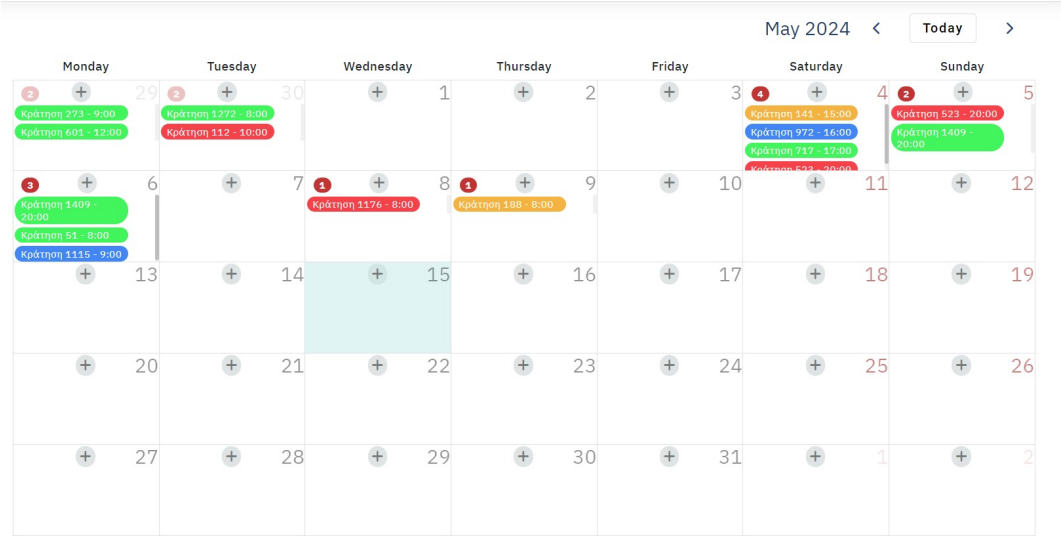
ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

INTERNATIONAL HELLENIC UNIVERSITY
DEPARTMENT OF INFORMATION AND ELECTRONIC ENGINEERING

DIPLOMA THESIS

Web application of booking, monitoring, management, usage statistics of shared halls and auditoriums of the IHU



Student:
Gakis Dimitrios, Tsilikas Ioannis
Student ID: 144237, 164768

Supervisor:
Marina Delianidi

15 May 2024

- Front-end: dim2409/Booking-Service
- Back-end: Ioatsi/BookingServiceBackend
- Live: booking.iee.ihu.gr

Web application of booking, monitoring, management, usage statistics of shared halls and auditoriums of the IHU.
Code of Dissertation 23361
Student's full name Gakis Dimitrios, Tsilikas Ioannis
Supervisor's full name Marina Delianidi
Date of undertaking 28-12-2023
Date of completion 23-05-2024

We hereby affirm the authorship of this paper as well as the acknowledgement and credit of whichever assistance We received in its composition. We have, furthermore, noted the various sources from which We extracted data, ideas, visual or written material, in paraphrase or exact quotation. Moreover, we affirm the exclusive composition of this paper by myself only, for the purpose of it being a dissertation, in the Department of Information and Electronic Engineering of the I.H.U.

The approval of this dissertation by the Department of Information and Electronic Engineering of the International Hellenic University, does not necessarily entail the adoption of the author's views, on behalf of the Department.

Dedication

We would like to dedicate and thank our beloved colleague and lifetime friend Mr Georgios Tsourdiou, who has been a major assistive factor in the completion of this work. We would also like to thank the supervising professors, pr Marina Delianidi, pr Konstantinos I. Diamantaras and pr Antonis Sidiropoulos who have guided and lead us through the making and conceiving of this project. We would also like to thank the entire faculty of the Department of Information and Electronic Engineering who provided us the tools and knowledge to complete our studies and acquire the necessary knowledge that has assisted us in this endeavor. Finally we would like to thank our family and friends without whom this thesis would not have seen the light of day.

Prolog

As universities continue to evolve in the digital age, the management of campus resources stands as a critical component of daily operations, impacting both staff and student activities. Efficient allocation and utilization of resources, such as campus halls, not only enhance the educational experience but also streamline the administrative workload. This paper introduces an innovative platform designed to address the inefficiencies plaguing the traditional methods of booking campus halls at a university.

The development of this platform was initiated to confront and rectify these inefficiencies, introducing a streamlined and effective system tailored to the unique needs of a university. Traditional methods, characterized by their lack of coordination, often led to confusion among university personnel and hampered the effective use of shared spaces.

This thesis will explore the development, implementation, and implications of this booking system. Through detailed analysis and real-world application, the paper will demonstrate how technology can effectively replace outdated systems, offering a case study in enhancing administrative efficiency and user experience in a university setting. The following chapters will delve into the technical architecture, user interaction, and the broader impact of the system on university operations, providing a comprehensive overview of a modern solution to an age-old problem.

By examining this transformative approach, the thesis aims to contribute valuable insights into the discourse on educational technology and administrative efficiency, fostering further innovation and adaptation in university settings.

- Front-end: dim2409/Booking-Service
- Back-end: Ioatsi/BookingServiceBackend
- Live: booking.iee.ihu.gr

Summary

This paper details the conceptualization and development of a digital booking platform specifically tailored for managing the reservation of campus halls within a university setting. The project addresses prevalent inefficiencies in resource scheduling and seeks to provide a structured, secure, and department-specific solution that leverages university credentials for access management and conflict resolution.

The research employs a theoretical approach, grounded in systems analysis and design methodologies, to construct a platform that integrates seamlessly with existing university administrative structures. The system's architecture enhances security, confidentiality, conflict resolution and ease of use, significantly streamlining the booking process compared to traditional methods.

In conclusion, this paper highlights the transformative potential of digital solutions in streamlining educational administrative processes. It suggests that the strategies outlined could be applied to broader resource management challenges within universities. It provides insights into the efficacy of the developed booking platform and offers proposals for its improvement. It underscores the potential of digital solutions to refine administrative processes within educational institutions.

Abstract

EN

This thesis explores the development and implications of a new digital booking system designed to assist and facilitate the management of campus hall reservations within a university setting. The primary objective of this project is to replace outdated administrative processes with a streamlined, secure, and user-friendly digital platform. This project was developed using a theoretical approach grounded in systems analysis and design methodologies to architect a solution that leverages existing university administrative structures, ensuring confidentiality and reducing booking conflicts.

The study establishes that the platform effectively addresses the identified inefficiencies in hall reservation processes by providing a structured, secure, and department-specific scheduling system. This has significant implications for improving administrative operations in educational settings, showcasing the potential for digital solutions to improve efficiency and user experience within university administration. The work contributes to the field by demonstrating how targeted digital solutions can transform university administration and by proposing further system development.

EL

Η παρούσα διατριβή εξετάζει την ανάπτυξη και τις επιπτώσεις ενός νέου ψηφιακού συστήματος κράτησης, σχεδιασμένου για να βελτιώσει τη διαχείριση των κρατήσεων αιθουσών στο πλαίσιο ενός πανεπιστημίου. Ο κύριος στόχος αυτού του έργου είναι η αντικατάσταση των παρωχημένων διοικητικών διαδικασιών με μια ασφαλή και φιλική προς τον χρήστη ψηφιακή πλατφόρμα. Το έργο αυτό αξιοποίησε μια θεωρητική προσέγγιση βασισμένη σε μεθοδολογίες ανάλυσης και σχεδιασμού συστημάτων για να δημιουργήθει μια λύση που ενσωματώνεται στις υπάρχουσες διοικητικές δομές του πανεπιστημίου, εξασφαλίζοντας την
εμπιστευτικότητα και μειώνοντας τις συγκρούσεις κρατήσεων.

Η μελέτη διαπιστώνει ότι η πλατφόρμα αντιμετωπίζει τα υπάρχοντα μειονεκτήματα στις διαδικασίες κράτησης αιθουσών παρέχοντας ένα δομημένο, ασφαλές και ειδικά προσαρμοσμένο σύστημα. Αυτό έχει σημαντικές επιπτώσεις στη βελτίωση της διοικητικής λειτουργείας, αναδεικνύοντας τη δυνατότητα των ψηφιακών λύσεων να ενισχύσουν την αποδοτικότητα και την εμπειρία των χρηστών στη διοίκηση των πανεπιστημίων.

Η έρευνα συμβάλλει στον τομέα, δείχνοντας πώς οι στοχευμένες ψηφιακές λύσεις μπορούν να μεταμορφώσουν τη διοίκηση των πανεπιστημίων και προτείνοντας βελτιώσεις για περαιτέρω ανάπτυξη του συστήματος.

Contents

Chapter 1

Technology Analysis

In the rapidly evolving field of web development, choosing a framework significantly influences the architecture, sustainability, and user experience of applications. A framework, by definition, serves as a foundational structure designed to support the development of software applications. It provides a pre-defined skeleton or template which developers can modify selectively by adding or customizing features. This approach utilizes common resources such as libraries and toolkits, ensuring consistency across different parts of the application.

When developing a robust web application, the choice of technology stack—Angular for the frontend and Laravel for the backend was crucial. These frameworks were not selected at random; instead, they were chosen based on specific criteria that align with the long-term goals and functional needs of the project. This section of the paper will delve into the reasons behind the selection of these technologies, comparing them against other popular frameworks based on several critical criteria: community support, ease of migration, and performance.

1.1 The Role of Frameworks in Web Development

Frameworks simplify the development process by providing a generic, yet customizable system. Developers can build upon the base functionality provided by the framework, focusing on adding unique features rather than reinventing common functionalities. This not only accelerates the development process but also reduces costs by minimizing the effort needed to create complex functionalities from scratch. The structured nature of frameworks ensures that applications are built with clean, maintainable code that adheres to industry standards, guaranteeing reliability and scalability.

Moreover, frameworks dictate the flow of control within the application, meaning they manage the order of processing. This inversion of control simplifies the management of dependencies and the execution flow, particularly in complex applications that handle multiple user interactions and data transactions, such as a booking system.

1.2 Criteria for Framework Evaluation

The criteria selected for evaluating web development frameworks in this project are foundational to its success and sustainability:

- **Community Support**: An active community is paramount to the longevity of a framework. Community support ensures that the framework stays updated with the latest web standards and security patches. It also provides a reservoir of resources for troubleshooting, shared knowledge, and pre-built modules or plugins, which are crucial for extending the functionality of the application over time.
- **Ease of Migration**: The ability to adapt to evolving needs without extensive overhauls is essential for any application expected to serve over a long period. Frameworks that allow easy migration to newer versions without significant downtime or rewriting of code stand out as particularly suitable for an educational institution's dynamic environment. This involves fewer dependencies and a modular architecture that can evolve as needs change.
- **Performance**: For an application expected to handle multiple simultaneous user requests, such as booking and managing events, performance is a critical measure. A framework that can handle high loads, process requests efficiently, and provide rapid responses enhances user satisfaction and contributes to the overall success of the application.

1.3 Structure of the Analysis

This analysis will compare Angular and Laravel with their contemporaries, dissecting how each criterion has influenced the choice of these frameworks for the development of the service. By examining these frameworks under the lenses of community support, popularity, migration ease, and performance, this section aims to provide a comprehensive understanding of why Angular and Laravel were the optimal choices for this project.

1.4 Frontend Frameworks Comparison: Angular, React, and Vue

As web technologies continue to evolve, the decision of selecting the right frontend framework becomes pivotal for the success of any web application. The development of the service, designed to manage bookings for campus halls through a dynamic calendar interface, brings unique challenges that require careful consideration of the framework's features. This section will explore and compare three leading frontend technologies: Angular, React, and Vue, focusing on their community support, performance, and code stability.

1.4.1 Why Compare Angular, React, and Vue?

Angular, React, and Vue stand out in the web development landscape for their capabilities and widespread adoption. Each offers distinct advantages and caters to different development philosophies and project requirements [1].

Angular

- Launched by Google in 2010 [2]
- Structured and extensive community support system [1]
- Over 1.7 million developers and contributors [2]
- Strong online presence with forums, blogs, tutorials, and courses [1]
- Comprehensive and regularly updated documentation [1]
- Rich set of development tools like Angular CLI [2]

React

- Developed by Facebook in 2013 [1]
- Large ecosystem with independent developers and agencies [1]
- Extensive contributions on GitHub [3]
- Abundance of resources including tutorials and start-up projects [3]
- Online support through forums, Stack Overflow, and social media [4]

Vue

- Created by Evan You in 2014 [1]
- Exponential growth in community [1]
- Smaller but open and inclusive community [5]
- Highly active on GitHub [3]
- Rich ecosystem of plugins and companion libraries [3]
- Clear and thorough documentation [1]

1.4.2 Community Support

Angular

Launched by Google in 2010, Angular offers a structured and extensive community support system. With over 1.7 million developers and contributors, Angular enjoys a vast online presence, including forums, blogs, tutorials, and courses. Its documentation is comprehensive and regularly updated. Additionally, Angular offers a rich set of development tools like Angular CLI [2].

React

Developed by Facebook in 2013, React has a large ecosystem with contributions from independent developers and agencies. The framework sees widespread activity on GitHub and provides a plethora of resources, including tutorials and start-up projects. Online support is available through forums, Stack Overflow, and various social media platforms [4].

Vue

Created by Evan You in 2014, Vue has experienced exponential growth in its community. While smaller compared to Angular and React, Vue's community is open and inclusive, with high activity on GitHub. Vue benefits from a rich ecosystem of plugins and companion libraries, and its documentation is clear and thorough [5].

Table 1.1: GitHub Repository Metrics

	Angular	**React**	**Vue**
Stars	83.9k	7.9k	32.7k
Forks	22.2k	6.4k	5.9k
Watchers	3.1k	199	746
Contributors	1614+	1622+	332+

1.4.3 Ease of Migration

This section evaluates the ease of migration for Angular, React, and Vue, examining their approaches and tools for managing upgrades.

Angular

Angular, developed by Google, underwent a significant redesign with Angular 2+ in 2016, marking a transition from AngularJS to a more robust framework using TypeScript. This shift was strate-

gic, positioning Angular for long-term viability. Angular's release cycle is highly structured, offering major updates every six months with backward compatibility guarantees, ensuring smooth transitions between versions. Additionally, Angular follows a long-term support (LTS) model, providing essential updates and security patches for older versions for extended periods, thus offering stability and reliability. The Angular CLI plays a pivotal role in simplifying upgrades by automating tedious tasks, checking for common errors, and offering recommendations for necessary changes, ultimately minimizing disruptions and reducing migration risks. This comprehensive approach to version management and tooling support makes Angular particularly suitable for enterprise-level applications and complex projects requiring a strategic and forward-thinking migration approach [2].

React

React, developed by Facebook, adopts a flexible approach to migrations by ensuring backward compatibility with its library updates, allowing developers to adopt new features without immediate overhauls of existing code. The use of codemods assists in this process by automating the refactoring of code to new patterns, facilitating periodic updates with minimal disruption. While React's flexibility is advantageous for many projects, it can lead to ambiguity in larger, more structured projects where consistent behavior across versions is crucial. Developers are tasked with devising careful migration strategies to ensure that component updates do not introduce unexpected issues, requiring meticulous planning and execution to maintain stability and consistency [4].

Vue

Vue, known for its developer-friendly approach, offers clear and detailed upgrade paths, making migrations generally straightforward and accessible. The framework's design allows for incremental adoption, beneficial for projects requiring gradual integration or updates, enhancing flexibility and developer convenience. Vue's community provides extensive documentation and tools like the Vue CLI to support migrations, ensuring developers of all skill levels can navigate the process effectively. However, Vue's relatively smaller ecosystem and lighter touch approach may necessitate more hands-on management during migrations, especially for larger scale applications. Developers may need to invest additional effort to maintain consistency and stability compared to Angular's comprehensive features and enterprise-focused tooling [5].

1.4.4 Performance

This section analyzes the performance characteristics of Angular, React, and Vue, considering factors such as rendering efficiency and scalability.

Angular

Angular is a TypeScript-based framework known for its comprehensive approach to building applications. It employs a full-fledged MVC architecture, effectively separating concerns for fine-tuned control over application behavior and rendering. A standout feature is Ahead-of-Time (AoT) compilation, which converts TypeScript and HTML into efficient JavaScript during build time, significantly improving startup speed and responsiveness. It also supports Server-Side Rendering (SSR) through Angular Universal, enhancing perceived load time and aiding SEO by rendering applications on the server [2].

React

React uses a Virtual DOM for efficient component updates by recalculating differences before rendering. Its component-based architecture and JSX integration allow for dynamic user interface development in the same file as JavaScript logic. Performance efficiency often depends on application complexity, with initial load times potentially lagging behind Angular's AoT compilation. React's flexibility and less opinionated structure may lead to varied implementations and efficiency levels, impacting performance predictability [4].

Vue

Vue offers efficient performance through its Virtual DOM and reactive data binding system, suitable for small to medium-sized projects. Performance remains commendable due to its simplicity and minimalistic approach, tracking dependencies for selective re-rendering. However, scalability may pose challenges, particularly in larger applications with complex state management and frequent DOM updates. The lack of built-in mechanisms like AoT compilation, as in Angular, may impact initial load time and runtime efficiency in very complex applications [5].

1.5 Front-end Conclusion

This analysis assesses web development frameworks for a university event booking platform, concluding that Angular is the most appropriate choice. This decision is based on a detailed evaluation of community support, migration ease, and performance. Below are the key factors that substantiate Angular's suitability for this project:

- **Community and Support**: Angular is developed and maintained by Google, which provides a significant advantage in terms of reliability and ongoing development. The framework has a large and active community which contributes to its vast array of learning resources, and tooling, notably the Angular CLI. This extensive support system ensures Angular's viability and continuous improvement, crucial for maintaining an operational booking system.

Categories	Angular	React	Vue
Type	Framework	Rich library to create UI	Library
Maintained By	Google	Facebook	Evan You and core team
Initial Release	September 2016	March 2013	February 2014
Development Approach	Based on TypeScript	Based on JavaScript	Based on HTML and JavaScript
Ideal For	Large-scale apps with rich features	User interfaces based on UI components	Single-page apps
Developer-friendly	Structure-based approach	Ensures flexible development	Focuses on the separation of concerns
Written In	TypeScript	JavaScript	JavaScript

Table 1.2: Comparison of Angular, React, and Vue

- **Migration Strategy**: Angular offers a structured approach to upgrades and migrations, featuring regular release cycles and backward compatibility. Its long-term support (LTS) model provides essential updates and security patches for older versions, minimizing the impact of upgrades on existing system functionality. This approach reduces the risks and costs associated with maintaining the system, making Angular suitable for long-term projects within dynamic environments like universities.

- **Performance Capabilities**: Angular is equipped to handle complex applications through mechanisms like Ahead-of-Time (AoT) compilation, which optimizes the application's performance by compiling TypeScript and HTML into JavaScript during the build phase. Angular also supports Server-Side Rendering (SSR), enhancing load times and search engine visibility. These features are critical for ensuring that the booking system operates efficiently under varying load conditions.

- **Framework Completeness**: Angular provides a complete solution for application development, which includes built-in capabilities for managing forms, routing, and state. This reduces the need to integrate multiple external libraries and simplifies overall system maintenance. For a university setting, where resource constraints are common, Angular's all-in-one nature allows for more streamlined development and lower long-term maintenance costs.

- **Stability and Scalability**: Angular's enterprise-level features ensure that the booking platform is both stable and scalable. This framework is capable of supporting the platform's growth and the integration of new features as needed. Angular's stability and capacity for expansion make it a strategic choice for educational institutions planning for future enhancements.

In summary, Angular is identified as the most fitting framework due to its robust support network, efficient migration strategies, performance optimization, and comprehensive development tools. An-

gular's features align well with the university's requirements for a sustainable, adaptable, and efficient system, making it the recommended choice for this project.

Question	Angular	Vue.js	React.js
Is only js import sufficient to get started?	No	Yes	Yes
Server-side rendering	Yes	Yes	Yes
The ability to write and maintain a large quantity of complex code	Yes	Partly	Partly
Should the developer rely on third party packages?	No	Partly	Yes
Data state management	Yes	Yes	Yes

Table 1.3: Comparison of Angular, Vue.js, and React.js

1.6 Backend Frameworks Comparison: Laravel vs. Node.js

In the realm of web development, selecting the right backend framework is crucial for the successful implementation and long-term maintenance of web applications. For our university booking system, which requires robust, reliable, and scalable solutions, the choice of backend technology is especially significant. This comparison evaluates Laravel and Node.js, two leading technologies with distinct approaches and capabilities.

1.6.1 Community Support

Laravel

Founded by Taylor Otwell in 2011, Laravel has quickly grown to become one of the most popular PHP frameworks. Extensive and active community contributes to a broad range of robust packages and tools, evident in its vibrant forums and social media presence. Rich educational resources, including comprehensive and regularly updated official documentation, extensive video tutorials from Laracasts, and a plethora of blogs and online courses [6] [7].

Node.js

Launched in 2009 by Ryan Dahl, Node.js has a vast and diverse community due to its use of JavaScript, one of the most widely used programming languages. Largest package manager (npm) hosts thousands of open-source libraries which foster a rich development environment and a wide array of tools and plugins that enhance productivity. Highly active on GitHub with a significant number of contributors continually improving the core and its peripherals, ensuring that new features and optimizations are regularly available. Wide range of learning materials available including official documentation and courses tailored to different expertise levels [8] [9].

1.6.2 Performance

Laravel

Processing Model: Laravel uses a synchronous processing model, which processes tasks sequentially within its PHP environment. This model can introduce latency but is supported by Laravel's cache system that can speed up responses for frequent requests. Tools for Optimization: Laravel includes features such as caching, queuing, and session management, which are designed to improve performance under load. Its architecture also supports horizontal scaling, allowing it to manage increased loads by distributing them across multiple servers. Database Interaction: Laravel's Eloquent ORM facilitates database queries with concise syntax, which can speed up development. View Compilation: Laravel's Blade templating engine precompiles views into plain PHP, caching them until they change, which minimizes overhead on subsequent requests [6] [10].

Node.js

I/O Model: Node.js operates on an event-driven, non-blocking I/O model, enabling it to handle multiple requests concurrently. This model is effective for applications that require high concurrency and real-time data interaction. Execution Engine: Node.js utilizes the Google V8 JavaScript engine, which compiles JavaScript directly into machine code. This feature enhances execution speed, particularly for processing numerous requests simultaneously. Microservices Architecture: Node.js is suited for microservices architecture due to its lightweight and efficient execution, which allows different parts of an application to scale independently. Real-time Capabilities: Node.js is optimal for applications that need to manage a lot of real-time data, such as applications involving web sockets for live interactions [8] [9].

