists in "Cool house 🏠🔧". Check if the send was successful and identify the name of the client who made this reservation.

16 respostas






9.2 Post Task Questionnaire

Post Task Questionnaire

Thank you for your cooperation with this study, which aims to evaluate the User Interface of the PropertEase app and, try to improve it following the Usability criteria.

Your collaboration is important for the success of this evaluation, so we ask you to complete this questionnaire, the data of which will be used in total anonymity for scientific purposes only.



*** Indica uma pergunta obrigatória**

Demographic data

N. Mec.

A sua resposta _____

Gender

Female

Male

Other

Age

A sua resposta _____

Previous experience with this type of application/system

None

Some

A lot

You are agreeing that you have read and accepted the conditions outlined in the General Data Protection Regulation.

Yes

No

Overall opinion on the application/system (SUS)

After using the application/system and taking into account your final assessment, check the circle that best reflects your opinion regarding its usage. If you believe that these quantifications are not applicable, choose NA.

I think that I would like to use this system frequently. *

- 1 - Totally disagree
- 2
- 3
- 4
- 5 - Totally agree
- NA

I found the system unnecessarily complex. *

- 1 - Totally disagree
- 2
- 3
- 4
- 5 - Totally agree
- NA

I thought the system was easy to use. *

- 1 - Totally disagree
- 2
- 3
- 4
- 5 - Totally agree
- NA

I think that I would need the support of a technical person to be able to use this system. *

- 1 - Totally disagree
- 2
- 3
- 4
- 5 - Totally agree
- NA

I found the various functions in this system were well integrated. *

- 1 - Totally disagree
- 2
- 3
- 4
- 5 - Totally agree
- NA

I thought there was too much inconsistency in this system. *

- 1 - Totally disagree
- 2
- 3
- 4
- 5 - Totally agree
- NA

I would imagine that most people would learn to use this system very quickly. *

- 1 - Totally disagree
- 2
- 3
- 4
- 5 - Totally agree
- NA

I found the system very cumbersome to use. *

- 1 - Totally disagree
- 2
- 3
- 4
- 5 - Totally agree
- NA

I felt very confident using the system. *

- 1 - Totally disagree
- 2
- 3
- 4
- 5 - Totally agree
- NA

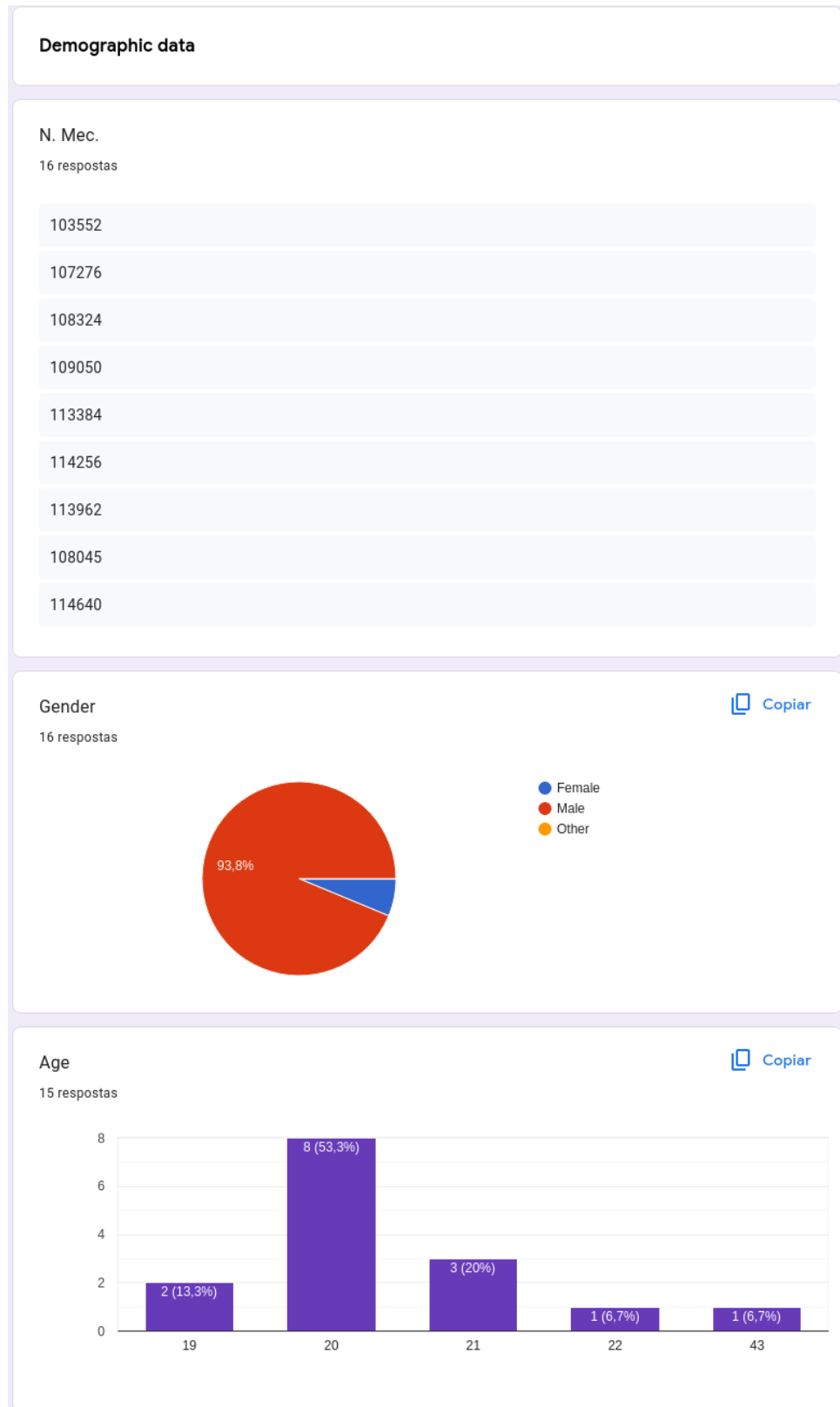
I needed to learn a lot of things before I could get going with this system. *

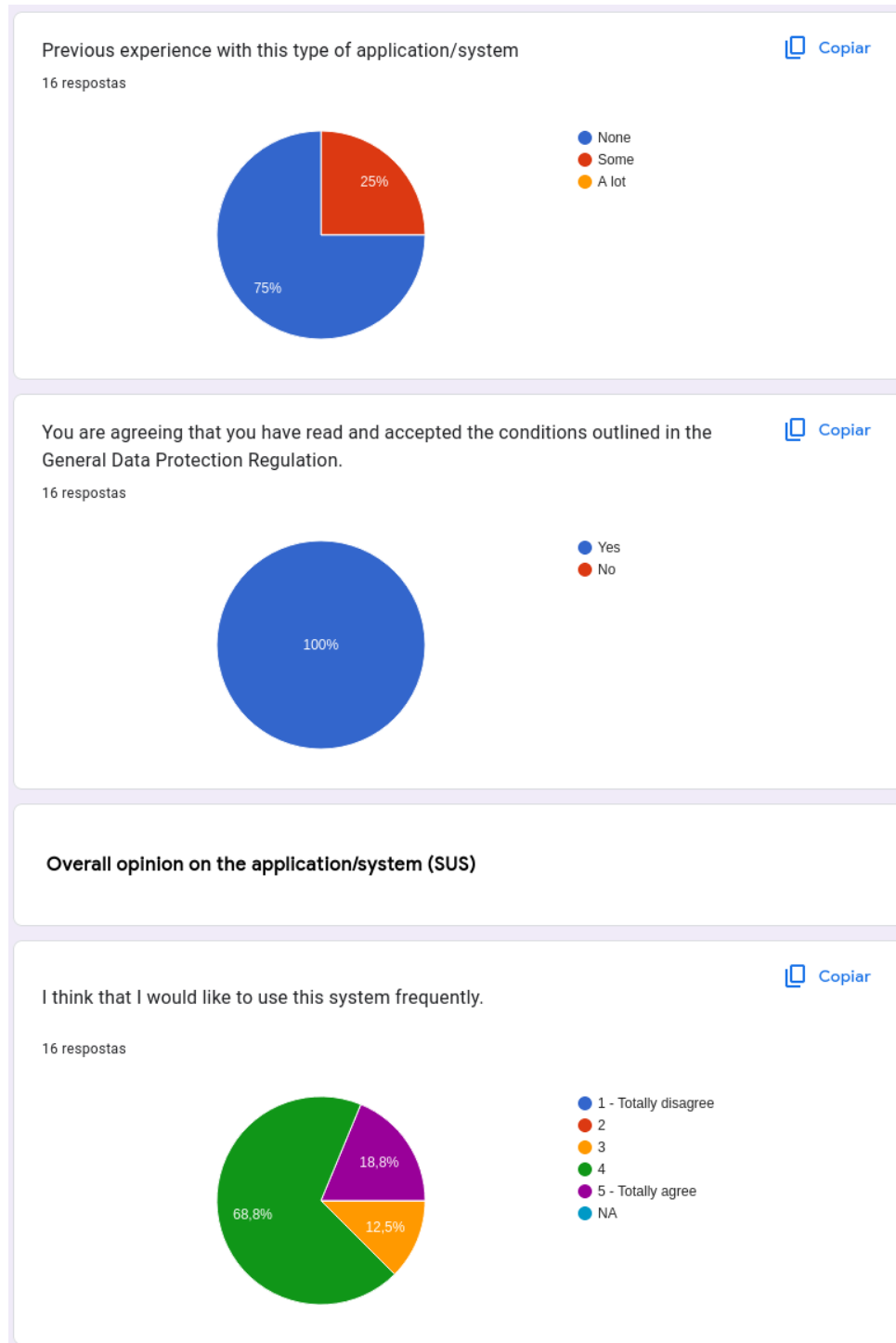
- 1 - Totally disagree
- 2
- 3
- 4
- 5 - Totally agree
- NA