1.6.3 Code Stability and Maintainability

Laravel

Structured MVC Architecture: Laravel follows the Model-View-Controller (MVC) architecture, which helps in organizing code in a structured way. This separation of concerns facilitates easier maintenance and debugging, as developers can focus on specific areas of the application without impacting others. Convention Over Configuration: Laravel's philosophy of convention over configuration means many decisions are predefined by the framework. This reduces the amount of boilerplate code developers need to write and maintain, leading to a more stable codebase. Built-in Tools: Laravel provides several built-in tools for common tasks such as migration, testing, and routing. These tools are tightly integrated and maintained as part of the framework, ensuring consistency and reducing the likelihood of conflicts that can arise with third-party plugins. Comprehensive Documentation and Community Best Practices: Laravel's extensive documentation and community-driven best practices guide developers in writing consistent, secure, and maintainable code. This support helps in maintaining code quality, especially as new developers join the project [6] [7].

Node.js

Modular Design: Node.js encourages a modular design through its extensive use of packages via npm. While this can enhance flexibility and customization, it may also lead to challenges in maintainability due to dependency management and version control issues. Asynchronous Programming Model: Node.js's asynchronous programming model can complicate code maintenance, especially for those not familiar with JavaScript's callback and promise patterns. This model can lead to "callback hell," where nested callbacks result in complex and hard-to-maintain code. Dynamic Typing: JavaScript's dynamic typing can lead to more runtime errors and less predictable code behavior, which can affect stability. This issue can be mitigated by using TypeScript with Node.js, but it requires additional setup and maintenance. Community and Ecosystem: The vast Node.js community provides numerous packages that can solve almost any problem; however, the quality and longevity of these packages can vary. Developers must carefully manage package selection and updates to maintain code stability [11] [8].

1.6.4 Scalability

Laravel

Queue System: Laravel includes a built-in queue system that helps defer the processing of a time-consuming task until a later time, which can improve request handling and user response times. This feature is vital for operations that can be processed in the background, enhancing the application's scalability. Caching Mechanisms: Effective use of caching can significantly reduce the load on the application, thereby improving response times and scalability. Laravel supports popular cache backends like Redis and Memcached directly, which can be essential for scaling applications. Database Indexing and Eloquent ORM: While Eloquent ORM makes database interactions easy and readable, Laravel also supports advanced features like indexing, which can optimize query performance and scalability when handling large volumes of data [6] [10].

Node.js

Non-blocking and Asynchronous Nature: Node.js excels in scalability due to its non-blocking and asynchronous I/O model. This allows Node.js applications to handle a large number of simultaneous connections with low overhead, making it ideal for real-time applications that require intensive I/O operations. Microservices Architecture Compatibility: Node.js is well-suited for microservices architectures, which are inherently scalable. Different components of the application can be scaled independently based on demand, which is efficient for managing resources and can improve application performance and resilience. Load Balancing: Node.js can be integrated with load balancers to distribute incoming network traffic across multiple instances. This distribution helps in handling more requests simultaneously, enhancing the application's ability to scale. Community and Third-party Tools: The vast Node.js community offers a plethora of tools and libraries that support scalability.

These tools can help in monitoring, logging, and dynamically scaling applications based on the load [11] [8].

Feature	Laravel	Node
Created	2011	2009
Category	Language (PHP) Framework	JavaScript Runtime Environment
Ease of Coding	Concise	Long
Popularity	0.35% websites	4.2% websites
Engine	Blade Template Engine	Google's V8 JavaScript
Package Manager	Composer Package Manager	Node Package Manager (npm)
Execution	Synchronous	Asynchronous
Execution Speed	Powerful and lightweight	Faster and lightweight
Concurrency	Multi-threaded blocking I/O	Event-driven non blocking I/O
Performance	Slower	Faster
Web Server	Doesn't require	Apache and IIS
Database	4 (MySQL, PostgreSQL, SQLite, SQL)	Relational and Conventional
JSON	json.encode	JSON.stringify() and JSON.parse
Latest Version	Laravel 9	Node 18.3.0
Community	Small but rising; shares PHP community	Vast online community

Table 1.4: Comparison of Laravel and Node

1.6.5 Conclusion: Why Laravel Over Node.js

Selecting the right backend framework for the university booking system required a thorough examination of both the technical requirements and the strategic needs of the application. After a detailed comparison between Laravel and Node.js, it became evident that Laravel is the more suitable choice. The decision was guided by several key factors that align closely with the operational, developmental, and maintenance goals of the project. 1. Structured and Maintainable Codebase 2. Long-Term Maintainability and Stability 3. Scalability Aligned with System Needs 4. Performance and Efficiency 5. Development and Operational Cost 19 Chapter 2. Technology Analysis In conclusion, while both Laravel and Node.js are capable frameworks with their respective strengths, Laravel's advantages in terms of code maintainability, structural integrity, scala- bility appropriate to the project's needs, and lower overall cost make it the optimal choice for the university booking system. This decision supports the system's requirements for a stable, maintainable, and scalable application, ensuring it remains robust and functional as it evolves to meet future demands. This strategic choice aligns with the project's goals of long-term viability and ease of management, ensuring that the system continues to serve effectively, with minimal disruption and high reliability.

Chapter 2

Problem Analysis

2.1 Current Booking Management System

The university currently relies on an Excel spreadsheet for managing bookings of campus halls. This spreadsheet is accessible to all university staff.

2.1.1 Spreadsheet System Efficiency

- Users manually register hall reservations, leading to inefficiencies and potential errors due to the volume of scheduling data.
- Without proper calendar tools, users struggle to manage bookings, risking booking conflicts and underutilization of resources.
- The lack of information about event types and expected attendance complicates the prioritization of bookings and hall selection.

2.1.2 Conflict Management

- Conflicts between multiple bookings for the same hall are common and identifying them requires manual detection in the spreadsheet, leading to inefficiencies.
- Resolving conflicts involves communication among staff members and the secretary's office, which is time-consuming and uncoordinated.

2.1.3 Recurring Bookings and Administrative Oversight

- Manual registration of recurring bookings is inefficient and prone to errors as there is no centralized system that can handle recurring bookings efficiently and provide administrative oversight.
- The absence of moderators overseeing the system results in arbitrary alterations by staff members.

The current system leads to issues in managing bookings and confusion among users. Lack of proper tools for calendar management, conflict detection, and administrative oversight exacerbates the inefficiencies. By analyzing the current booking management system, it's evident that there are several shortcomings that hinder efficient scheduling and conflict resolution. The proposed system aims to address these issues by introducing a comprehensive solution that offers enhanced functionalities for users and administrators alike.

Chapter 3

System Solution for Reservation Conflict Management

3.1 Introduction

In order to solve the above mentioned reservation and conflict management problem of the department, a system solution was proposed. The system in question is the subject of this paper and it aims to aid administration as well as faculty to make better use of the communal lecture halls. To achieve this, the booking system was designed to have two main functionalities. One that would allow users, depending on their type, to peruse a calendar with listings of all the lectures and events happening in campus halls across campus as well as make their own reservations and host these lectures or events. This would consist of the client side of this system and solve half the problem by creating a central hub that informs the public of the campus time slot schedule when it comes to conference halls. In addition it would provide a convenient portal between the faculty that want to reserve these halls and the administration. The other functionality concerns the management side of this system that would allow the administrations to manipulate the schedule of the hall time slots according to demand and need. Also, to correctly inform these management decisions the system would provide relevant statistics about the kind and frequency of use for each hall thus allowing management to determine the best schedule configuration at each given time.

Furthermore, one of the most significant purposes of this system is to offer a better alternative solution to the existing one. Therefore the system had to also be capable of detecting timing conflicts between bookings, as well as provide a simple and efficient way of resolving them. We believe that we have achieved this through the development of the web app this paper pertains to and will further explain its use and construction below.

3.2 Similar Solutions

Several existing solutions were examined as potential solutions, with their various features tailored to different needs. Understanding these systems helped in identifying the gaps and justifying the development of a custom solution that caters to specific requirements such as security, functionality, and adaptability.

3.2.1 Microsoft Calendar

Accessible through platforms like Delos365 for users with academic credentials, Microsoft's Calendar [12] is a prominent solution for similar cases. With seamless integration with other Microsoft Office tools, it provides scheduling features, reminders, and collaboration capabilities. However, it is not a custom application, and this lack of customization can be a limitation for specific use cases. The rigid structure may not adapt well to specialized needs, and the reliance on Microsoft's ecosystem could pose security and accessibility concerns.

3.2.2 SuperSaaS

SuperSaaS [13] offers a comprehensive scheduling platform with a wide array of features. This service supports various types of bookings, user management, and payment processing, making it suitable for both educational and commercial purposes. Despite its versatility, SuperSaaS might not meet all the unique requirements of an academic environment. The general-purpose design can lead to unnecessary complexity and potential security vulnerabilities.

3.2.3 Google Calendar

Google Calendar [14] is a widely-used tool with user-friendly interface and integration with Google services. It supports recurring events, detailed scheduling, and sharing options, making it a popular choice for personal and professional use. Nonetheless, its limitations include dependency on Google's infrastructure and a lack of specialized features as well as rigidity on its account based structure. These limitations make it impossible to be used as a solution for the concerning problem.

3.2.4 Justification for a Custom Booking System

The existing solutions, while functional, often fall short in areas crucial for a specialized booking system. Key factors necessitating a custom solution include:

- Security: Ensuring data privacy and secure access control is essential, especially in this academic settings where sensitive information is handled.

- Functionality: Custom features tailored to specific needs, such as room based structure and specialized user roles are essential for optimal performance.
- Adaptability: The ability to adapt the system to evolving requirements without being constrained by a third-party platform's limitations is crucial for long-term sustainability.
- Cost: A custom solution can be more cost-effective in the long run by eliminating subscription fees and allowing for targeted investments in necessary features.
- Data Ownership: Having control over the generated data over time will allow for unique assessments in the efficacy of management and provide insights otherwise impossible without a specialised solution.

By developing a custom booking system, we address these critical aspects, providing a tailored solution that enhances efficiency, security, and user satisfaction. This approach ensures that the system evolves in alignment with specific needs and institutional policies.

3.3 Objectives

The main objectives of the proposed system are as follows:

- Help administration and faculty optimize the use of communal lecture halls.
- Provide users with the ability to view event calendars, make reservations, and host events.
- Assist administration in scheduling hall time slots based on demand and need.
- Provide relevant statistics to inform management decisions.
- Detect and resolve timing conflicts across bookings efficiently.

3.4 Functionalities

The proposed system offers two main functionalities:

1. User Functionality: Allows users to browse event calendars, make reservations, and host events.
2. Management Functionality: Enables administration to alter schedule hall time slots and access relevant statistics.

3.5 First Functionality: Calendar and Booking

In order to fulfill the first purpose of this system a calendar component is required, in particular one that is capable of showcasing multiple events per day as well as allowing for the insertion of additional components on top of it that inform the user further with details for each given event. The user should be able to traverse through months and have a complete view of past present and future events. They should be able to see the specific time each event will take place along with any other relevant information. This functionality would be available to all users regardless of role and would constitute the central hub of information regarding the conference hall schedule. The next part of this functionality provides the ability to schedule a booking and is meant for faculty users who want to make use of the campus conference halls. The faculty user should be able to navigate the calendar and choose the day and time that fits their needs and then schedule an event for that date and time. They should be able to add relevant information and pick any room they please as well as have the option to create a recurring event across multiple rooms and days of the week as well as on different times.

3.5.1 Calendar Display

The calendar component should fulfill the following requirements:

- Showcase multiple events per day.
- Allow for the insertion of additional components for event details.
- Enable users to traverse through months.
- Provide a complete view of past, present, and future events.

3.5.2 Booking

The booking feature is meant for faculty users who want to make use of the campus conference halls. Faculty users should be able to:

- Navigate the calendar.
- Choose the day and time that fit their needs.
- Schedule an event for the selected date and time.
- Add relevant information to the event.
- Pick any available room.
- Optionally, create recurring events across multiple rooms, days of the week, and different times.

This feature grants faculty users complete freedom of choice for their events and puts the burden of resolving any occurring conflicts on the system and the managing administration. It encourages more event activity throughout the campus and increases the usage of all conference halls.

3.6 Second Functionality: Management and Conflict Resolution

The second functionality concerns the managing administration. Having the burden of resolving the oncoming conflicts the system should assist and provide a powerful tool in the resolution of conflicts and management of the conference hall time schedule.

3.6.1 Booking Oversight

Each room within the booking system would have overseeing moderators. These moderators should have access to view all pending and approved bookings, along with relevant information such as the room, date and time of the event, and the user who made the request.They should be able to add to, edit, approve or reject any of the existing bookings and thus have complete ability to alter the existing state of the calendar schedule when it comes to the room they moderate.

3.6.2 Conflict Detection and Resolution

Additionally and more importantly the system should be able to detect upon the creation or editing of an event whether a conflict occurs with any existing bookings. If so it should separate these bookings and allow the moderator to resolve the conflict in any of the following ways:

- Separate conflicting bookings and present them to the moderator for resolution.
- Allow the moderator to choose from the following conflict resolution options:
    1. Keep any of the conflicting bookings and reject the rest.
    2. Edit conflicting bookings and move them to a different time slot or room to resolve the conflict.
    3. Utilize a combination of the above two methods for conflict resolution.

It is important to note that the system must be able to detect and inform of any conflicts that arise when the moderator edits the bookings so as to avoid the creation of even more conflicts.

3.6.3 Database Integration for Statistic Charts

An important addition to this functionality involves leveraging the system database to provide useful information to moderators about the frequency and state of bookings in their relevant rooms. This is achieved through dynamic statistic charts, which should provide the following information:

- Booking frequency of each hall based on a time range (e.g., hour of day, day of week, etc.).
- Frequency of the duration of bookings in a given room based on a time range.
- Occupancy rate for a hall in a range of time, calculated by the amount of time a hall is or is not available for booking compared to its total capacity for booking in a given time range.

These statistics should be presented with the option to dynamically change the sample in order to avoid and filter for extenuating circumstances such as periods with extreme amounts of activity or extreme lack there of. The information provided by these charts would allow the moderators to make informed decisions when it comes to managing the schedule, determine which rooms are over or under utilized and which time periods tend to be free so as to provide a convenient slot to place conflicting bookings.

3.6.4 Solution Conclusion

Thus in conclusion the system will holistically act as the solitary portal to inspecting the campus event schedule as far as it concerns the conference halls from the side of the attendees. It will act as the portal to request the reservation of the space of a campus conference hall from the side of the event organizer whether that be faculty or administration. And thirdly, it will act as a tool that the administration can use to both manage the space and time slots of campus conference halls as well as determine the state and efficiency of the system at any given time using the provided statistics.

Chapter 4

Database Breakdown

The database is an essential structural part of the system and had to be robust in its design so as to support the aforementioned functionalities as well as allow further improvements and additions that will be implemented in the future. The schema was designed following Laravel's Model-View-Controller architecture and MySQL was utilized to persistently store and manage the application's structured data. In general, this includes the following central entities:

- **Models**: Models represent the data and business logic of the application. In Laravel, models are PHP classes that typically correspond to database tables. They encapsulate data access and manipulation logic, making it easy to interact with the database [15].
- **Views**: Views are responsible for presenting data to the user. In Laravel, views are typically Blade templates that contain HTML mixed with PHP code [16].
- **Controllers**: Controllers handle the incoming requests, process the data, and return the appropriate response. They serve as an intermediary between models and views, coordinating the application's flow [17].

It is worth noting that names of models and controllers must fit Laravel's strict naming conventions to ensure normal functionality of the framework. In our case, the design relied heavily on the following models and relationships.

4.1 Models

- **User model**: The user model that holds all relevant information about users participating in the system. Its most significant contribution to the system's functionality is to identify and authenticate the users during their activity.
- **Role model**: The role model defined the various roles in the system and the privileges provided to them.

- **Semester model**: The semester model consists of a start and end attribute as well as a flag attribute to signify the one that is current. Functionally all newly created bookings are created in the current semester.
- **Room model**: The room model holds all associated information about the conference hall existing on campus. It holds an associated color attribute that is used for visual representation purposes in the calendar component.
- **Booking model**: The booking model represents the central object of all functionality and its more essential attributes include:
  - The triplet that comprises the date and time of any given booking (date, start, end).
  - The status which denotes the state of the booking as it pertains to the management's decision regarding it, i.e., An active booking is one that has been approved based on the management's ruling and thus will take place at the given time and place.
  - Publicity which determines whether the booking is visible to the public or just to the relevant participants.
  - Type which denotes whether the booking is a normal or a recurring one.
- **Recurring Model**: Essentially an instance of the booking model, this model exists for grouping and functional purposes. As will be explained in the following sections, recurring bookings are not created in the booking model until approved by a moderator in order to avoid overpopulating the schedule with illegitimate bookings. The recurring model holds all essential information for the bookings that will be created once it is approved.
- **Day model**: The day model is used to store the relevant days of the week for each recurring booking along with its start and end times.

4.2 Key Relationships

- **User-Role Relationship**: Users are associated with distinct roles governing their permissions and responsibilities within the system. This relationship enables role-based access control (RBAC), ensuring proper authorization and security.
- **Booking - Room**: Each booking belongs to a single room.
- **Booking - Semester**: Each booking belongs to the semester in which it was created.
- **Room-Building-Department**: Each room belongs to a building and a building can belong to many departments.
- **Recurring-Day**: Each recurring booking is associated with a number of days that mark the specific days of the week as well as the times that the recurring booking is taking place.

- **Booking-Recurring**: When a recurring booking is approved, the system automatically creates the corresponding normal booking throughout the semester and associates them together using a unique foreign id. This allows for the editing of a single recurring booking without interfering with the rest.

4.3 Database Schema

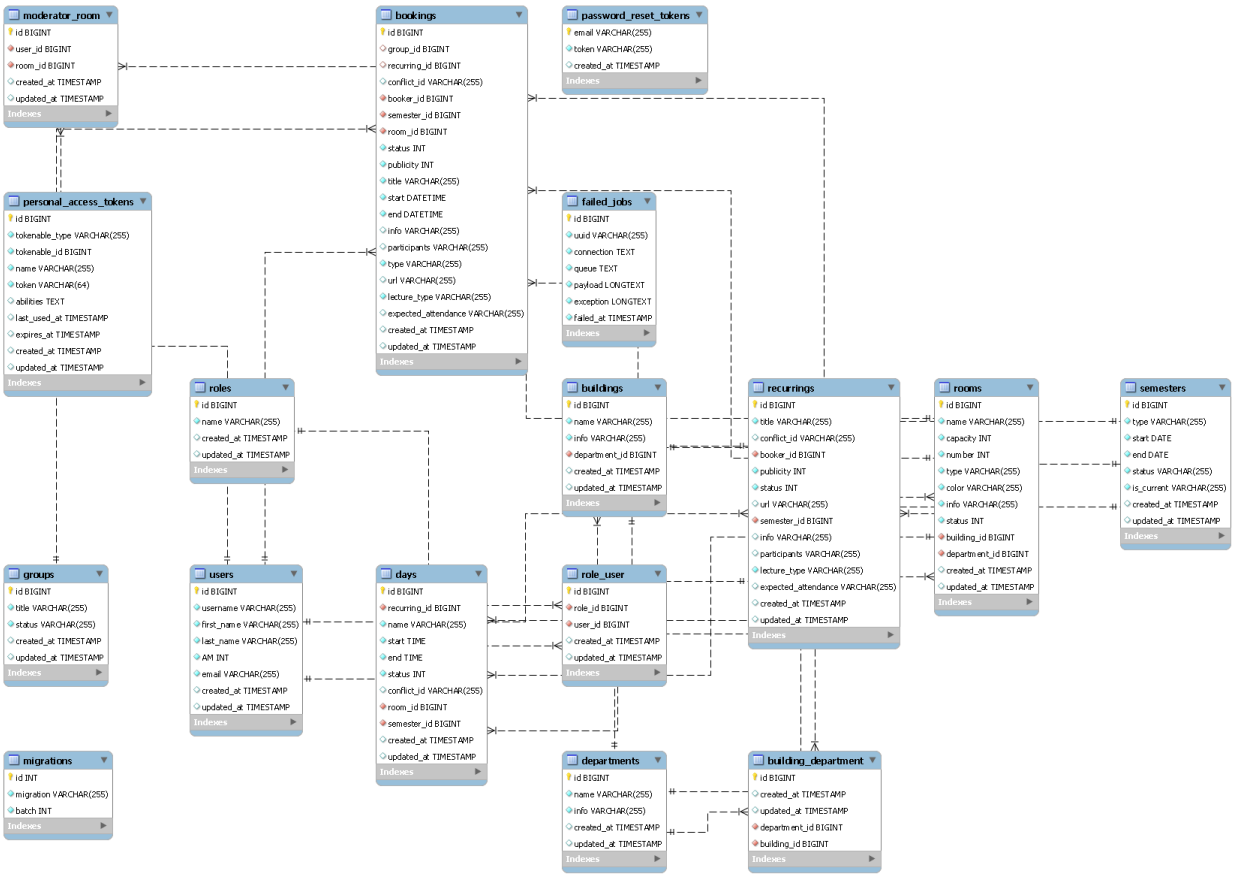


Figure 4.1: Database schema

Database Conclusion

In conclusion, the database design forms the backbone of the booking system, providing a comprehensive foundation for its functionality and future scalability. By adhering to Laravel's Model-View-Controller architecture, we have structured our database schema to seamlessly integrate with the application's logic and presentation layers.

The models encapsulate the essential data and business logic of the system, from user management to

booking management. Relationships between these models define the intricate connections within the system, enabling efficient data retrieval and manipulation.

Key entities such as users, roles, semesters, rooms, and bookings are modeled to accurately represent the real-world concepts they embody. Additionally, normalization techniques ensure data integrity and minimize redundancy, enhancing the database's efficiency and maintainability.

The defined relationships facilitate role-based access control, resource allocation, and management of recurring bookings, ensuring proper authorization and streamlined functionality.

Overall, our database design not only meets the current requirements of the booking system but also lays the groundwork for future enhancements and expansions. By adhering to best practices and leveraging Laravel's features, we have created a resilient and versatile database structure poised to support the system's evolution and growth.

Chapter 5

Front End

This chapter introduces the technologies and logic used in the construction of the front end part of the application. Beginning with a brief introduction to the frameworks and libraries used and continuing with an analysis of the core components and other noteworthy modules and functionalities. The purpose of this chapter is to explain the thought process and reasoning behind the design of the application as well as explain the benefits and constraints of the technologies used.

5.1 Technologies Used

5.1.1 Angular

Developed by Google, Angular is a design framework used to construct single-page applications. At its core, Angular is a component-based framework adhering to modular principles of constructing applications. In essence this means that Angular applications are primarily built using elements called components, which can be any unified and autonomous part of an application that can work independently or as part of an architecture tree with parent elements, descendants, and siblings. Typically, a component mainly consists of:

- The template, which is written in HTML and is responsible for what is rendered in the browser.
- The scripting that handles all the necessary actions for the function of the component, written in TypeScript.

Angular offers efficient and developer-friendly two-way data binding within and between components. Additionally, it provides a built-in router that handles navigation that adheres to the single-page principle. Sections of Angular applications are defined as routes, and the router is responsible of navigating between them. A route is essentially a component that contains all relevant children components for the particular section. It is structured in the same way as a regular component and is declared in

the application's routing module. In that module, route locations are defined along with the specific component that is loaded when the user navigates to said route. In order to share logic between components, Angular utilizes services which are defined by a TypeScript class and are used to inject code from a single source of truth.

After version 15, Angular introduced a new organizational parameter based on standalone components, contrasting the previously used module-based logic. This new pattern was followed for the booking service as it allowed the construction of more independent and scale-able components while avoiding the need to construct the underlying module architecture. In the old configuration, a module class would need to be constructed to handle all component dependencies while now this can be done in the component itself.

This application has benefited from the use of this framework as it not only offers a set of standardized configurations and libraries but also a productive and consistent way to structure code. This ensured a smooth and efficient developing process as well as allowing future scaling of the application.

5.1.2 Angular Material

Angular Material is a UI component library designed and developed by Google to equip Angular with pre-constructed general-purpose components. These components ensure consistency in both look and feel across the application. They follow the Material Design specification, which is Google's open-source design system. Angular Material also provides straightforward and extensive APIs for each component, something that allows vast configuration and customization to adapt each component as needed. Every basic element of this application was built using this library, some examples include:

- Forms
- Input fields
- Buttons
- Date pickers

The use of this library shortened development time and helped ensure a consistent user experience throughout the application.

5.1.3 Angular Calendar

The main calendar component used in the application is a pre-built package installed component developed by Matt Lewis. The component provides multiple views (month, week, day) as well as event functionalities for displaying and updating events. It uses the Date format configuration for these events and allows for extensive customization. The customization implemented for this application will be explained in following sections.

5.1.4 Chart JS

Chart JS is an open-source JavaScript-based library used to display dynamic charts and allows for customisation via a variety of options. It is compatible with all popular JavaScript frameworks including Angular. In this application, it was utilized to construct the dynamic statistical charts for the frequencies of bookings, and their function will be further explained in following sections.

5.2 Component Breakdown

This section will go in-depth about the development and function of some of the more structurally significant components of the application.

5.2.1 Routes

Beginning with the following, the application is sectioned into these routes:

- **Home Route:** Contains the calendar and all its functions including the booking request forms.
- **Moderator Dashboard:** Contains lists of all booking categories and conflicts along with pagination, sorting, and filters to ease the managing process. It is the main hub of function for the moderator of a room. It includes conflict resolution, all kinds of booking management, as well as the option to add an event.
- **Room Management:** This route contains functionality that allows the system administration to manipulate the state of the available rooms without having to manipulate the database directly.
- **Semester Management:** This route acts in the same way but for semester management. New semesters can be added or edited. Worth noting that this is where the current semester is defined.
- **User Booking Management:** This route is meant to allow the user to manage their own bookings prior to approval.
- **Statistics:** This route contains the general and the dynamic statistical charts that aid the moderator in managing the system.

5.2.2 General statistics

The general statistics that are included in the statistics route are the following:

1. Total bookings in the current month, semester, week.

Figure 5.1: General Statistics

2. The three busiest rooms this semester and this week along with the amount of bookings for each one.
3. Rate of booking approval along with the amount of bookings approved out of the total.
4. Capacity indicators for the current week and month.
5. The remaining time-slots in hours for the current week and month.
6. The mean booking duration for the current semester.

5.2.3 Moderator Dashboard Components

Before moving to the components, it is worth mentioning the Moderator Dashboard structure as it plays a role in many of the following components and vice versa. The dashboard is comprised of three tabs that contain separate components corresponding to different categories that are as follows:

- **Normal Bookings:** for all regular non-conflicting bookings.
- **Recurring Bookings:** for the recurring bookings displayed as groups. It is worth noting that when a recurring booking is approved, its individual bookings can be displayed in the normal bookings list but the collective group that refers to all of them is displayed in this list. This way either an individual can be manipulated or the entire collection.
- **Normal Conflicts:** a list of sets of conflicting bookings, separated into time-slots.
- **Recurring Conflicts:** a list of sets of recurring conflicts.

5.2.4 Card List

Another noteworthy clarification is that recurring bookings can conflict as a group when they, as a group, are stepping on the same days and hours but an individual booking of theirs conflicting with a normal booking will not cause a recurring conflict but rather a normal one.

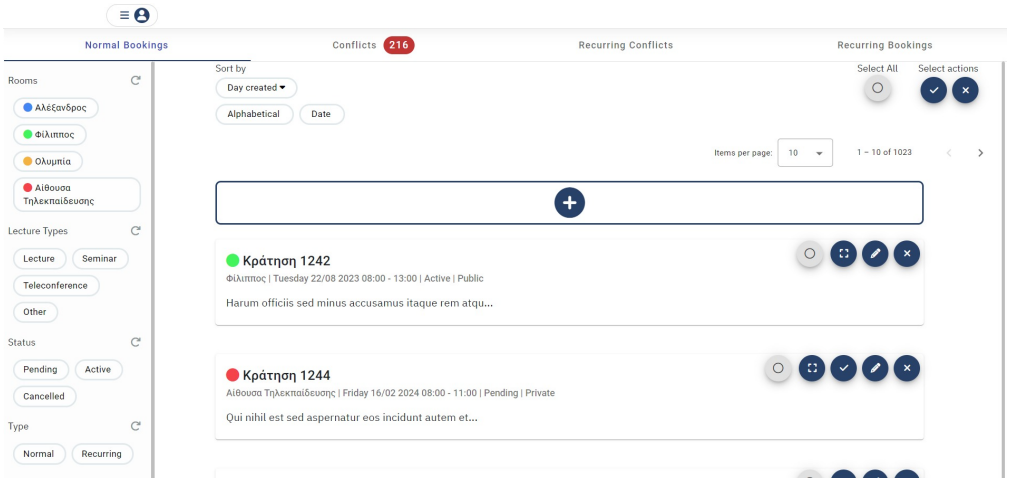


Figure 5.2: Moderator Dashboard

5.2.5 Calendar Component

In some part this component's event type informed the construction of this app's event structure. Even still some customisation was required in order to make this component match the particular needs of this application. By far the most significant customisation was configuring the calendar to fit the specified dimensions regardless of number of days in the month. Originally the calendar would dynamically generate the cells for each day of the month in the same dimension ignoring the total number of days in the month. This caused the calendar to increase in overall size depending on the number of days in the month. In order to resolve this a custom event binding function captures the month value when the scrolling of the calendar event is emitted. From there the number of days is calculated relative to the starting day of the week, this determines the amount of cells required to display the entire month. In order to set the overall height of the calendar to a consistent amount, the required height is divided by the cell count, in the style class for the calendar cells, using a variable.

5.2.6 Card List

Being one of the central components in the structure of the application, the card list is a custom component created using the Angular Material card component structure. Leveraging Angular's data binding mechanisms, it was constructed in a modular way. By providing the component with an iterable set of items that poses a correct configuration of properties (i.e., bookings, rooms, etc.), it is possible to display a list of multiple items quickly and effortlessly in any part of the application. The card list was used to display the lists of bookings in the Moderator Dashboard route, the rooms in the Room Management route, and even the semesters in the Semester Management route. It consists of a title along with an optional color icon, a subtitle that is used to display secondary information, and a set of buttons that are used for certain actions. The buttons emit events that are captured by the parent components and execute the actions as a response.

It is worth noting that one of these buttons can be a select toggle button used for multiple item control

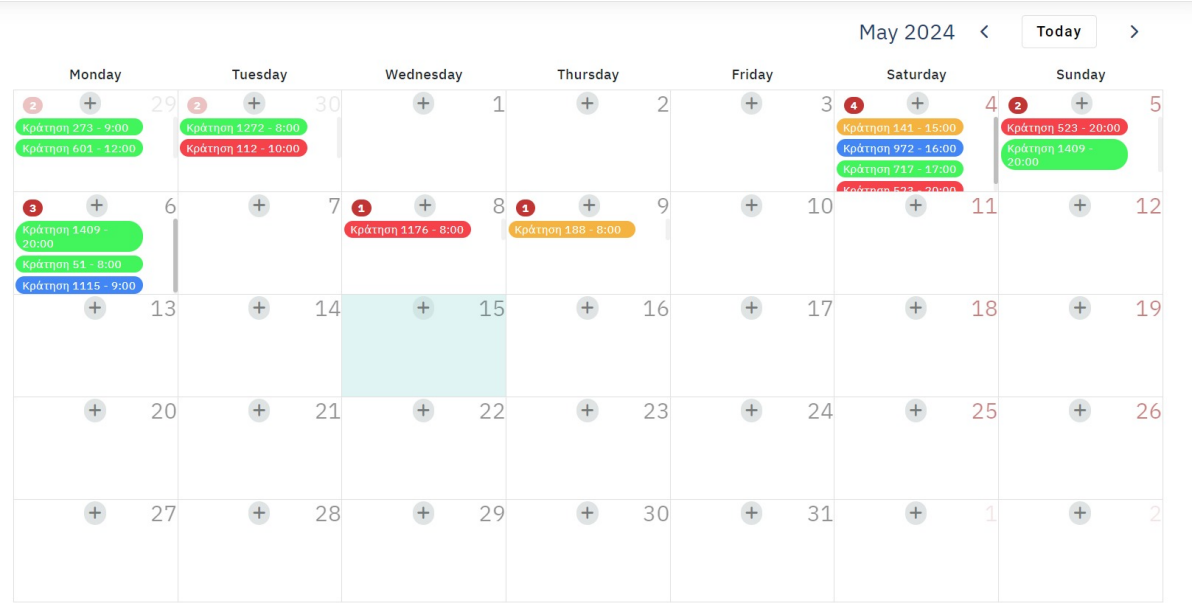


Figure 5.3: Calendar component

as will be explained in other components. The items are passed to this component from the parent and are bound, meaning reactive to changes applied to the set. This allows for dynamic adjustment of the information displayed on the screen. This also means that the component had to be designed in an independent way and is thus not responsible for any action. Essentially it is simply a container and event emission is used to inform the parent components of user action.

5.2.7 Conflict

The Conflict component is another custom component responsible for rendering the bookings that comprise a particular conflict. Because the number of conflicting bookings is not certain, an Angular Material expansion panel is used. This panel has an open and closed state allowing for a compact option when showing multiple conflicts and letting the moderator decide which one to tackle first by seeing a more holistic picture. In its open state, the panel consists of a card list of all conflicting bookings for the particular room and timeslot on a particular date. The bookings are colored according to the action that will be applied to them when the moderator presses the resolve conflict button, the configuration is as follows:

- Green: This booking will be kept no matter what.
- Red: This booking is conflicting and will be discarded.
- Blue: This booking has been edited and is no longer conflicting so it will be kept.

The logic behind this follows the following algorithmic steps. First, the first booking in the list is given the flag toKeep this signifies the green state mentioned above and is used in later checks. If the

moderator chooses to edit one of the rest, the new timing parameters are sent to the backend via the API and the system checks for conflicting bookings at that particular time slot. Provided that there are no conflicting bookings there, the booking is given the flag Resolved which is associated with the blue color mentioned above. When the moderator is happy with the state of the resulting set and they press the resolve button, the list of bookings is sent to the backend with all the edited changes and flags. The backend checks these flags and takes the necessary actions and updates the bookings accordingly.

5.2.8 Control Card

This custom component was constructed in order to give the option to the moderator to handle multiple bookings en masse. It consists of a select all toggle button that toggles the selected flag for all items in a list, a set of action buttons, and a set of pagination controls. Essentially this allows the moderator to choose how many items they see on the screen using the pagination controls, select multiple if not all of them and take the desired action en masse.

5.2.9 Filters

Another very structurally important part to the functionality of the application is the custom component Filters. The template consists of an Angular Material chip list which is a list of toggle-able buttons that correspond, in this case, to options passed to the backend API and are used to filter the content of the returning response. When a list of items is first loaded the front end sends an empty request to the appropriate endpoint. This tells the backend that no filtering is to be applied to the returning set, thus the entire list is returned. When a chip is pressed the request is sent anew with an appropriate set of options for all the selected chips. This allows for multiple option filtering and the backend simply filters for all the requested options. When the set is returned it is dynamically updated in the child component utilizing Angular's data binding mechanisms.

A complete list of the filters is as follows:

- Room filter: filters by room, the list is retrieved from the API depending on the intended use. For example, the rooms for moderation should only include the ones the particular moderator moderates.
- Status filter: filters by booking status, active, pending, or canceled.
- Type filter: filters by booking type, normal, or part of a recurring group.
- Publicity filter: filters by booking publicity, which is whether the booking is available to the public.
- Months filter: filters by month of year.
- Days filter: filters by day of the week.
- Lecture type filter: filters by lecture type.

5.2.10 Statistic

The statistic is a custom component used to display the dynamic statistical charts. It uses Chart.js which utilizes the canvas HTML element to render various chart types. The component contains a main menu control that selects the statistic type. This component is dynamically generated in the Statistics route with a press of a button. It is possible to generate multiple statistics on screen, allowing for a more complete view of the desired information. Adding to this is the option to compare statistics between rooms on the same chart. These two options allow for the ability to no only compare rooms but also sample sizes and different options.

A complete list of the available statistics is as follows:

- Hour of Day - Frequency: the amount of bookings for a particular room each hour of the specified day of the week.
- Day of Week - Frequency: the amount of bookings for a particular room for each day of the week.
- Day of Month - Frequency: the amount of bookings for a particular room for each day of the specified month.
- Month of Semester - Frequency: the amount of bookings for a particular room for each month of the specified semester.
- Day of Week Duration - Frequency: the amount of bookings grouped by their duration for a particular day of the week.
- Month Duration - Frequency: the amount of bookings grouped by their duration for a particular month.
- Occupancy by Day of Week: the amount of hours a room has been occupied on a given day of the week.
- Occupancy by Month: the amount of hours a room has been occupied in a given month.
- Occupancy by Semester: the amount of hours a room has been occupied in a given semester.
- Occupancy by Date Range: the amount of hours a room has been occupied in a given date range.
- Date Range - Frequency: the amount of bookings for a particular room for each day of the specified date range.
- Date Range Duration - Frequency: the amount of bookings grouped by their duration for a given date range.

In addition, it contains various menus with necessary options for each particular statistic. These include:

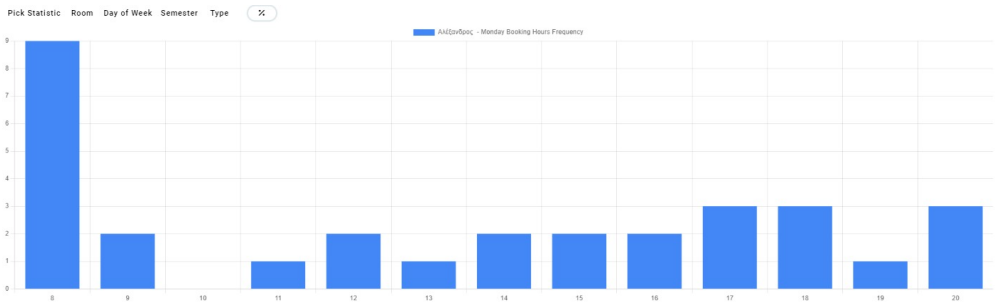


Figure 5.4: Frequency Statistic

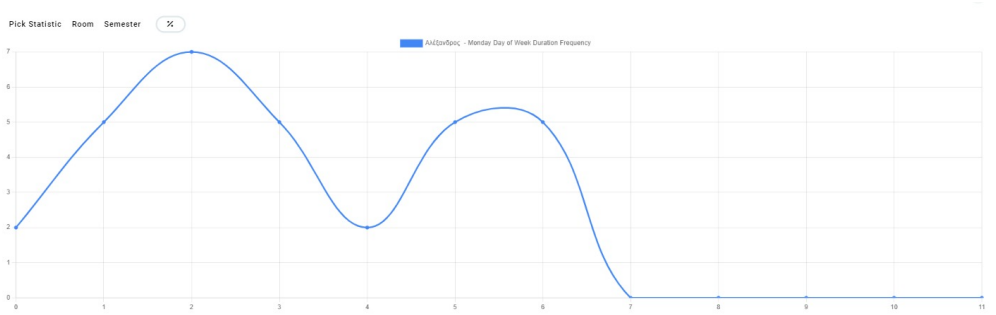


Figure 5.5: Duration Frequency Statistic

- Room picker: contains a list of rooms. Multiple ones can be selected, and each will create a separate dataset to be displayed in the chart.
- Day picker: a list with each day of the week. Multiple ones can be selected which will increase the range of the query to apply all the selected days.
- Month picker: a list with each month of the year. Multiple ones can be selected which will increase the range of the query to apply all the selected days.
- Semester picker: a list of all semesters contained in the system. Multiple ones can be selected, and doing so will essentially increase the sample for each statistic allowing for either a more holistic or constrained view of the statistic.
- Date picker: an Angular Material date picker that allows the selection of a date range. This is used in the date range statistics.
- Type picker: a list of all available booking types allowing the filtration of the statistics by each type specifically.

5.2.11 Dialogs

Dialogs are modal type components of Angular Material. They appear over the rendered page and act on their own independently until closed. In the context of this application they are used to inform,

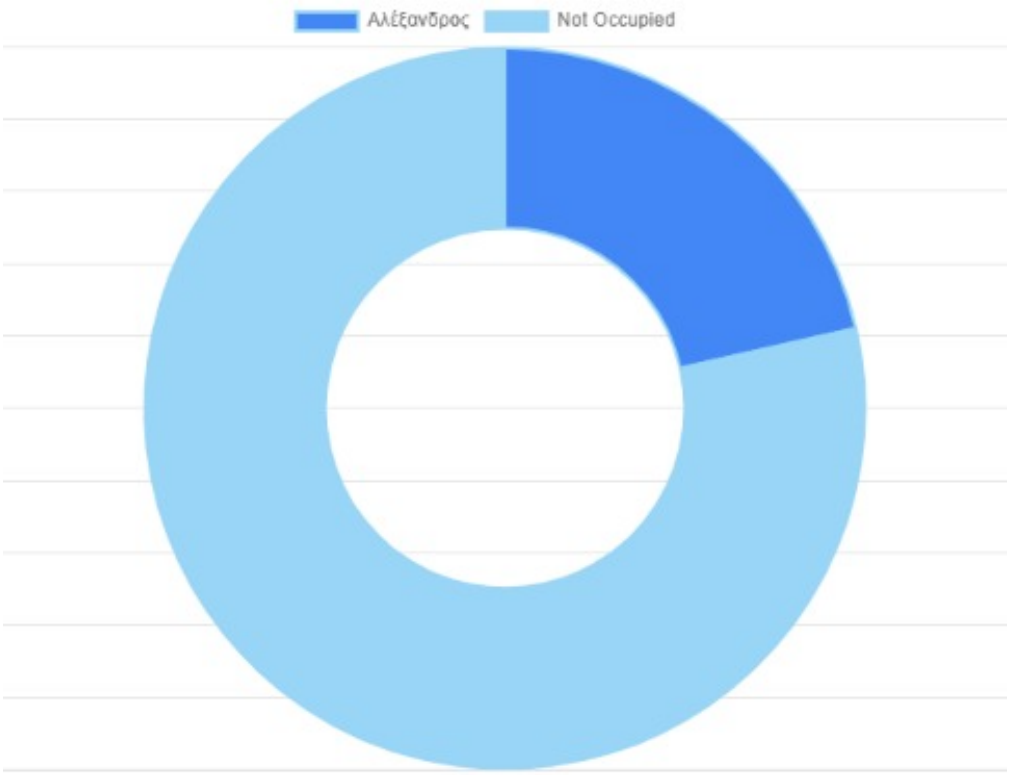


Figure 5.6: Occupancy Statistic

get consent to actions, create and update items. The most noteworthy of the dialog components is the Booking Form Dialog. This is a custom draggable dialog that contains a form used to either create or update bookings. When opened from the calendar the dialog is initialized without data and thus the form is empty. When opened from the moderator dashboard to edit a booking, the booking is passed to the dialog and its data fills the form accordingly. Upon submission the dialog closes and the booking is created or updated accordingly by sending the appropriate request to the API. Analogous dialogs exist for rooms and semesters while all the rest are for displaying messages or asking the user to confirm an action.

5.3 Services

Services in the Angular framework are responsible for exchanging data or code between components. Each service is called in the constructor part of the component class. A service can have multiple functions and it acts as a gate to all of them when called.

In this application, the following services exist:

- **Booking Service:** Responsible for retrieving all relevant booking data while also submitting requests that alter the data in the database. In addition to all the functions that communicate with the API, it also includes a mapping function that maps the returning bookings so they are able to be inserted into the calendar component.
- **Dialog Service:** Used to open all the dialog components for the application. It has a separate function for each dialog, and these functions set the dialog data, size, and any other necessary attributes.
- **Filter Service:** Retrieves all necessary parts for the operation of the filters. By providing it with a list of the desired filters, it returns a list with the labels and action calls for each one of the filters.
- **General Service:** Reserved for all necessary communication with the API that doesn't strictly fit into any of the other services.
- **Room Service:** Similar to the booking service, this one is responsible for all relevant data exchange relating to rooms.
- **Sidebar Service:** Responsible for retrieving the items that fill the sidebar.
- **Statistic Service:** Has a double function. On one hand, it retrieves all the statistics with their associated picker options, and on the other, it handles all the back and forth to retrieve the statistic datasets from the API.

Chapter 6

Back End

As mentioned above, Laravel is a prominent PHP web framework renowned for its adherence to the Model-View-Controller (MVC) architectural paradigm. It was used to build the backend system for this application as it offers a comprehensive suite of tools and functionalities essential for crafting sophisticated web applications. In general, typical Laravel project architecture was followed with the bulk of development being in the construction of an API to support database manipulation, conflict detection, and the generation of statistical data using MySQL queries.

6.1 Architecture Overview

Following MVC principles, the application's backend is constructed mainly by using the following parts:

- **Routes:** Used for mapping URLs to controller actions. These routes direct incoming requests to the appropriate controllers which then execute the relevant actions.
- **Controllers:** Controllers contain the application's logic for processing requests and generating responses. Controllers interact with models to retrieve or manipulate data and return appropriate responses to the frontend for presentation.
- **Models:** Models represent the data structure of the application, encapsulating database tables and their relationships. Models interact with the database through Laravel's Eloquent ORM (Object-Relational Mapping), which is an intuitive interface for performing CRUD (Create, Read, Update, Delete) operations.
- **Database Migrations and Seeders:** Laravel's migration and seeding system is used to manage database schema manipulation and database data initialization. Migrations were used to define the database schema using PHP, while seeders were used to insert test and base data into the database.