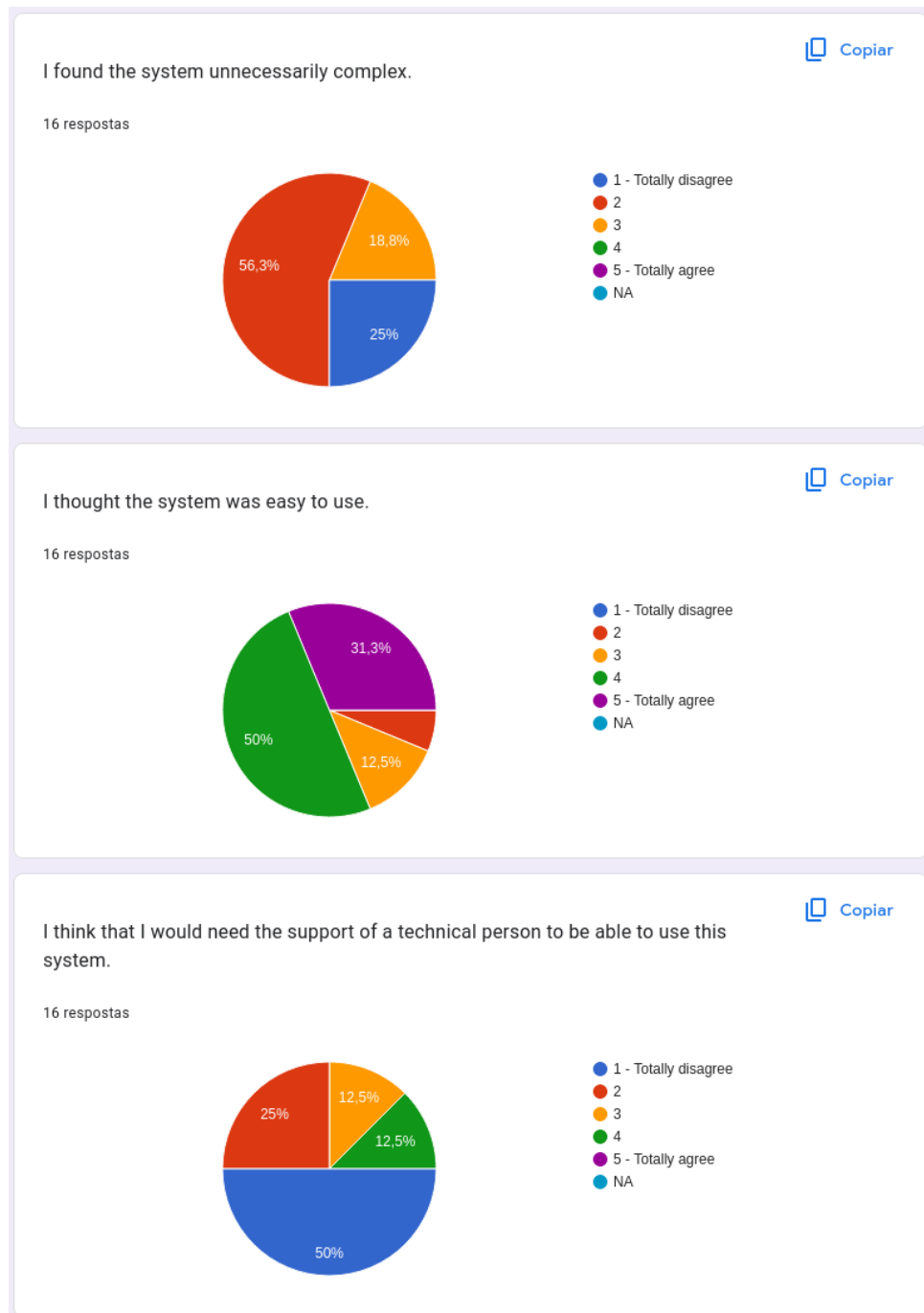
Please leave any comments about the user experience provided by the application/
system:

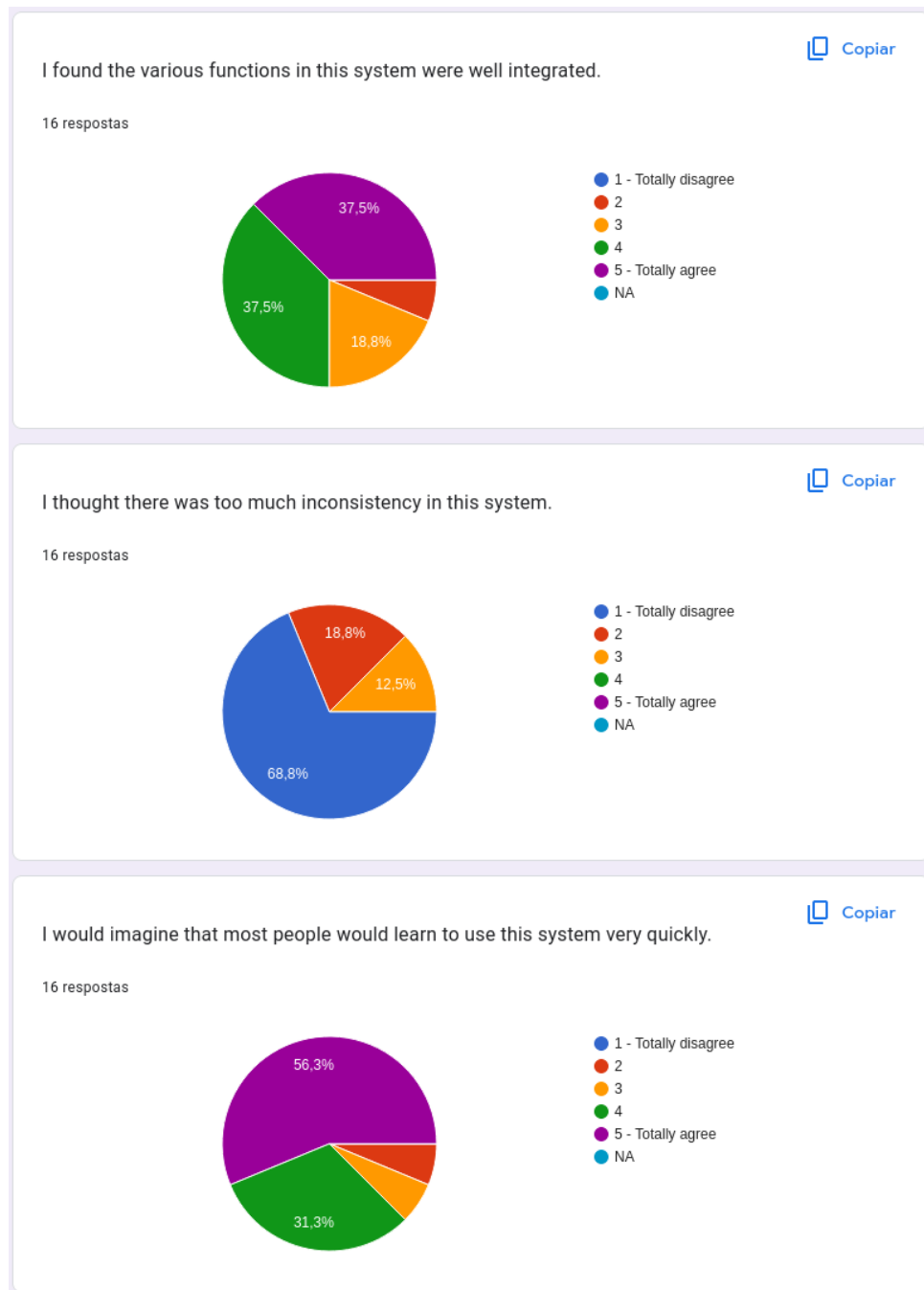
A sua resposta

9.2.1 Post Task Questionnaire Responses





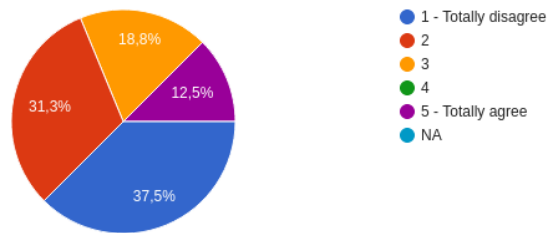




I found the system very cumbersome to use.

 Copiar

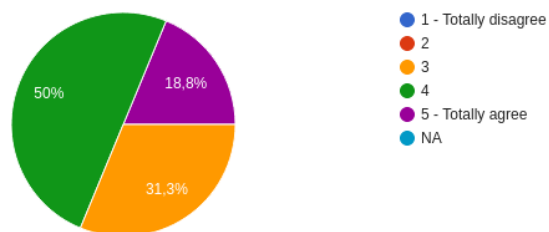
16 respostas



I felt very confident using the system.

 Copiar

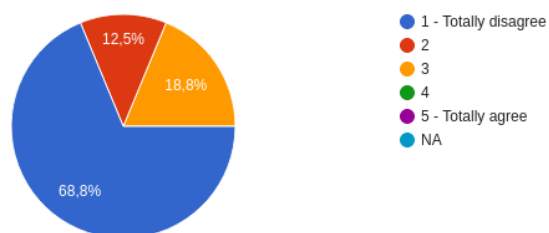
16 respostas



I needed to learn a lot of things before I could get going with this system.

 Copiar

16 respostas



Please leave any comments about the user experience provided by the application/system:

6 respostas

Aparecer em properties a integration

Se o calendário desta semana tivesse completo com alguma reserva, seria mais facil de perceber a vantagem do mesmo

Adicionar ordenação por Arrival e Departure na tabela de eventos

Gostei de usar o sistema e acho seria algo útil na vida real. Tentava colocar mais feedback ao utilizador após fazer certas tarefas.

given during evaluation

Dito durante os testes

10 Appendix C - Example Property Schema

```
1 {
2   "id": 1,
3   "user_email": "someemail@gmail.com",
4   "title": "Coolest house ever",
5   "address": "Braga (obvio que nao)",
6   "location": "Porto",
7   "description": "Dont enter",
8   "price": 100,
9   "number_guests": 20,
10  "square_meters": 4000,
11  "bedrooms": {
12    "bedroom_1": {
13      "beds": [
14        {
15          "number_beds": 1,
16          "type": "single"
17        }
18      ]
19    }
20  },
21  "bathrooms": {
22    "bathroom_1": {
23      "fixtures": [
24        "bathtub",
25        "shower"
26      ]
27    }
28  },
29  "amenities": [
30    "free_wifi",
31    "pool",
32    "kitchen"
```

```
33 ],
34 "after_commission": false,
35 "house_rules": {
36   "check_in": {
37     "begin_time": "10:59",
38     "end_time": "20:40"
39   },
40   "check_out": {
41     "begin_time": "10:59",
42     "end_time": "20:40"
43   },
44   "smoking": false,
45   "parties": true,
46   "rest_time": {
47     "begin_time": "23:00",
48     "end_time": "06:00"
49   },
50   "allow_pets": true
51 },
52 "additional_info": "So pessoas fixes",
53 "cancellation_policy": "",
54 "contacts": [
55   {
56     "name": "Alvaro Barbosa",
57     "phone_number": "+351969314716"
58   }
59 ],
60 "services": [],
61 "recommended_price": 95,
62 "update_price_automatically": false
63 }
```