6.2 API Overview

In order to construct the application, API endpoints were defined as routes. The frontend part of the application handles user action and data retrieval through these endpoints. All API endpoints are defined in the `web.php` file to allow access via HTTP, and the following is a list of the available endpoints and the actions they perform:

6.2.1 Booking Associated Endpoints

- **/getBookings:** Retrieves and filters bookings based on various parameters. This route is used to retrieve bookings that are displayed in the moderator dashboard.
- **/getRecurring:** Retrieves and filters recurring booking groups based on various parameters.
- **/getConflicts:** Retrieves and filters conflict groups based on various parameters. The groups are checked for and created on the spot, the logic for this is explained further below.
- **/getRecurringConflicts:** Retrieves and filters recurring conflict groups based on various parameters.
- **/checkConflict:** Checks for conflicts when editing a booking for conflict resolution. A similar process is followed when creating a booking to assign a conflict id. The logic will be explained below.
- **/resolveConflict:** Used to resolve conflicts between bookings based on various parameters, such as rescheduling or canceling conflicting bookings.
- **/resolveRecurringConflict:** As above but for recurring booking groups.
- **/getActiveBookings:** Retrieves active bookings from the database. This route is used to retrieve upcoming or ongoing public bookings for the calendar component to display.
- **/getUserBookings:** Retrieves bookings associated with a specific user.
- **/createBooking:** Creates a new booking in the database.
- **/approveBooking:** Approves a pending booking request.
- **/cancelBooking:** Cancels a booking.
- **/editBooking:** Edits an existing booking.

6.2.2 Room Associated Endpoints

- **/getRooms:** Retrieves rooms from the database based on various parameters.
- **/getDepartments:** Retrieves departments.
- **/getBuildings:** Retrieves buildings.
- **/getAllRooms:** Retrieves all rooms from the database.
- **/createRoom:** Creates a new room in the database.
- **/deleteRoom:** Deletes a room from the database.
- **/editRoom:** Edits an existing room.
- **/getModeratedRooms/id:** Retrieves rooms moderated by a specific user.
- **/getPossibleModerators/:** Retrieves possible moderators for rooms.
- **/getAllSemesters:** Retrieves all semesters from the database.

6.2.3 Semester Associated Endpoints

- **/createSemester:** Creates a new semester in the database.
- **/updateSemester:** Updates an existing semester.
- **/deleteSemester:** Deletes a semester from the database.

6.2.4 Statistic Associated Endpoints

- **/roomHourOfDayOfWeekFrequency:** Retrieves statistics on room booking frequency by hour of a specific day of the week.
- **/roomDayOfWeekFrequency:** Retrieves statistics on room booking frequency by day of the week.
- **/roomDayOfMonthFrequency:** Retrieves statistics on room booking frequency by day of the month for a specific month.
- **/roomMonthOfSemesterFrequency:** Retrieves statistics on room booking frequency by months of a given semester.
- **/roomDateRangeFrequency:** Retrieves statistics on room booking frequency within a specific date range.

- **/roomDayOfWeekDurationFrequency:** Retrieves statistics on room booking duration frequency by day of the week.
- **/roomMonthDurationFrequency:** Retrieves statistics on room booking duration frequency by month.
- **/roomDateRangeDurationFrequency:** Retrieves statistics on room booking duration frequency within a specific date range.
- **/roomOccupancyByDayOfWeekPercentage:** Retrieves statistics on room occupancy by day of the week.
- **/roomOccupancyByMonthPercentage:** Retrieves statistics on room occupancy by month.
- **/roomOccupancyBySemester:** Retrieves statistics on room occupancy by semester.
- **/roomOccupancyByDateRange:** Retrieves statistics on room occupancy within a specific date range.
- **/generalStatistics:** Retrieves all general statistics about bookings in the application. These statistics are used in the statistics dashboard.
- **/getOccupancyCharts:** Retrieves occupancy charts for visualization in the statistics dashboard.
- **/bookingTotals:** Retrieves total booking counts for visualization in the statistics dashboard.
- **/approvalRate:** Retrieves the approval rate of bookings for visualization in the statistics dashboard.
- **/meanDuration:** Retrieves the mean duration of bookings for visualization in the statistics dashboard.

6.3 Booking Controller

The vast majority of the applications logic for managing the booking system exist within this controller in the form of functions, each one responsible for one action in the system. Their broad categories are split between normal, recurring and conflicting bookings. These categories are generally separated on whether the action is about retrieving data, manipulating the database in some way or some other miscellaneous action. Since these functions comprise the core of the application their logic will be analyzed in depth in this section.

6.3.1 Filtering and Pagination

Since almost every function in this controller utilizes filtering and pagination to limit the responding data within a particular set we should start by explaining how this is implemented in a broad sense.

Every function is accessed through an endpoint via a POST HTTP request, carrying a payload in said request that contains all the necessary information about how to filter the corresponding response. This information is carried in the form of an array of objects, with a key index specifying the specific filter to be applied and an associated array of values to filter by. Using an array for the values allows for the use of Laravel's Eloquent mechanism to filter between multiple values. For example, to filter the bookings for two specific rooms the ids for those specific rooms would be passed in the room ids array to be used in the whereIn part of the query that retrieves the bookings. Typically all filter arrays are filled with a default value if a value is not provided in the request. This default value is usually a set of all possible values that effectively means that the response is not filtered by that particular filter category. As for the pagination part of this process, Laravel has an integrated mechanism for applying pagination to a set of returning entries. Every request carries a set of two values that are relevant to that pagination. The page index that signifies the page that needs to be retrieved and the amount of entries each page should have. It is important to note that each time a filter is updated these values are reset so as to retain consistency in the pagination of the results without missing any of the entries between pages.

Table 6.1: Booking Controller Function Filter Support

Functions	Filters						
	room	lecture type	months	days	type	publicity	status
index	✓	✓	✓	✓	✓	✓	✓
getRecurring	✓	✗	✗	✓	✗	✓	✓
getUserBookings	✓	✗	✗	✗	✗	✗	✗
getActiveBookings	✓	✗	✗	✗	✗	✗	✗
getConflicts	✓	✗	✓	✓	✗	✗	✗
getRecurringConflicts	✓	✗	✗	✓	✗	✗	✗

6.3.2 Index Functions

Controllers typically correspond to their synonym model and the index function is normally responsible for retrieving the entries of that particular model from the database. In the case of this application multiple get functions were implemented within the same controller since they belonged in the same semantic category. All these functions typically follow this structure:

1. Set filter and pagination values to requested or default values.
2. Query the Model for the desired entries.
3. Map the resulting set into the correct form.
4. Return the response.

6.3.3 Get Conflicts

As the conflicts pose a particularly important part of the application each action concerning them should be analyzed. For the retrieval of normal conflicts the first step remains the same as the other get functions. Afterwards all bookings that match the specified filters are retrieved and grouped by their conflict id. Bookings with the same booking id are conflicting with each other, this id is applied to them during the creation of one of those bookings. This process will be explained in detail later. Once the conflicting bookings are retrieved they are mapped into grouped object arrays based on their conflict id. Finally they are returned as the response, this process is the same for both normal and recurring bookings .

6.3.4 Store Functions And Conflict Generation

In this subsection we present the process that is followed on the creation of a new booking as well as the detection and generation of conflicts for both normal and recurring bookings. Initially the same function is triggered for both types and a flag is used in the case of the recurring. Both functions begin by validating the inserted data, ensuring data integrity. Then an entry is created using the corresponding eloquent model. In the case of normal bookings on the creation a scope constraint is defined that triggers when creating a new booking. This constraint is responsible for scanning the database for conflicting bookings. It retrieves all bookings for the particular room that are between the start and end times of the same date as the booking that is being created. If it finds any it checks for an existing conflict id and applies to the new booking. If there are conflicting bookings that do not have an existing conflict id then a new one is generated and applied to all. This is the case when two bookings are not conflicting with each other but a third one is inserted in between and conflicts with both. In this case all three bookings are grouped together. In the case of the recurring booking a similar constraint is applied but it goes through the Day model in order to detect conflicts. The function that is responsible for creating a recurring booking also creates the corresponding entries in the Day model. This model holds the room, day name, start and end values for each day of the week that the recurring booking will take place. This was necessary to allow recurring bookings to take place in separate rooms during the week. The constraint function retrieves the days of the recurring booking that is being created and checks their hours against those of existing days of the week that are for the same room. If it finds conflicting days it marks the recurring bookings with either a new or existing conflict id. Note that these constraints are also triggered in the case of editing to detect and generate conflicts.

6.3.5 Approve Recurring Booking

The case of approving a recurring booking also bears a brief explanation as it is a special case and functions in a more complex way than one might think at face value. Recurring bookings are set separately from normal bookings until they are approved. As explained above the Day model holds the information about the occurrences of the recurring booking and there they exist alone until approval. When a moderator approves a recurring booking the function in question iterates through the days

associated with the booking and creates a regular booking for each week of the current semester on that particular day and time. Of course the normal conflict detection and generation constraints still apply for these bookings and any newly created conflicts that include them are treated as normal with each of them as individual bookings.

6.3.6 Check Conflict

This function is used to scan existing bookings for conflicts with a date and time, prior to resolving a conflict. When a moderator wants to resolve a conflict they have the option of editing any of the conflicting bookings in order to achieve this. In order to inform the moderator if the choice they made when editing is a valid one, meaning it doesn't create any new conflicts, the system gives the edited date and time to this function. Essentially this function then runs the relevant queries to detect whether or not there are bookings on that date and time in that particular room. In either case it returns a flag with the appropriate response and the booking is marked in the front end accordingly informing the moderator of the validity of their choice.

6.4 Statistics Controller

This controller contains all functions that are responsible for querying the database to retrieve the necessary datasets for the application's statistics dashboard. In this section, the logic behind these statistical functions will be broken down and analyzed. The basic idea behind these functions is to assist in determining the state and rate of efficiency in the system and how well the hall resources are being utilized, as well as informing management decisions concerning the forming of the schedule. In general, the sample is set by default to the current semester with the option to include any and all of the stored semesters as a sample. This allows the information to remain flexible and able to meet changes in the state of the system as well as expand or narrow its scope. Adding to this, the endpoints return separate datasets for each room provided, to be used in the same charts to compare between. There is also the option to filter by booking time in order to limit the information to only include the relevant bookings. Finally, all datasets include a form of the data as a percentage of the total depending on the sample which can be toggled between on the front end using a button.

6.4.1 Booking Frequency

Separated by unit of time, these functions return datasets concerning how many bookings occurred by room on a specific range of time. They are the following:

- `roomHourOfDayOfWeekFrequency`
- `roomDayOfWeekFrequency`
- `roomDayOfMonthFrequency`

- `roomMonthOfSemesterFrequency`
- `roomDateRangeFrequency`

These functions utilize options that can change the range they examine, for example, which months or which day and multiple ones can be selected at once. They start by retrieving the desired options from the request, then they iterate through each of the wanted rooms in order to create a separate dataset for each one. Afterwards, depending on the type of time range each function is for, relevant labels are constructed and are placed into an array. For example, the `roomHourOfDayOfWeekFrequency` function creates a label for each working hour in a day. Then the total amount of relevant bookings is calculated for the given sample which is used to calculate the percentage of each frequency. Afterwards, a query is constructed that utilizes the Illuminate library to select the unit of time that is measured from the start column of the Booking model and to count the amount of bookings for each fraction of the unit. The query also filters for lecture type, sample, and approved bookings. Finally, the query groups by the selected unit range. Then the frequency and its percentage representation is mapped into arrays, making sure that the index of these arrays correspond to the correct index of the labels array. Some more mapping is done for organizational purposes and the process is repeated for each room. Finally, the result containing all said datasets is returned.

6.4.2 Booking Duration Frequency

Similar to the above, these functions concern the frequency of the duration of bookings by room on a specific time range. They sample the current semester by default and in general support all the options that the above functions support as well. They are the following:

- `roomDayOfWeekDurationFrequency`
- `roomMonthDurationFrequency`
- `roomDateRangeDurationFrequency`

The process these follow is essentially the same as the above but they calculate the duration of each booking using the Illuminate library again and utilizing the start and end columns of the Booking model. The rest stays the same and the returned datasets are again separated by room and they are used in line charts on the front end.

6.4.3 Room Occupancy

These functions are used to calculate the amount of time a room is occupied in a given time range. They support the cumulative addition of multiple rooms allowing for the direct comparison of these rates on the same chart. They also use the current semester as the default sample while allowing similar options as the functions above. They are the following:

- `roomOccupancyByDayOfWeekPercentage`
- `roomOccupancyByMonthPercentage`
- `roomOccupancyBySemester`
- `roomOccupancyByDateRange`

They begin by retrieving the relevant options from the request and then begin the iteration through the specified rooms. Then they calculate the total amount of time slots that exist in the specified time range and sample in order to calculate the percentage. The total essentially increases when rooms or more time options are added. The relevant bookings are then retrieved and counted. Finally, after some organizational mapping, the resulting datasets are returned to be used on pie charts on the front end.

6.4.4 General Statistics

The following functions are used to construct the general statistics that exist on the top of the statistics dashboard. Their purpose is to provide some general information about the state and performance of the halls and the system itself. These are these functions and the logic behind their result.

- `bookingTotals`: This function calculates the total amount of bookings in the current semester, month, and week. It uses Carbon, an API extension of DateTime to calculate the current month and week and retrieve the bookings that fall within those. The semester is retrieved from the Semester model and using its id the relevant bookings are retrieved easily.
- `approvalRate`: This function returns the number of approved and canceled bookings on the semester and status. It begins by retrieving the current semester and the total number of all the bookings in said semester. Then the counts for the bookings in said semester that have been approved and canceled are calculated and the percentage rate is calculated using the total. Finally, the results are mapped and returned.
- `meanDuration`: This function calculates the average duration of bookings within the current semester. It uses the AVG and TIMESTAMPDIFF function in MySQL to calculate the average duration of bookings in the semester and returns that number.
- `bussiestRooms` and `bussiestRoomThisWeek`: The `bussiestRooms` function uses the familiar process to calculate the frequency of bookings in the current semester for all rooms. It then groups them by room and orders them by that frequency in descending order. Then it takes three of them, meaning the three busiest rooms this semester by amount of bookings. Finally, the `bussiestRoomThisWeek` function is the same as the above but it only returns the single most busy room for the current week.
- `weekCapacityIndicator` and `monthCapacityIndicator`: These functions calculate the remaining hours in the current week and month, adjust them based on existing bookings,

computes a capacity indicator percentage, and returns an array with the capacity indicator, the remaining hours in the week, and the dividend percentage. Essentially, they return the amount of time slots that are still available for the current month and week.

- `generalStatistics`: This function collects all general statistics in order for the system to be able to retrieve them collectively from a single endpoint.
- `getOccupancyCharts`: This function utilizes the previously defined occupancy functions to calculate the occupancy rate for the current week, month, and semester for all rooms. These are used in the top right charts of the statistics dashboard.

Chapter 7

Conclusion

The objective of this thesis was to develop and evaluate a digital booking platform tailored for managing campus hall reservations in a university setting. This platform was designed to address the inefficiencies inherent in traditional booking systems, primarily reliant on manual processes such as Excel spreadsheets, which are prone to errors and operational challenges. The digital solution proposed in this work not only sought to overcome these shortcomings but also aimed to enhance the administrative operations within educational institutions through improved resource management.

The development of the platform was grounded in a thorough analysis of both front-end and back-end technologies, culminating in the selection of Angular for the front-end and Laravel for the back-end. This choice was justified through a comparative study of technology frameworks, evaluating criteria such as community support, ease of migration, performance, and long-term maintainability. Angular was selected for its robust structure and reliable upgrade path, which align with the long-term strategic goals of university administration. Similarly, Laravel was chosen for its scalability and efficiency in managing complex queries and data integrity, which are crucial for the backend operations of a booking system.

The system was designed to feature two main functionalities: user functionality that allows faculty and staff to efficiently manage their event scheduling and reservations, and management functionality that enables administrative oversight and conflict resolution. This dual functionality ensures that the platform not only supports everyday users in their operational needs but also provides administrators with the tools necessary to oversee and optimize the use of resources.

The implementation of this system was anticipated to significantly reduce the time and effort required to manage bookings, mitigate errors due to manual entry, and enhance the overall user experience through an intuitive and user-friendly interface. The platform's capability to handle conflicts in scheduling and its robust statistical tools further assist administrators in making informed decisions that optimize resource allocation and usage.

Despite these advancements, the study acknowledges several limitations. The scope of the technology evaluation was confined to popular frameworks, potentially overlooking emerging technologies that might offer additional benefits. The system's performance was also evaluated in a controlled setting,

which might not fully capture the complexities and unpredictable user behaviors in a live environment. Additionally, while the system is designed for scalability, actual deployment could reveal unforeseen challenges that could affect system performance or user satisfaction.

7.1 Future Research

For future research, it would be beneficial to explore the integration of more advanced technologies such as artificial intelligence and machine learning to predict peak times and suggest optimal booking periods. Further studies could also assess the user adoption rate across different departments, identifying specific barriers to uptake and developing strategies to enhance user engagement with the platform. Lastly, a longitudinal study on the platform's impact on administrative efficiency and resource utilization could provide deeper insights into the long-term benefits and areas for improvement in digital booking systems within university settings.

In conclusion, this thesis contributes to the field of educational administration by demonstrating the potential of digital solutions to enhance the efficiency and effectiveness of university operations. It lays the groundwork for future innovations in digital administration, encouraging continued exploration and development in this vital area.

Appendix A

User Tutorials

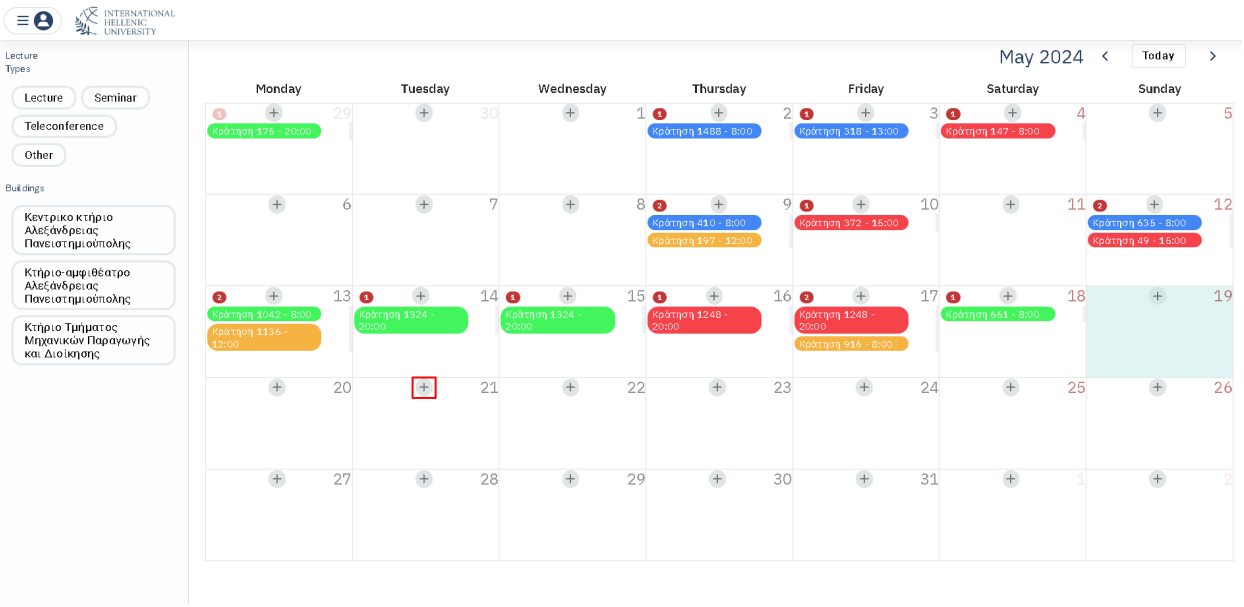
This appendix provides a brief tutorial for creating and editing bookings step by step from the point of view of a regular user. The basic tutorial will cover the creation of a regular booking while the rest of the sections will cover the creation of recurring bookings as well as the editing of any booking created by the user.

A.1 links

- Front-end: dim2409/Booking-Service
- Back-end: Ioatsi/BookingServiceBackend
- Live: booking.iee.ihu.gr

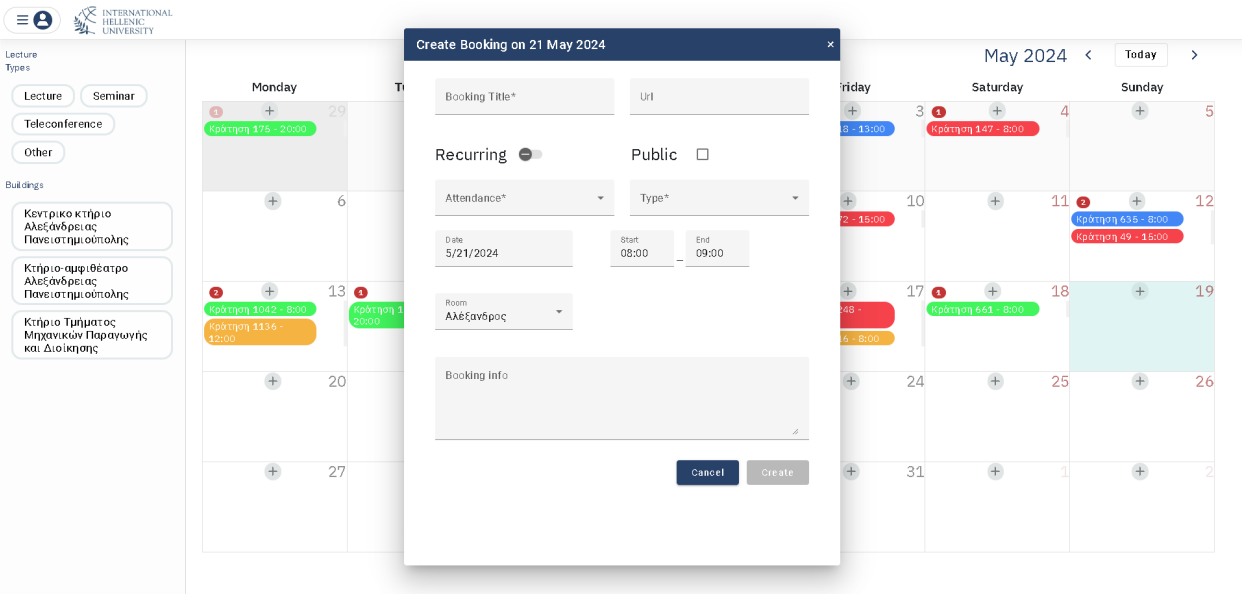
A.2 Basic Tutorial

A.2.1 Step 1: The homepage



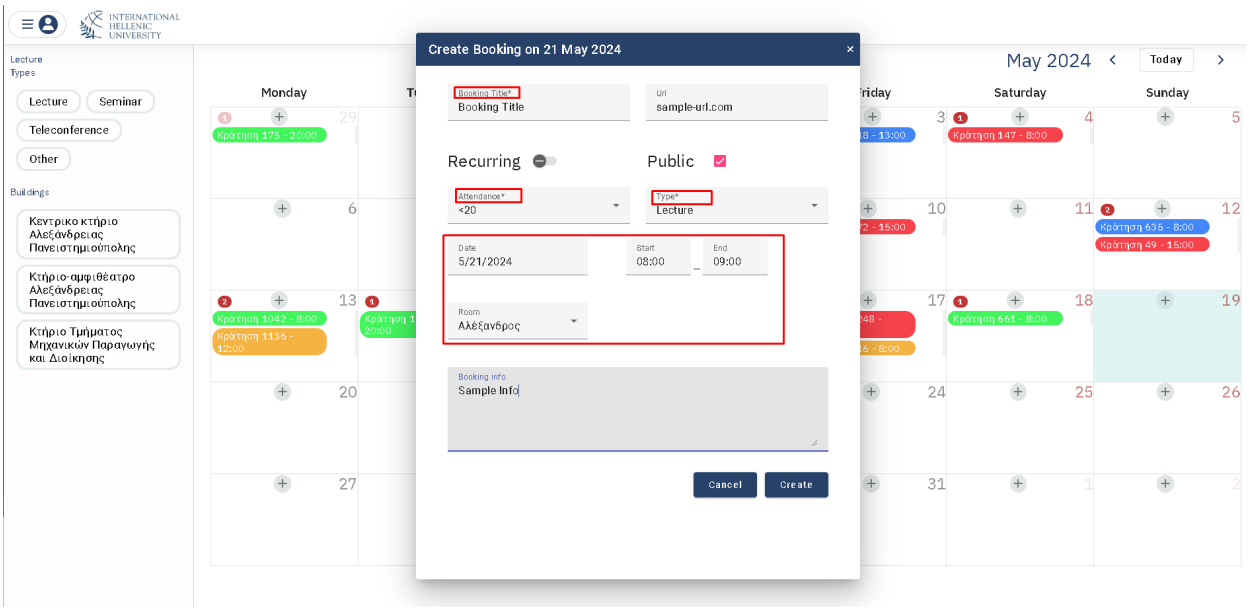
From the homepage calendar you can navigate and pick the date at which you want to create a booking and by pressing the + button on that specific date.