11 Appendix D - Contract Test Source Code Example

```
import pytest
import requests
from api_schemas.zooking_schema import ZookingPropertyBase
from api_schemas.base_schema import ReservationBase
from pydantic import ValidationError

ZOOKING_URL = "http://localhost:8000"

def test_zooking_property_schema():
    zooking_property = requests.get(f"{ZOOKING_URL}/properties/1")
    try:
        ZookingPropertyBase.model_validate(zooking_property.json())
    except (ValidationError, requests.exceptions.JSONDecodeError):
        pytest.fail("Failed to validate Zooking API property schema.")

def test_zooking_reservation_schema():
    # this endpoint returns an array of reservations for property 10
    zooking_reservation = requests.get(f"{ZOOKING_URL}/reservations/9")
    try:
        ReservationBase.model_validate(zooking_reservation.json()[0])
    except (ValidationError, requests.exceptions.JSONDecodeError):
        pytest.fail("Failed to validate Zooking API reservation schema.")
```

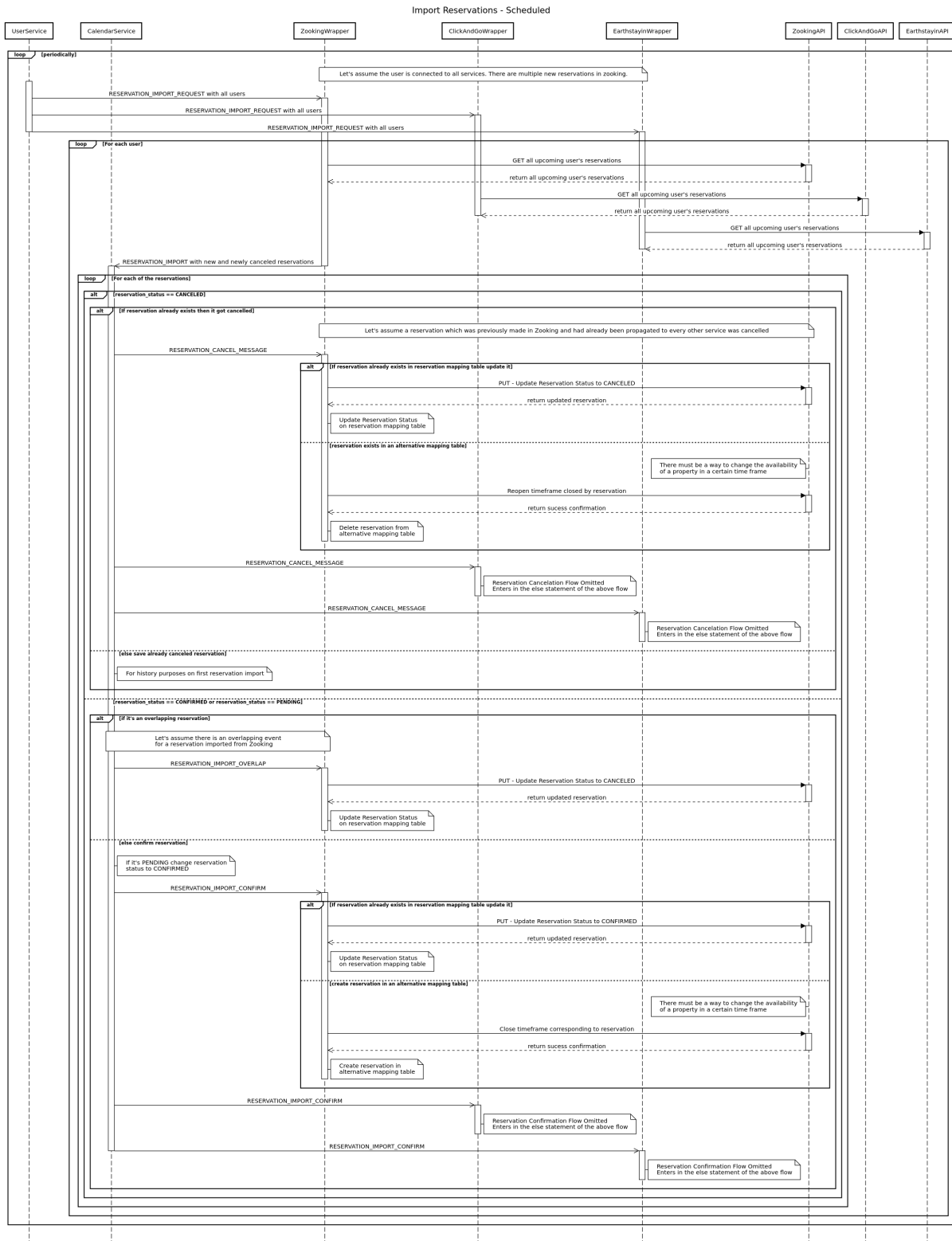



Figure 39: UML Sequence Diagram of scheduled flow to import reservations from external services