A.2.2 Step 2: The booking creation form



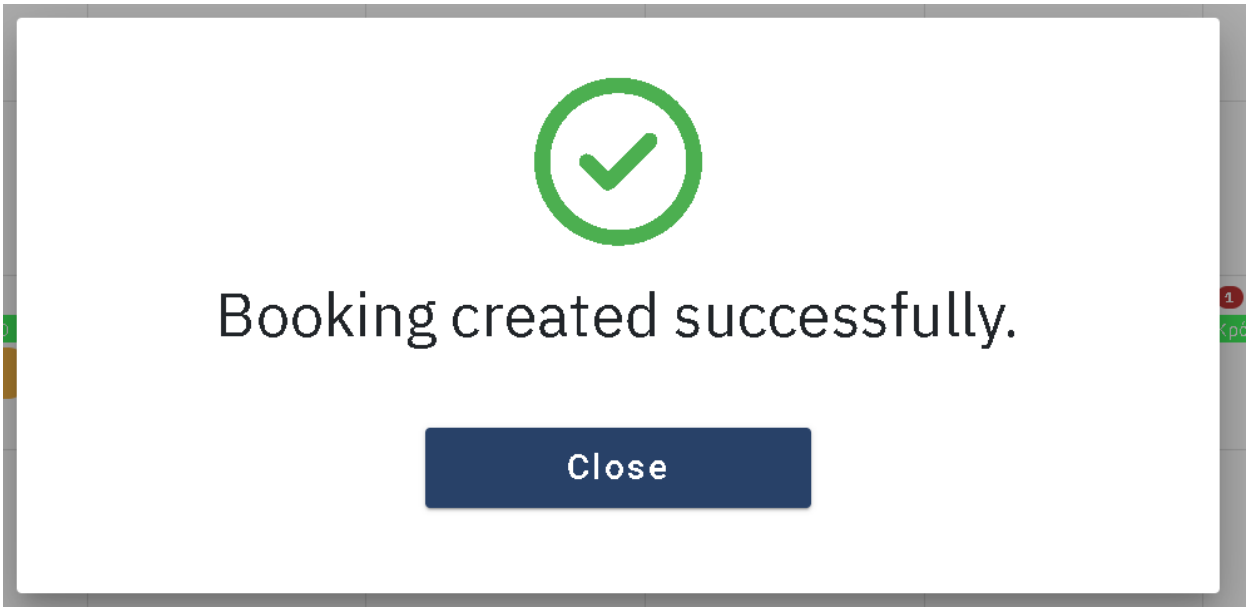
After pressing the + button a dragable dialog appears, you can notice the date field is already loaded with the date selected form the previous step.

A.2.3 Step 3: Filling the fields



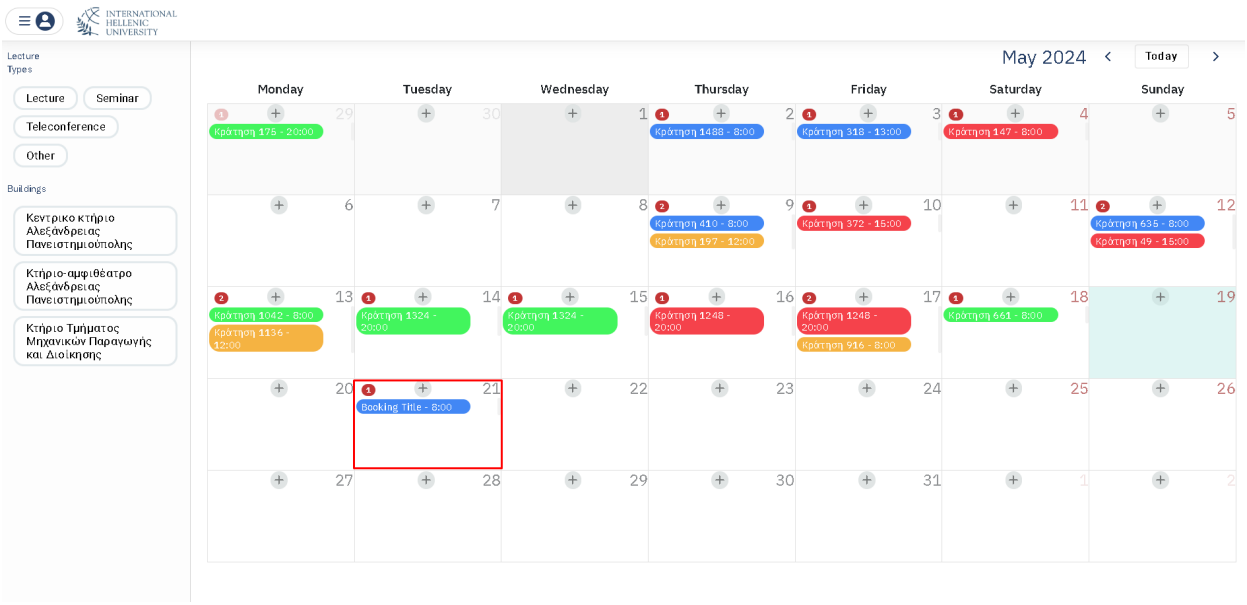
The marked fields are the ones required to create a valid booking. If these fields are not filled a booking will not be created. The rest of the fields can remain empty. After all necessary fields are filled you can press the create button to create the booking.

A.2.4 Step 4: The Booking has been created



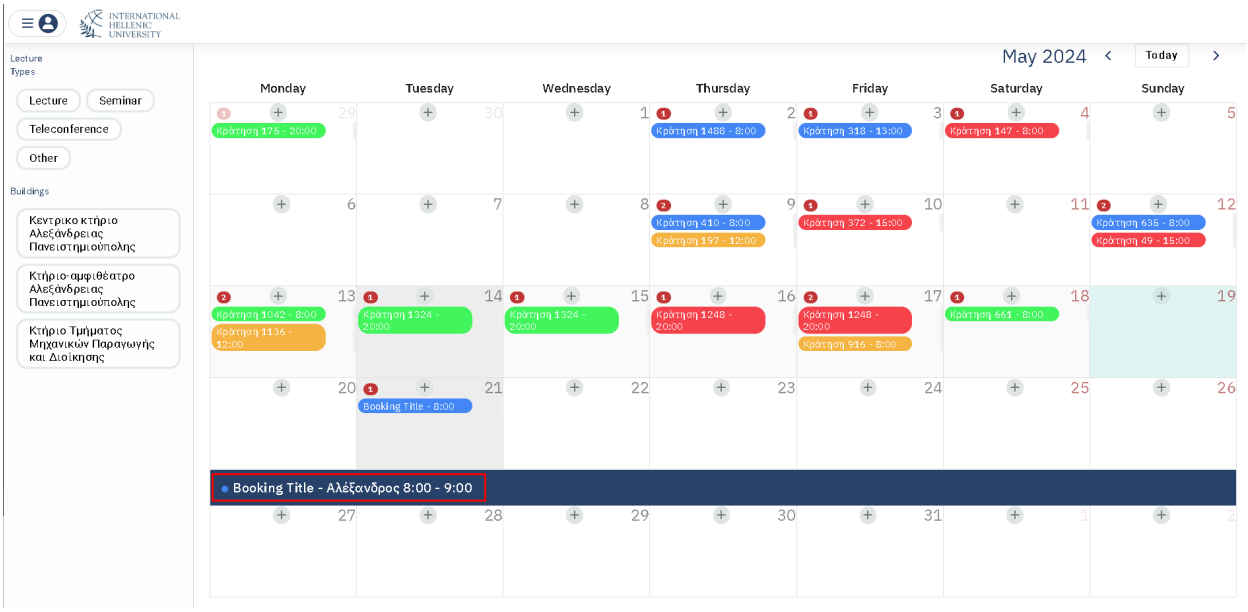
Provided all the information was valid the system will respond with this success dialog informing you that the booking has been created.

A.2.5 Step 5: Inspecting the created booking



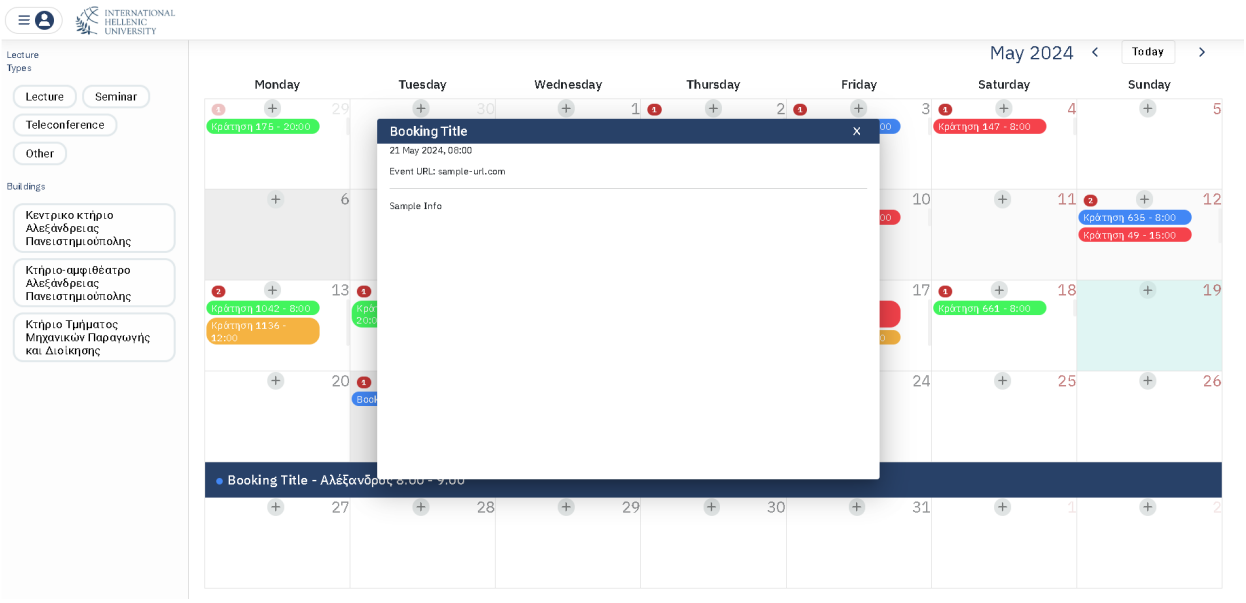
Afterwards provided the booking was marked public and is not conflicting with an existing booking, it will be visible in the homepage calendar.

A.2.6 Step 6: Opening day cell view



By pressing on the day cell a section opens up that displays the title and hour of all bookings at that particular day. Here you can see your newly created booking.

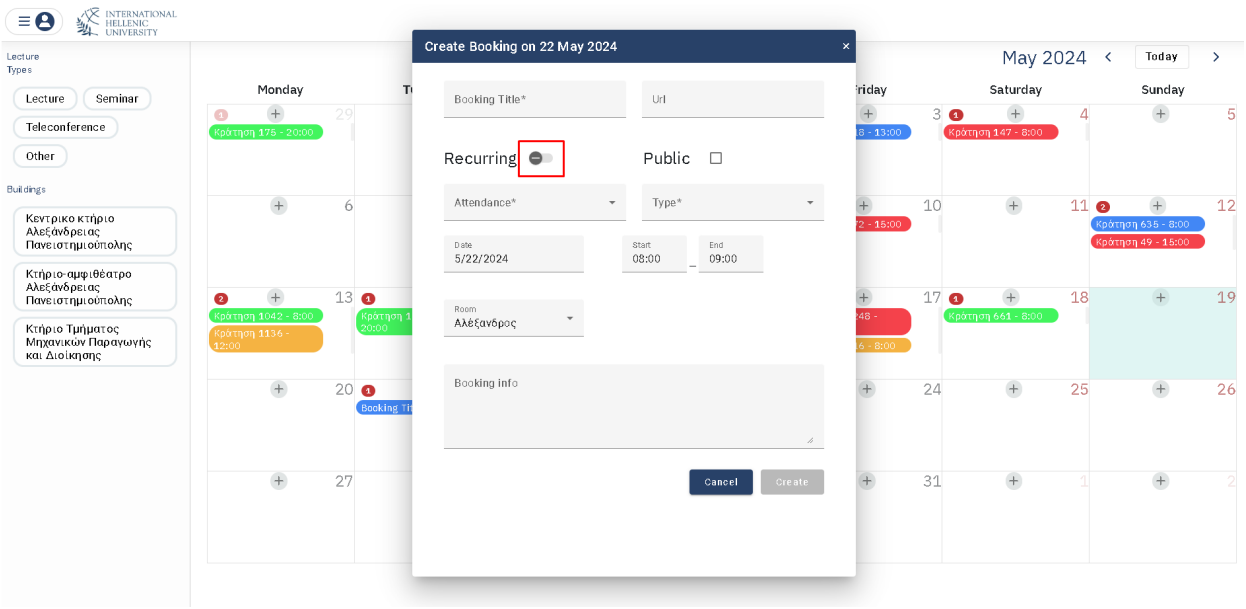
A.2.7 Step7: Inspecting booking information



From the open day view you can press any of the listed bookings to view additional information concerning it. As you can see a dialog appears that lists all the provided information regarding the booking.

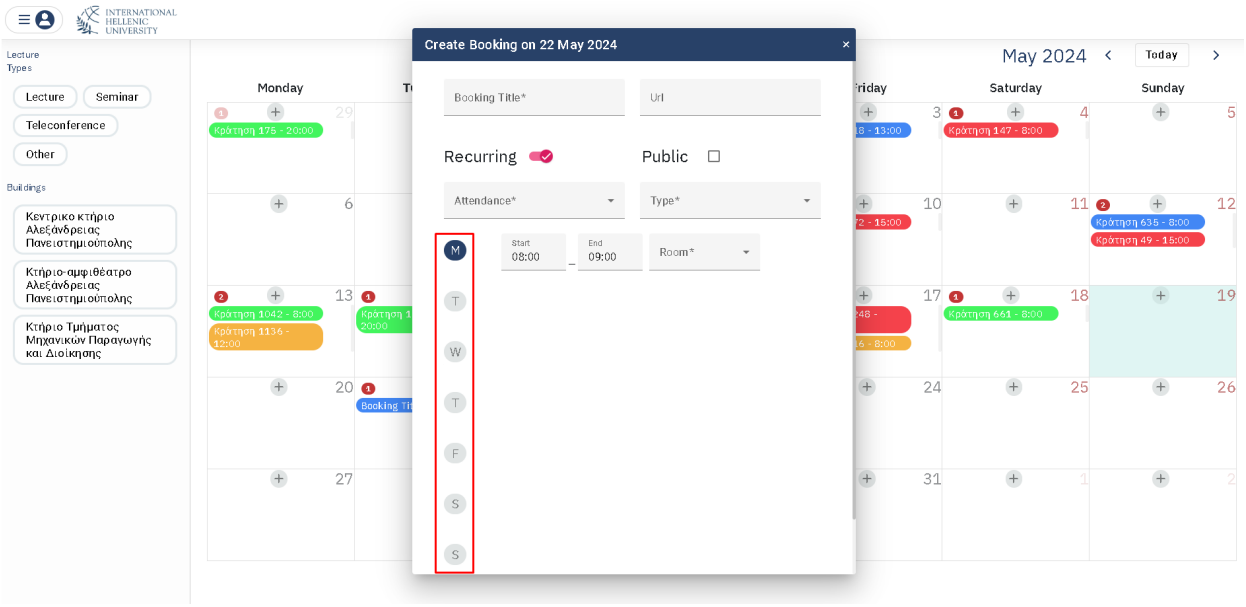
A.3 Creating a recurring booking

A.3.1 Step 1: Switch to Recurring Form



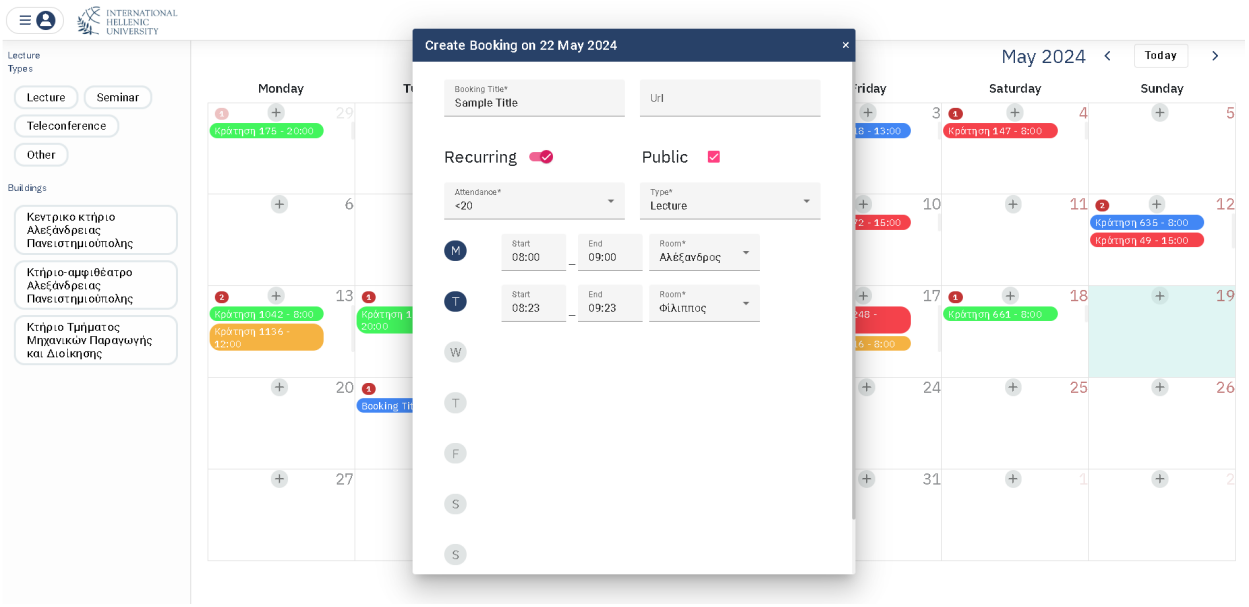
To create a Recurring Booking one needs to follow the same steps as above to get to the booking form dialog and then locate the Recurring check switch pictured here.

A.3.2 Step 2: Recurring Days



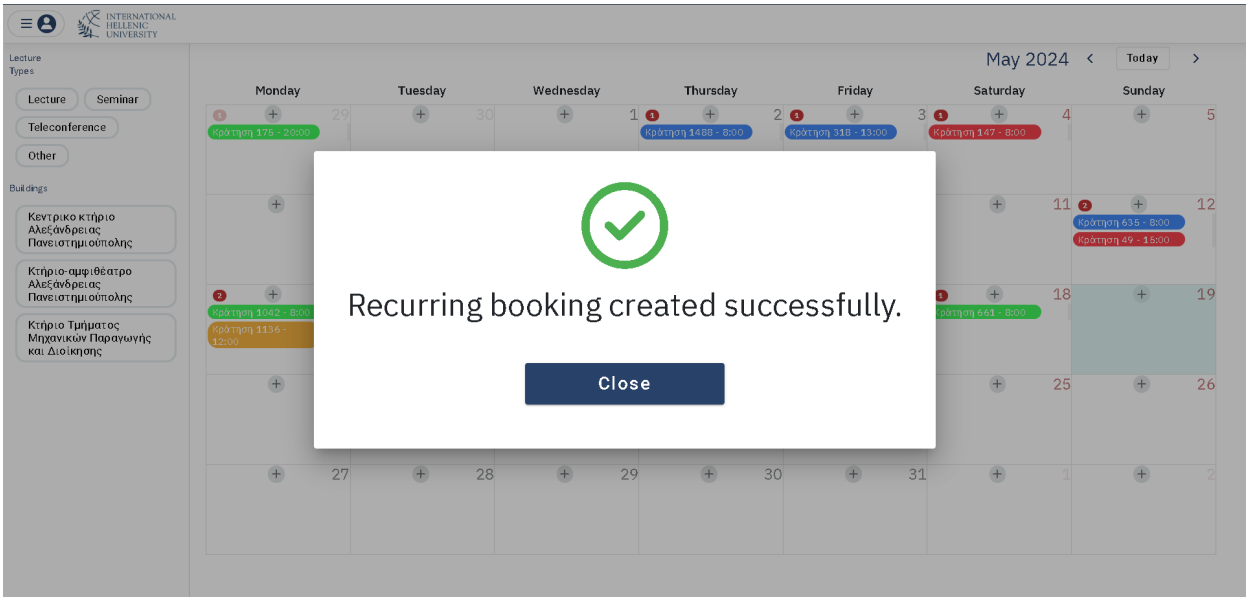
Once the dialog switches to its recurring version the available days are pictured here.

A.3.3 Step 3: Days, times and rooms



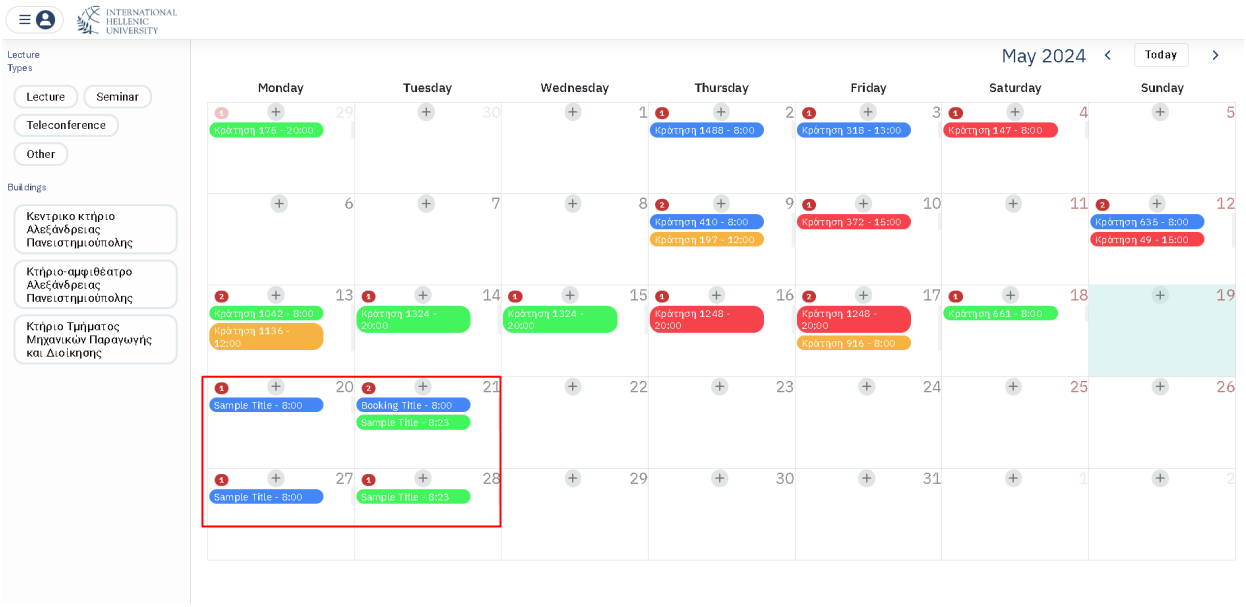
Then which days,times and rooms on which the booking should occur each week can be selected.

A.3.4 Step 4: Created Successfully



After pressing create the system once again informs after creating the booking. In the case of recurring bookings a moderator has to approve the relevant bookings in order to create their regular booking instances.

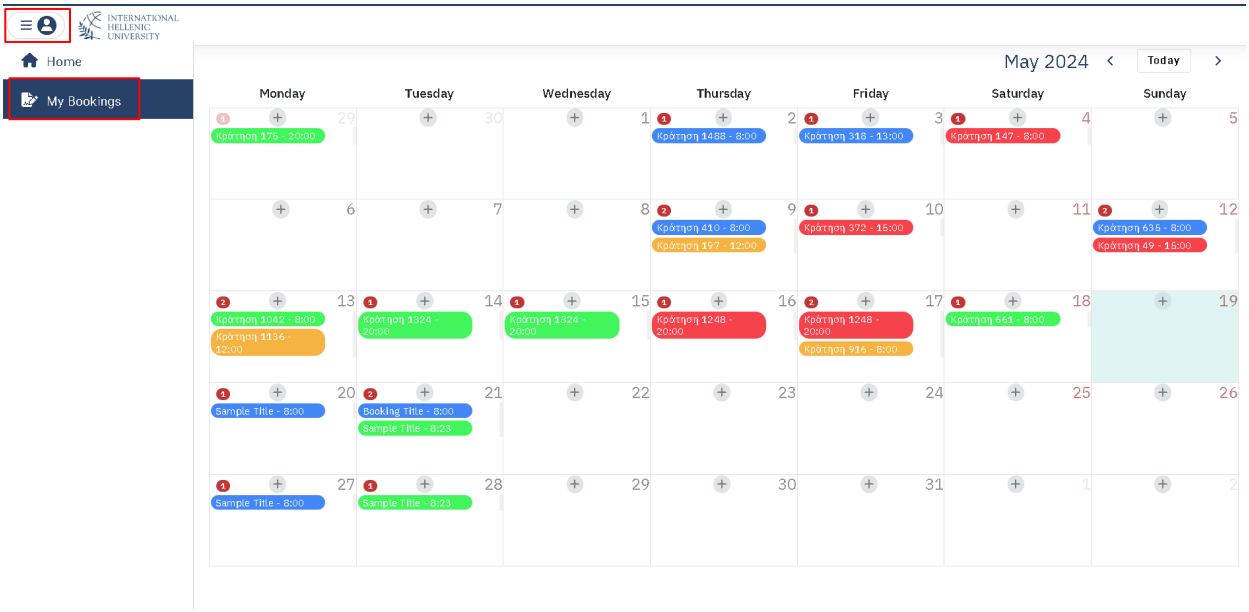
A.3.5 Step 5: Inspect Created Bookings



Once a moderator has done so and provided the booking was marked public the bookings can be seen in the homepage calendar.

A.4 User Edit Booking

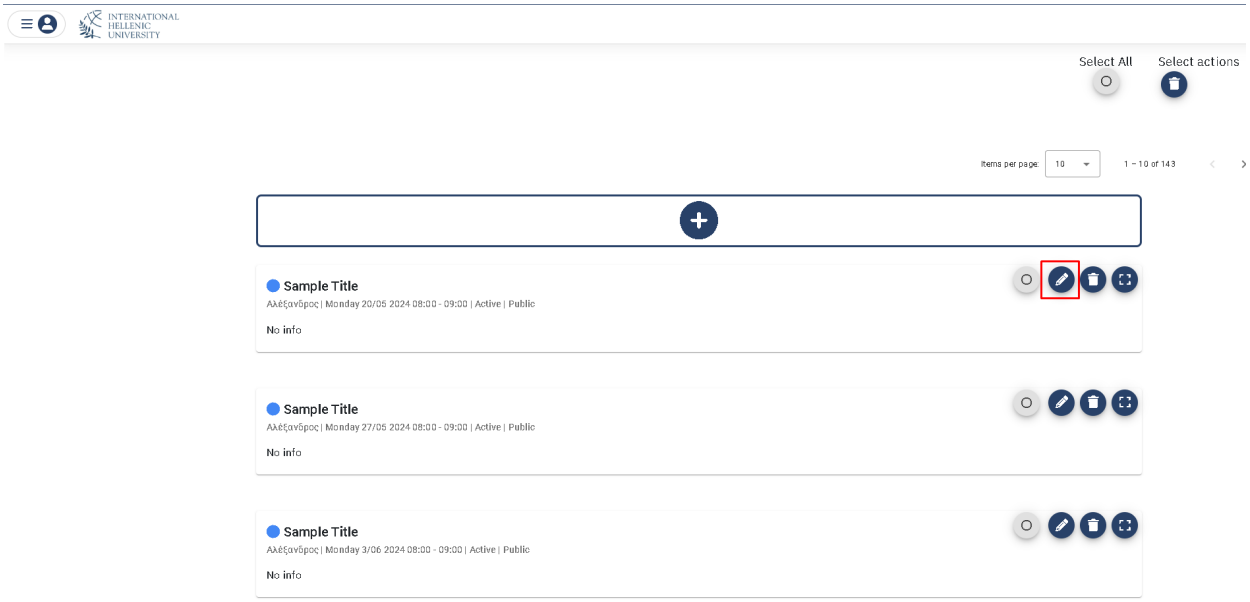
A.4.1 Step 1: Navigate to My Bookings



From the homepage by clicking on the sidebar buttons the sidebar opens and there is the option to

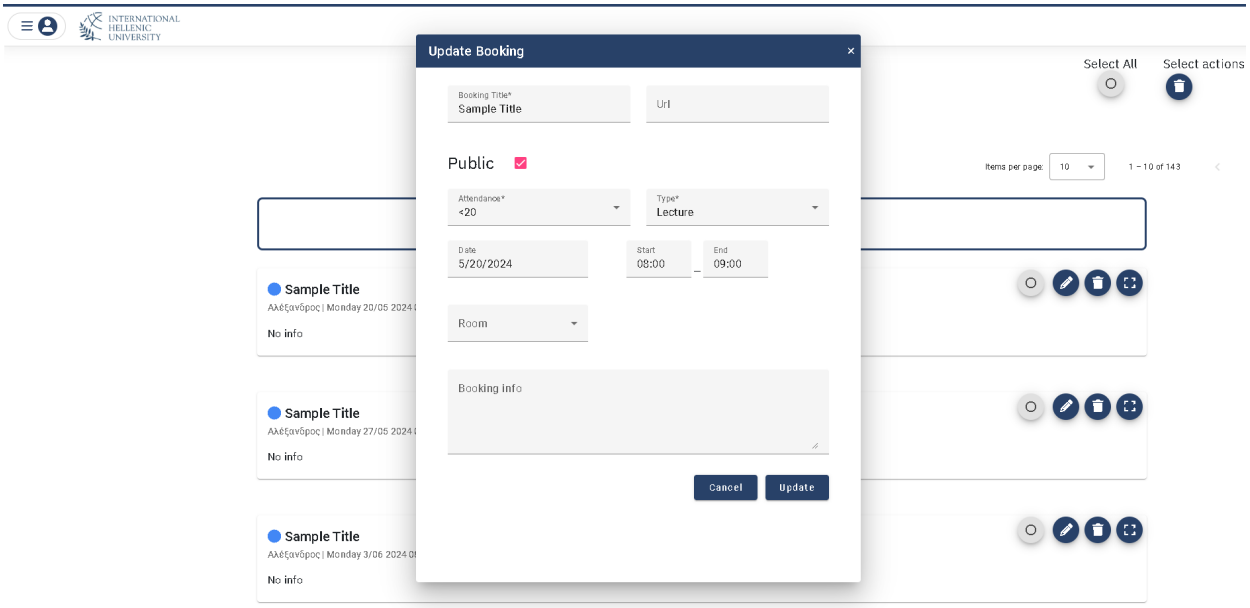
navigate to the My Bookings page.

A.4.2 Step 2: Open Edit Booking Form



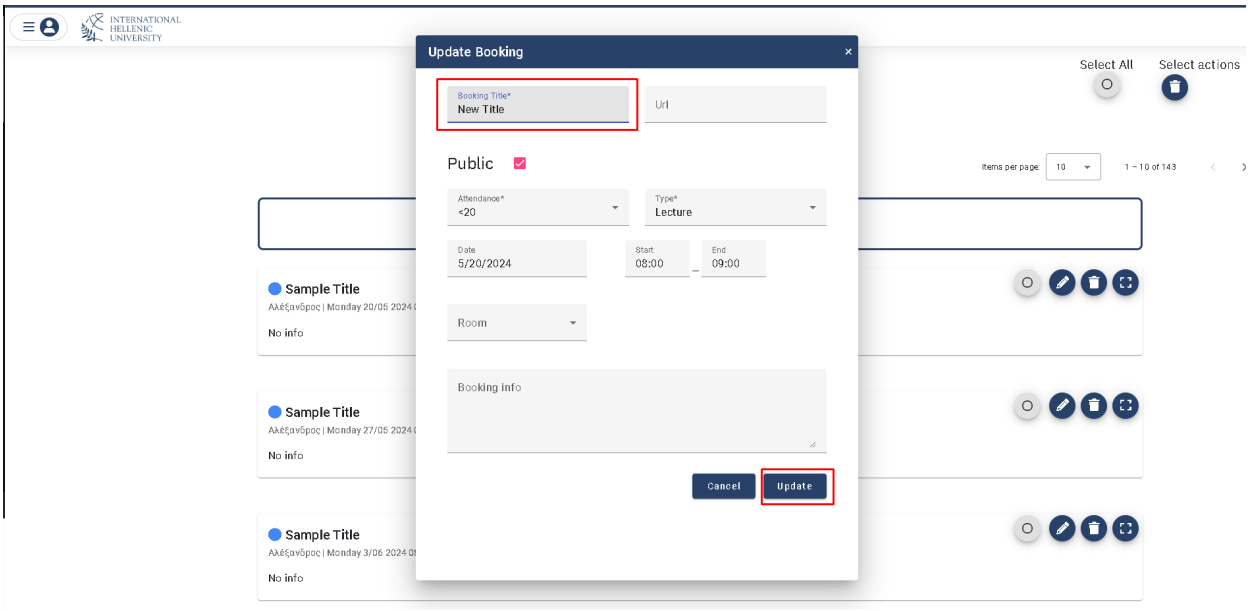
In this page a list of the bookings created by the User can be seen. By pressing the pencil symbol the edit booking form will open.

A.4.3 Step 3: Inspect Booking Form



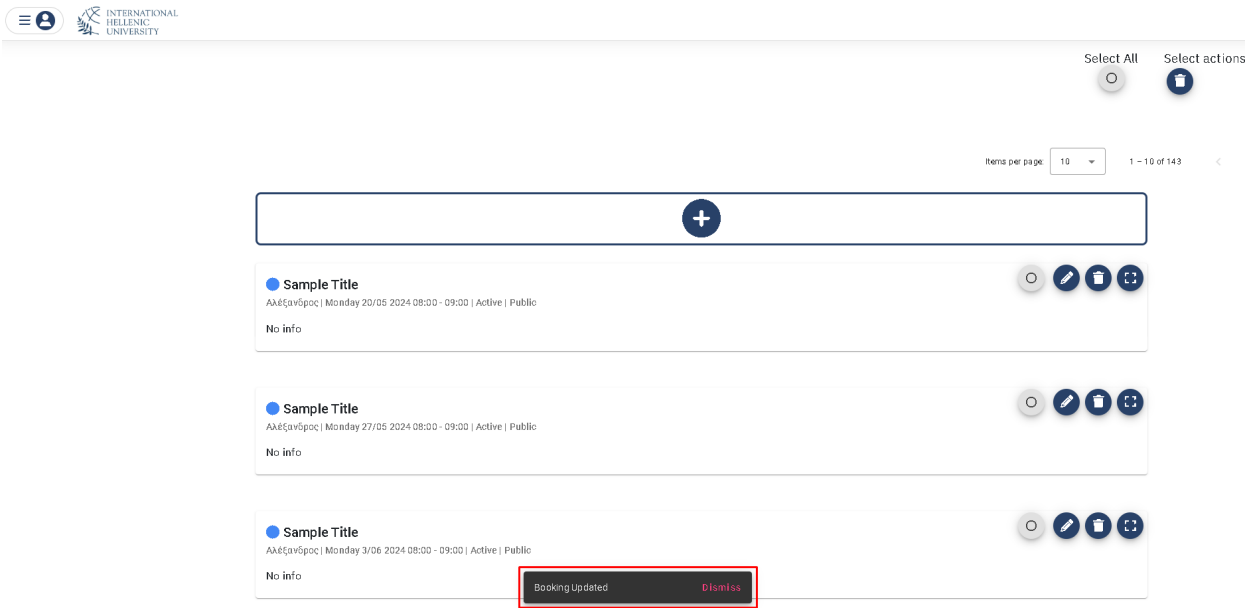
Here all existing relevant information for the booking can be seen.

A.4.4 Step 4: Make desired changes



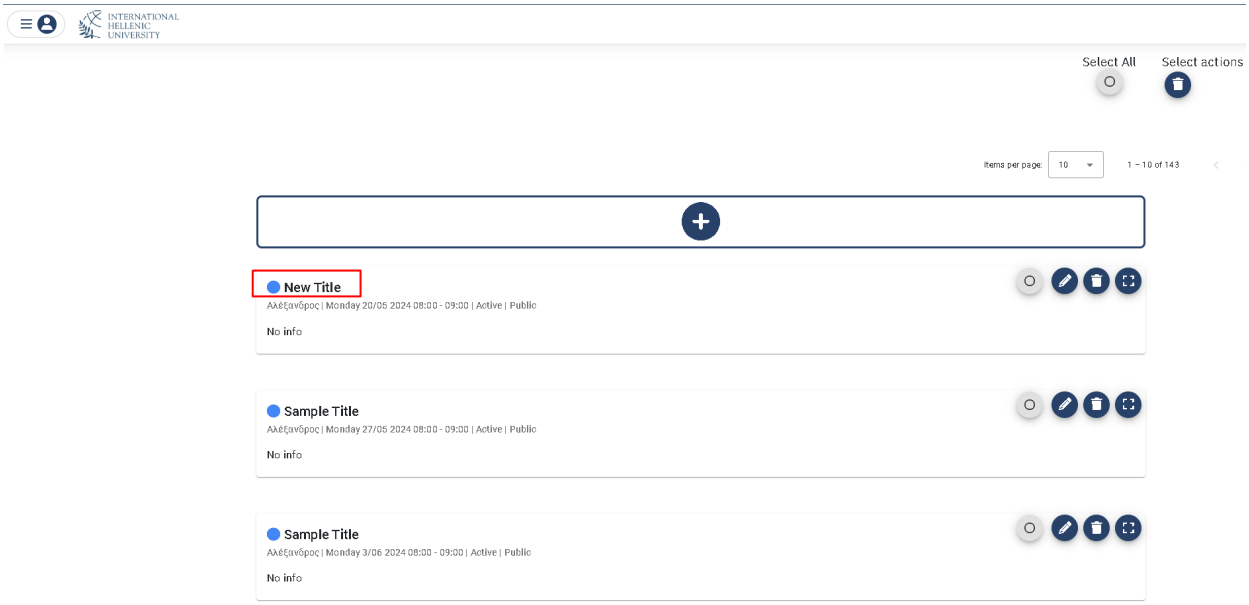
Change the value of any field to update the booking accordingly and press update.

A.4.5 Step 5: Booking Updated



Once the booking has been updated the system will inform you as pictured here.

A.4.6 Step 6: Inspect Changes



Then the booking list will be refreshed and the updated information can be seen.

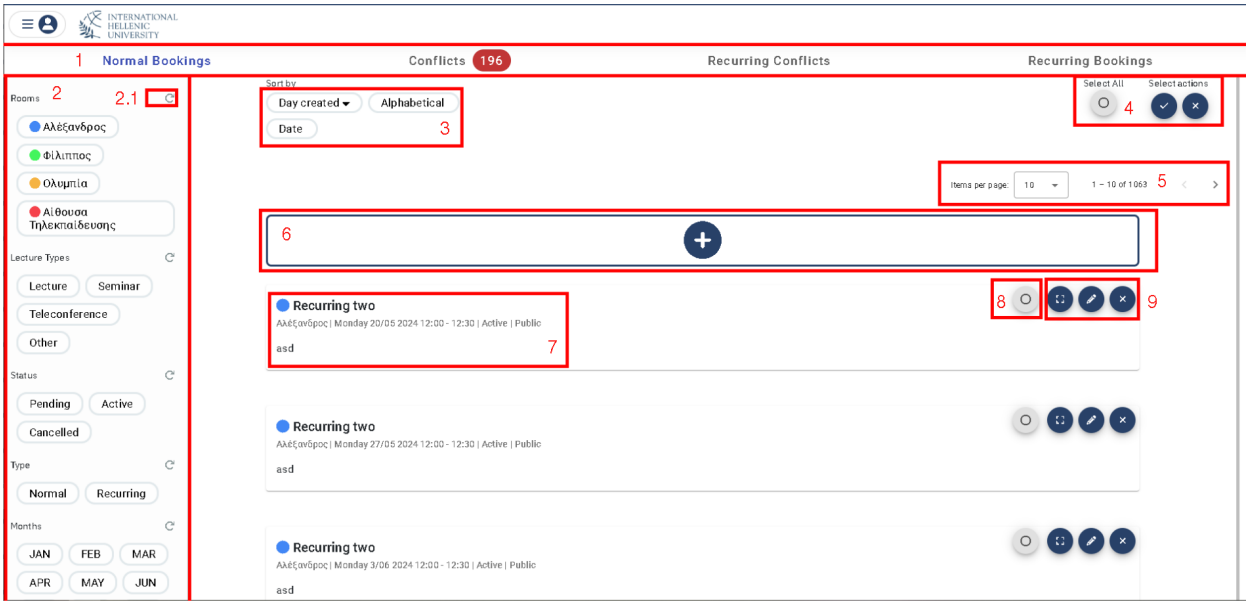
Appendix B

Moderator and Admin Appendix

In this chapter all the functions available to the moderators and administrators of the system will be walked through and explained.

B.1 Moderator Dashboard Map

Bellow is a general map of all the available features of the moderator dashboard.



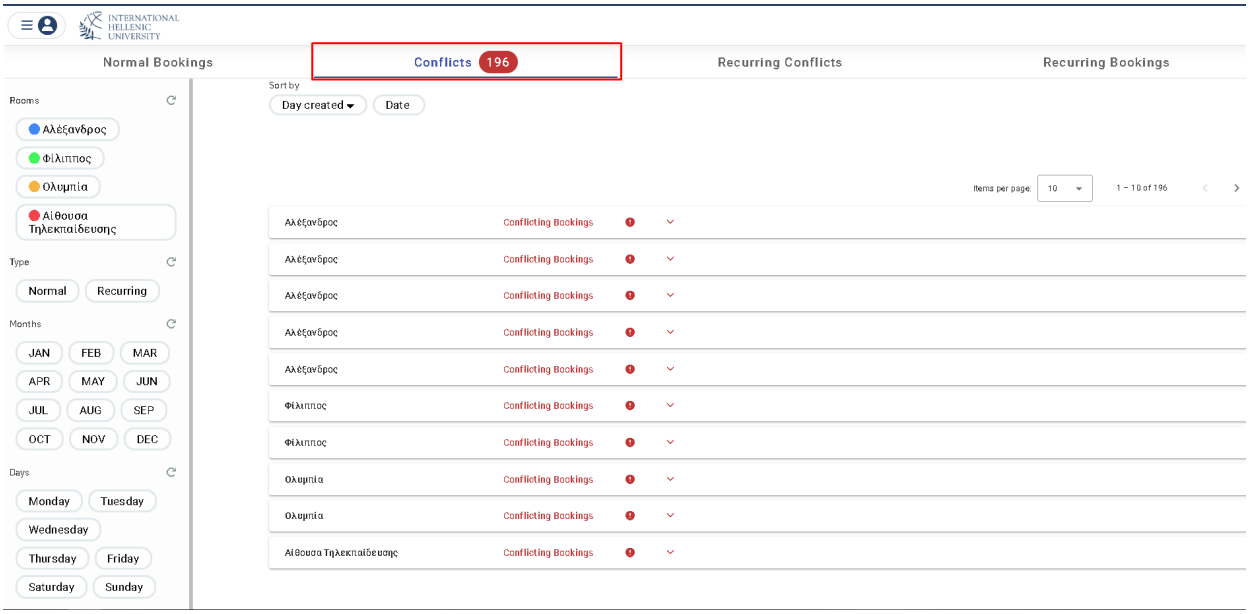
1. Tab control buttons. Each tab contains a list for the corresponding type of entity.
2. Filters by which to filter the listed entities. They work by clicking them to toggle them and multiple once can be selected

3. Filter control button. It deselects all filters for the specific category
4. Sorter controls. These sort the listed entities based on the selected option
5. Select Actions. These are controls that apply actions to all selected items. Either approval or deletion are the typical options. The circle button is used to select all displayed items of the current page
6. Pagination Control. This controls the pagination, including the page index and page size.
7. Create Item. This button opens the corresponding item creation form.
8. Item information.
9. Select button
10. Item actions. Typically include info,edit and delete.

B.2 Conflict Resolution

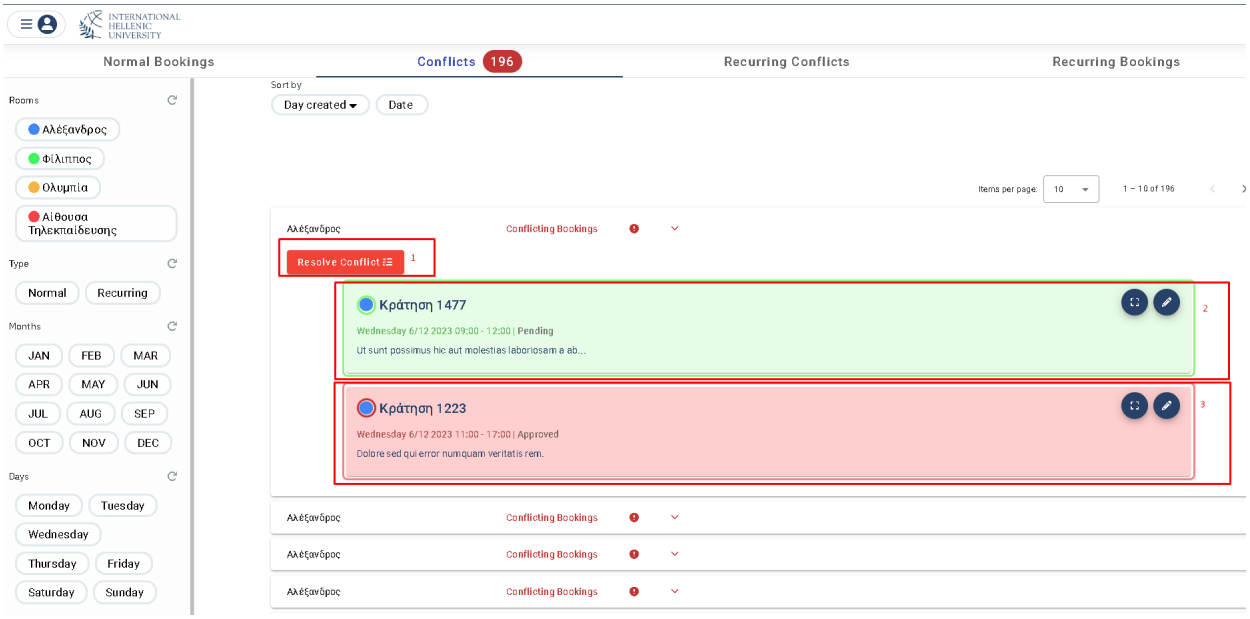
This sections explains the necessary steps and options to resolve a booking conflict. These steps are for the resolution of a regular booking conflicts but the same basic flow can be followed for recurring booking conflicts.

B.2.1 Step 1: Navigate to Conflicts tab



Here you will see a list of collapsible components with the name of the room that the conflicting bookings take place.

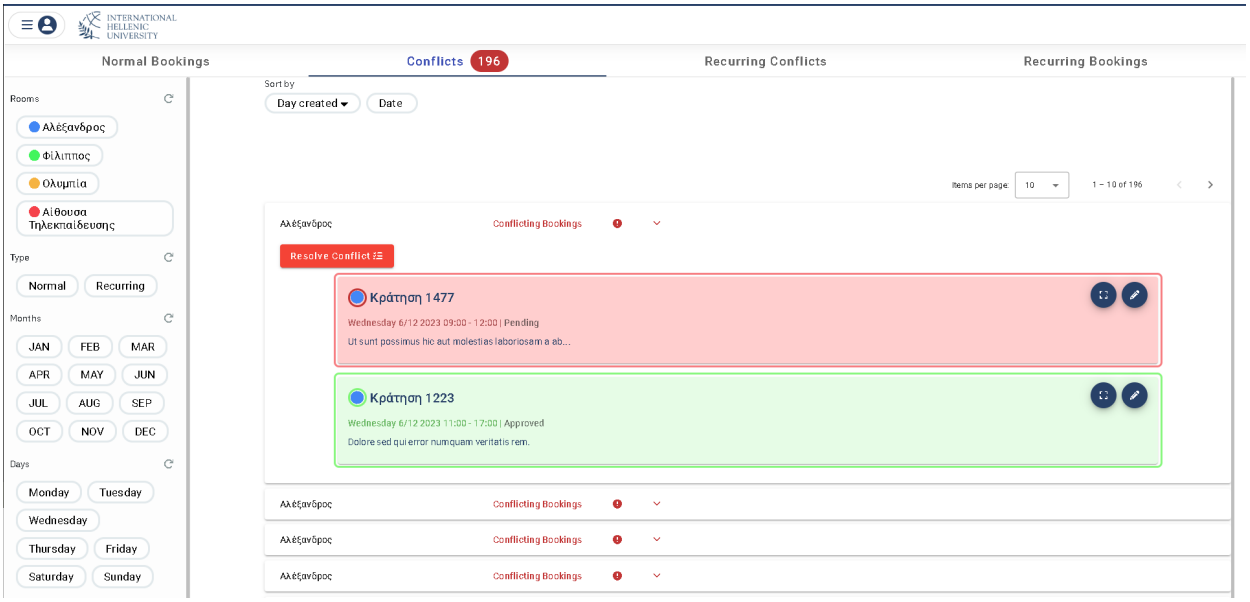
B.2.2 Step 2: Expand conflict list



Once clicked the component expands to show a list of all conflicting bookings that are part of this conflict group. These are all the bookings that conflict with each other for the specific room,date and time. Highlighted above are the following:

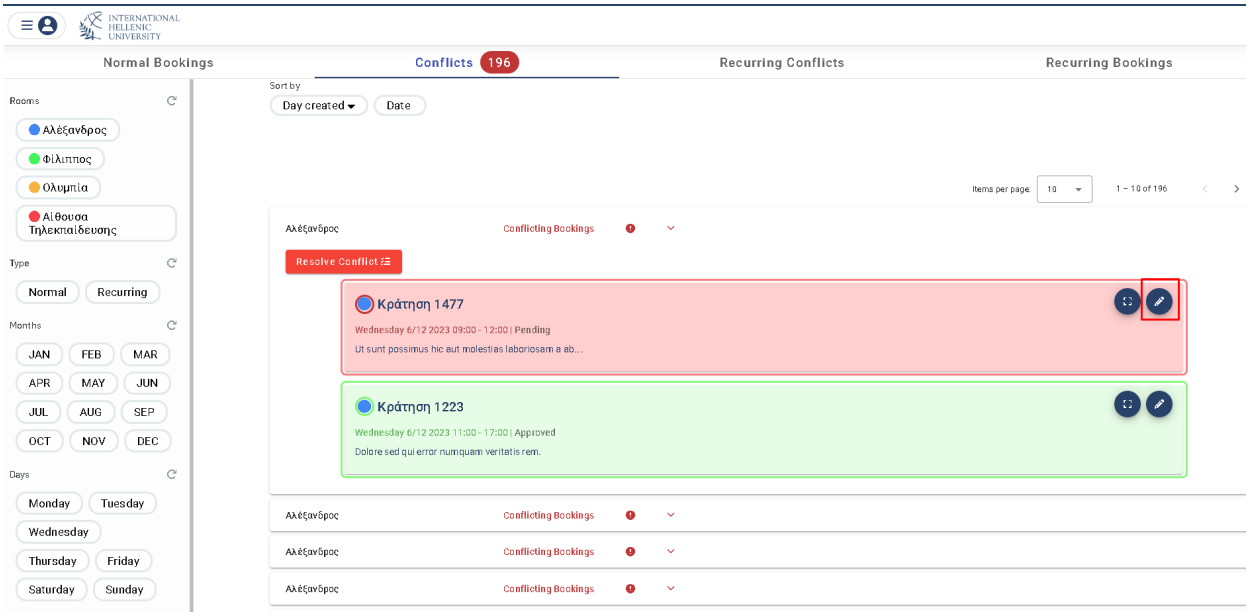
1. Conflict Resolution button. Once this button is pressed the system will resolve the booking according to the selected options. In this case it would keep the green colored booking and delete the red colored one.
2. Green color denotes the booking that will be kept.
3. Red color denotes the booking that will be deleted.

B.2.3 Step 3: Change Booking to Keep



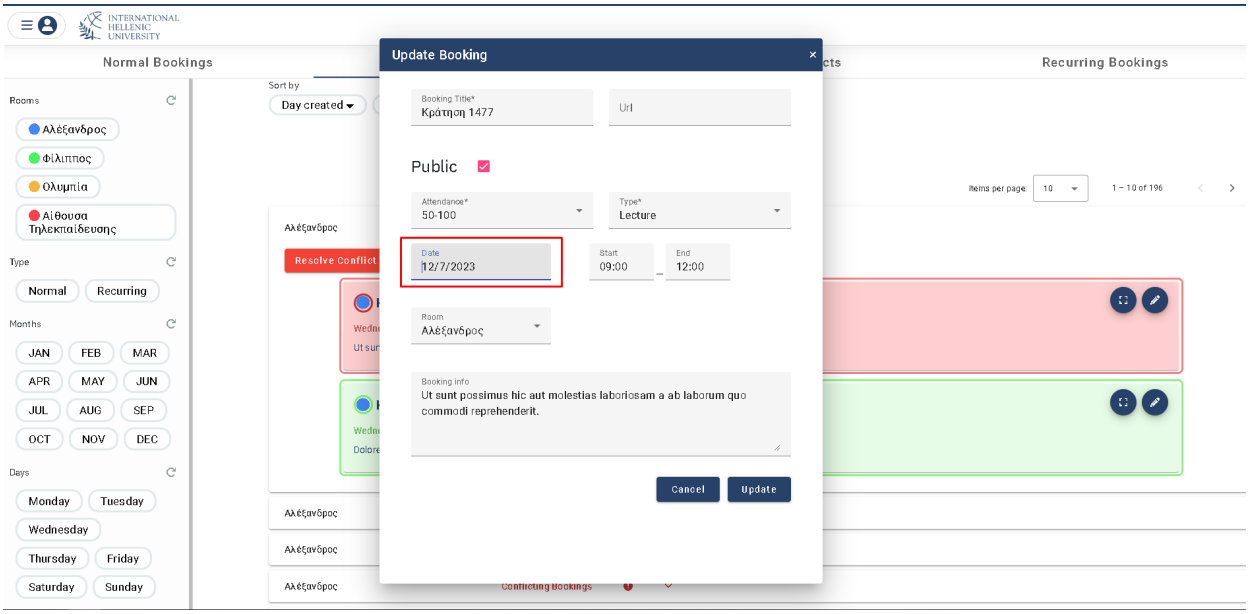
Clicking on a booking marks it as the one to keep while marking the rest as the ones to delete.

B.2.4 Step 4: Open edit Booking Form



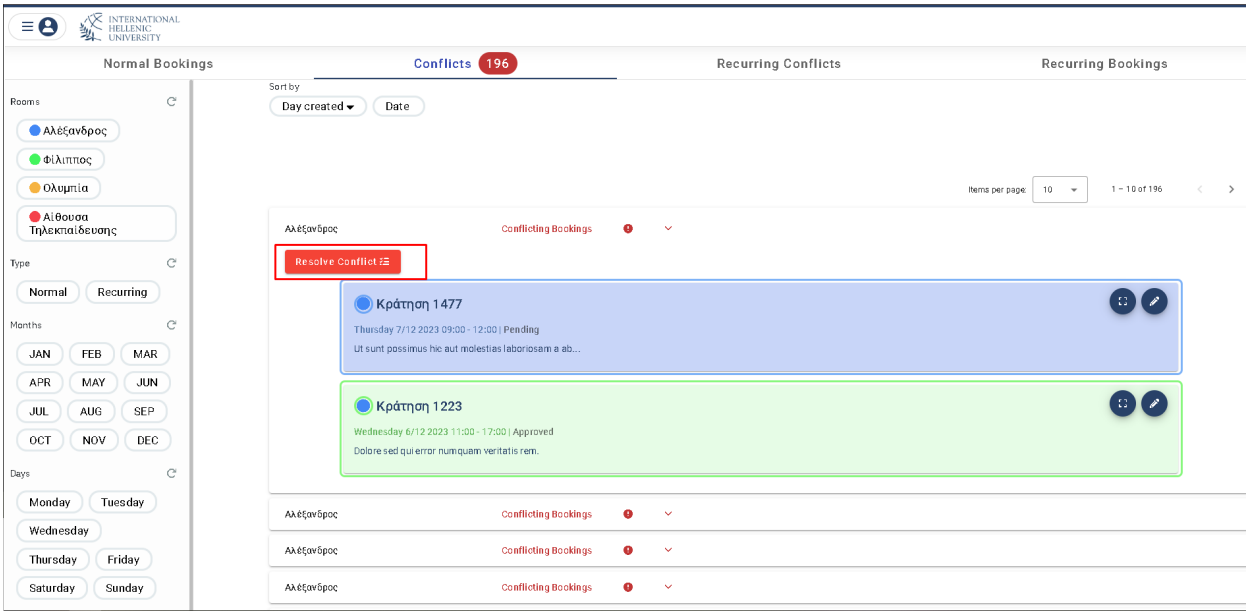
Pressing the pencil button will open the edit booking form where the booking's timing can be adjusted to resolve the conflict.

B.2.5 Step 5: Adjust timing accordingly



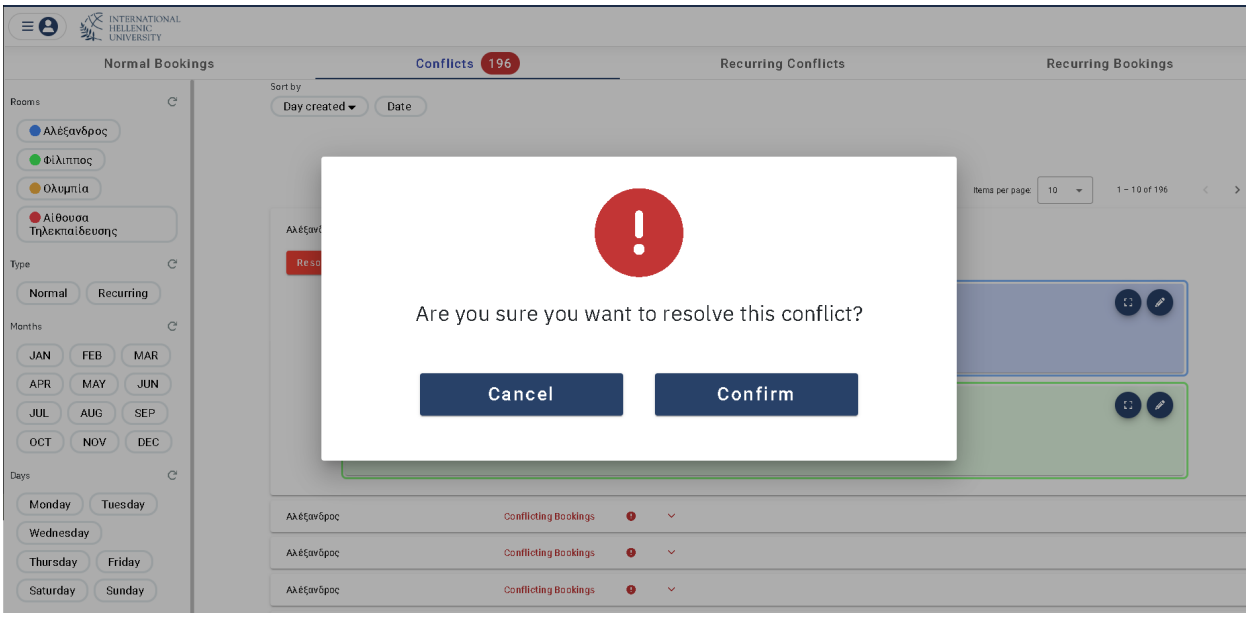
Moving the date of the booking will cause these two bookings to no longer be conflicting. Once the update button is pressed the system inspects the new date and time to determine whether it conflicts with any existing bookings.

B.2.6 Step 6: Inspect and Resolve



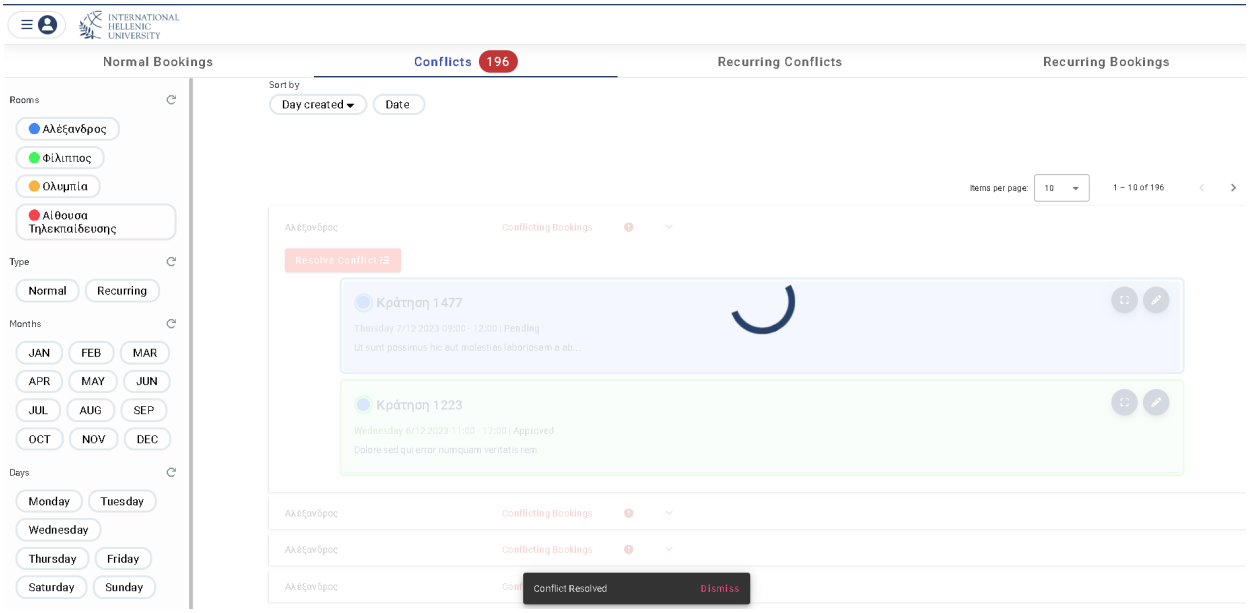
Here it is visible that the system has found that the edited date and time is available and has thus colored the edited booking blue. The blue color marks resolved bookings. Now by pressing the resolve button the changes made in the edit will be save and the conflict group will be resolved.

B.2.7 Step 7: Confirm



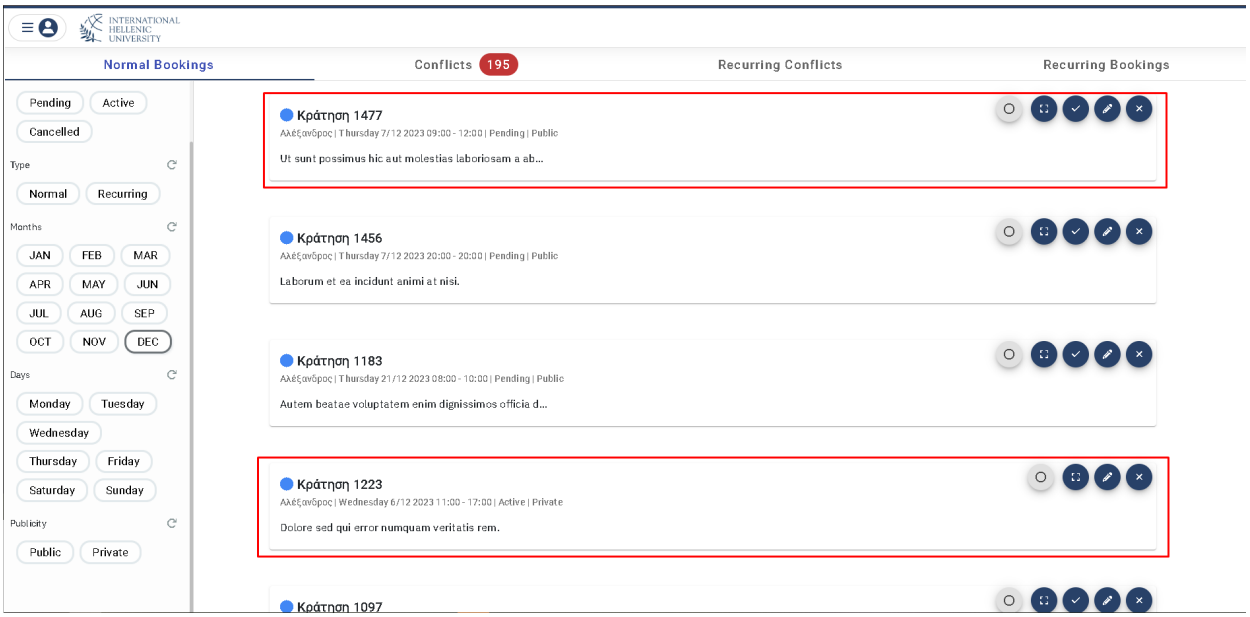
It is important to pay attention when resolving conflicts as deleted bookings can not be restored by the system.

B.2.8 Step 8: Conflict Resolved



Once the changes are finalised the systems informs the user and refreshes the lists.

B.2.9 Step 9: Inspect Changes

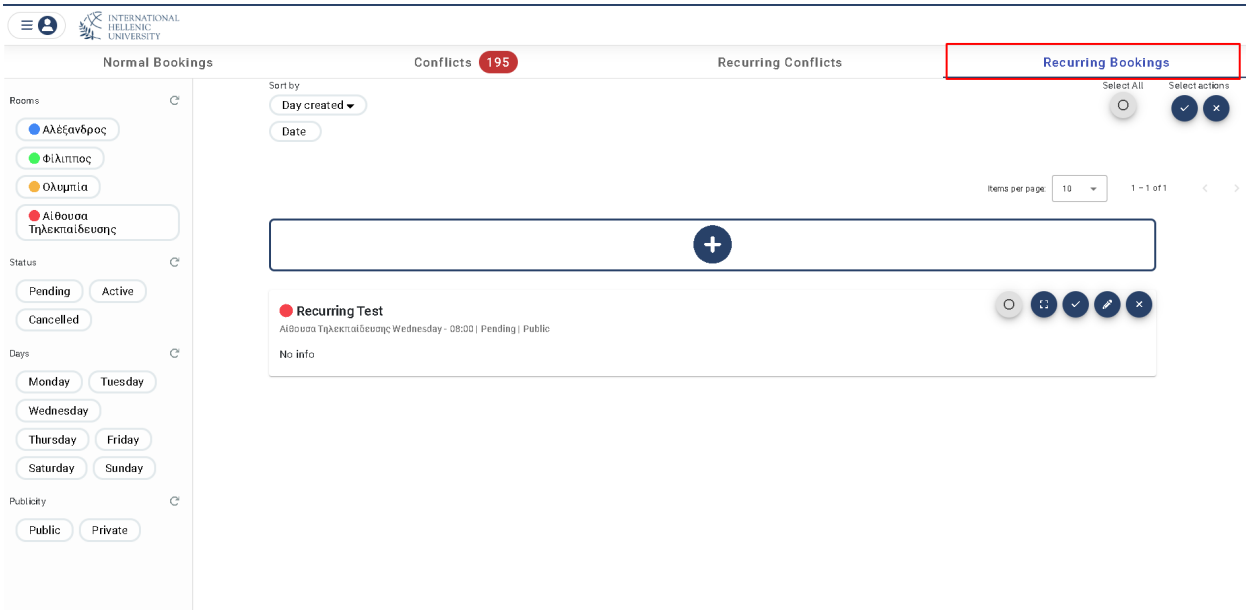


Here we can see the previously conflicting bookings now existing normally in the list of normal bookings with the updated date and time.

B.3 Approve Recurring

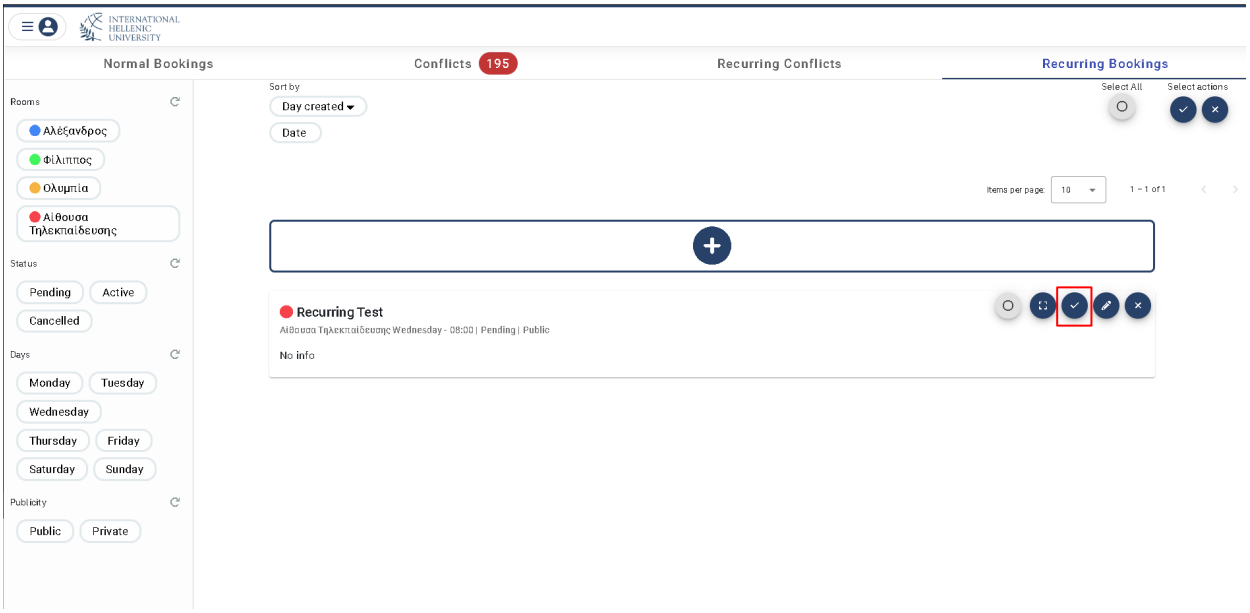
In this section we break down the specific steps needed to approve a recurring booking and thus create its normal booking occurrences.

B.3.1 Step 1: Navigate to the Recurring Bookings tab



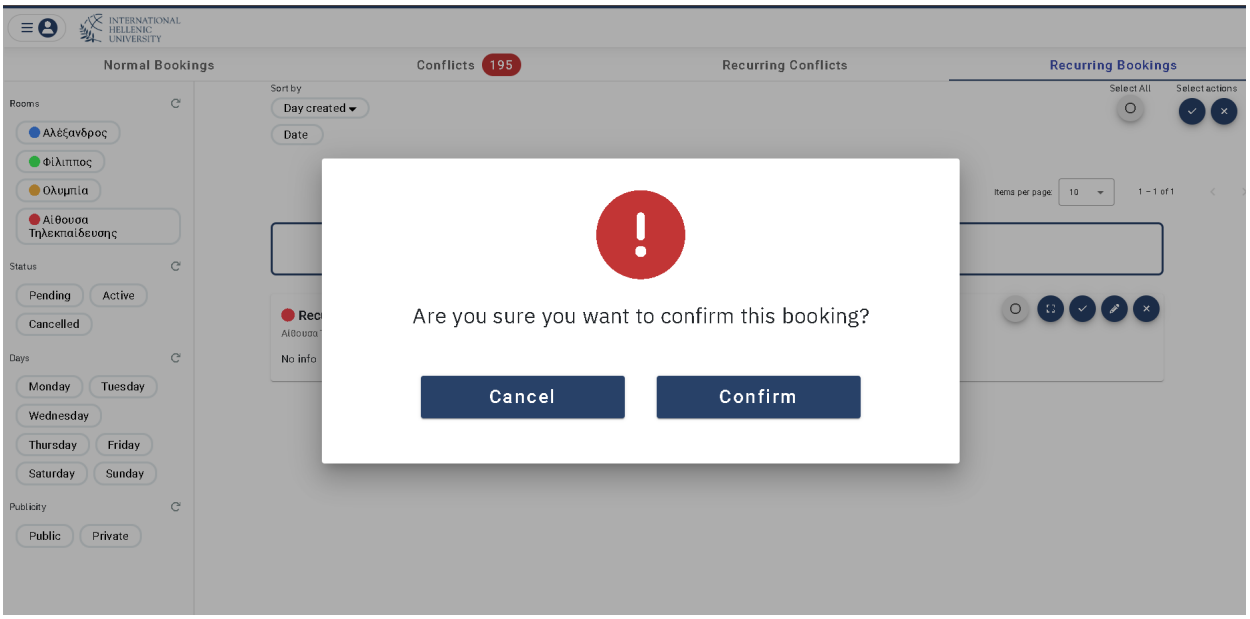
In this tab contains a list of all the recurring booking groups.

B.3.2 Step 2: Press Approve Button



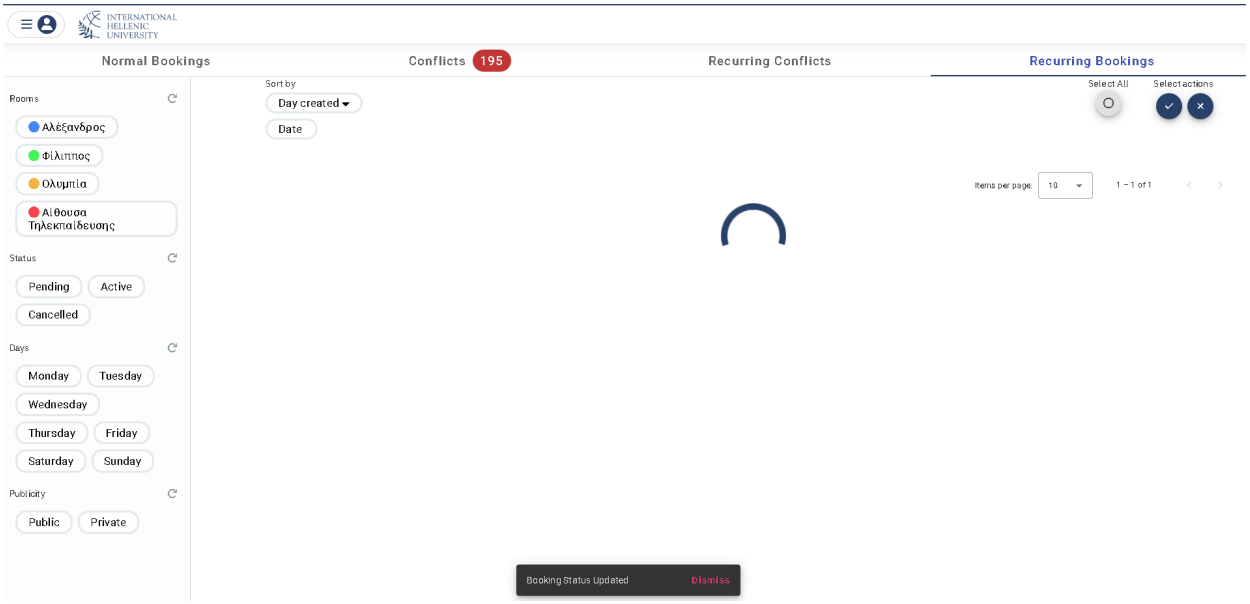
Pressing this button begins the approval process.

B.3.3 Step 3: Confirm Choice



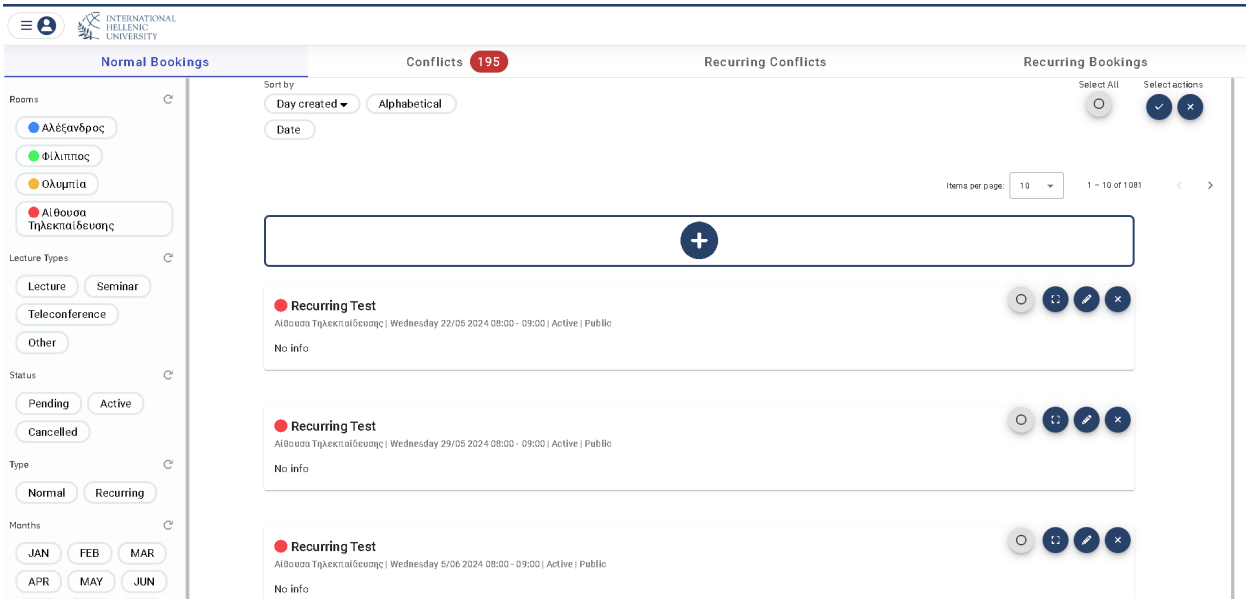
Approving a recurring booking will create normal booking instances in every week of the semester. This means that approving recurring bookings can quickly fill the available time slots in a semester.

B.3.4 Step 4: Receive System Confirmation



Once the approval process is done and all the normal booking instances are created the system notifies and refreshes the lists.

B.3.5 Step 5: Inspect Results

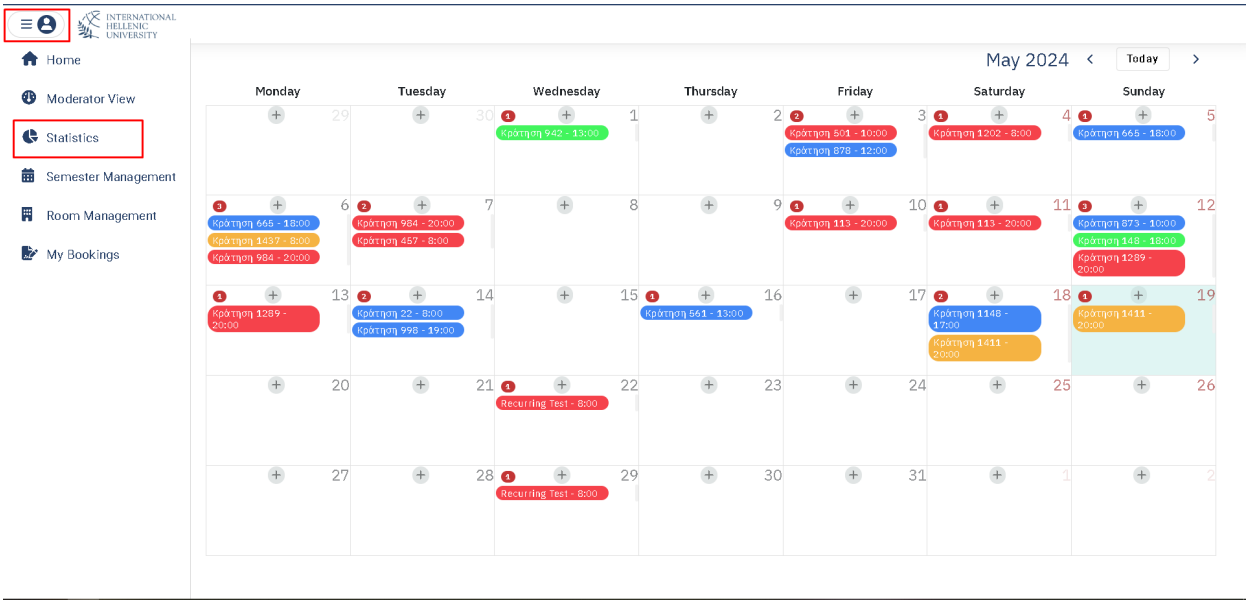


The normal booking instances can be observed in the list of the Normal Bookings tab.

B.4 Statistics

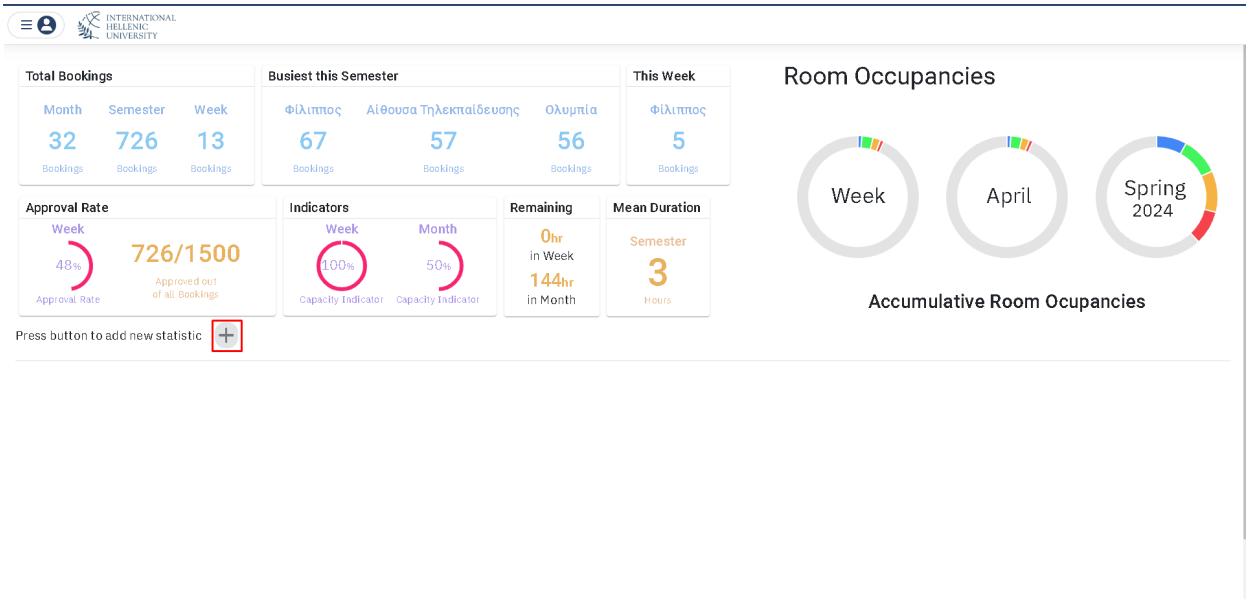
This section explains the basic flow of actions that create the statistical charts. The exact information provided by each stat and the available options for each are explained in the main text of this paper.

B.4.1 Step 1: Navigate to the Statistics page



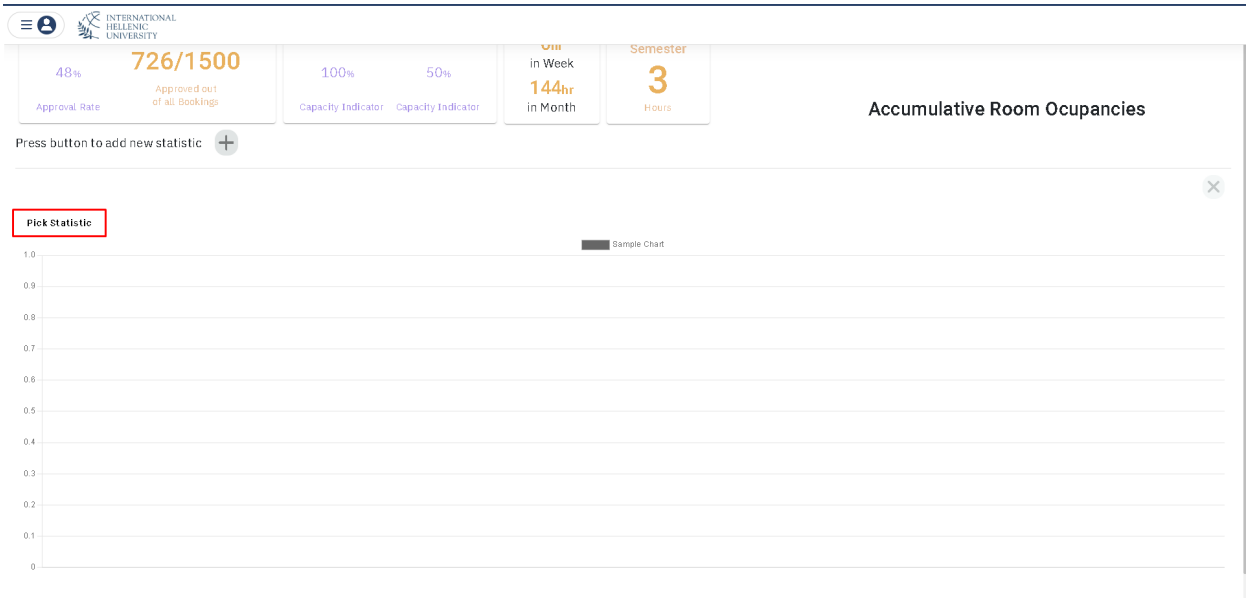
Using the sidebar select the Statistics option.

B.4.2 Step 2: Add new chart



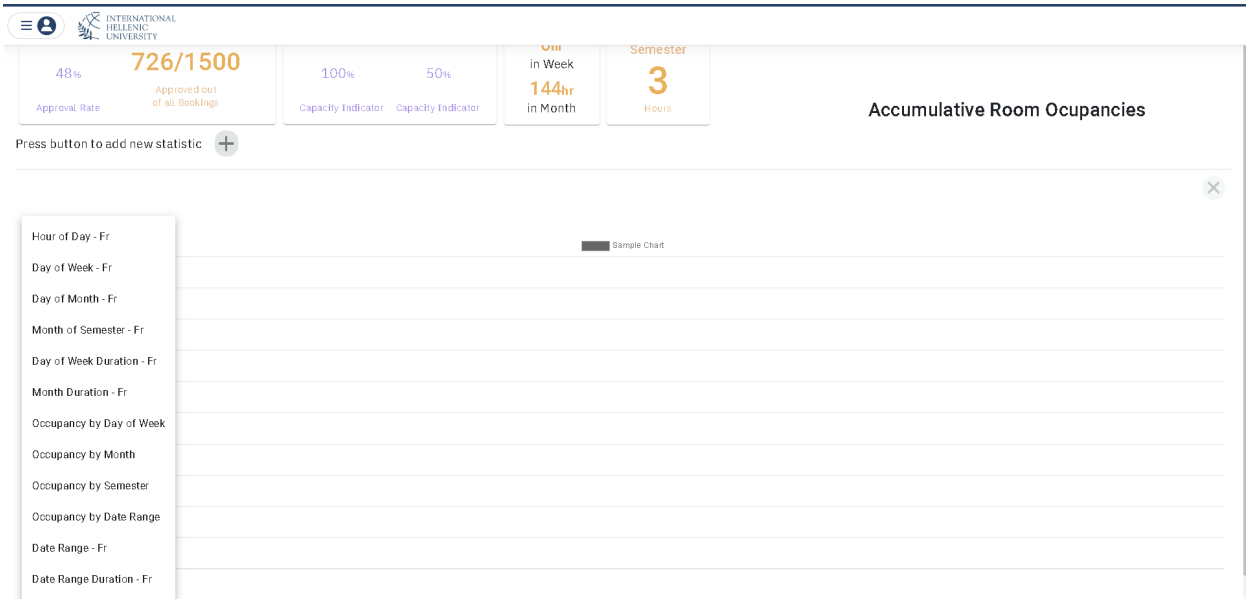
By pressing the highlighted + button a new chart can be added.

B.4.3 Step 3: Select Chart Type



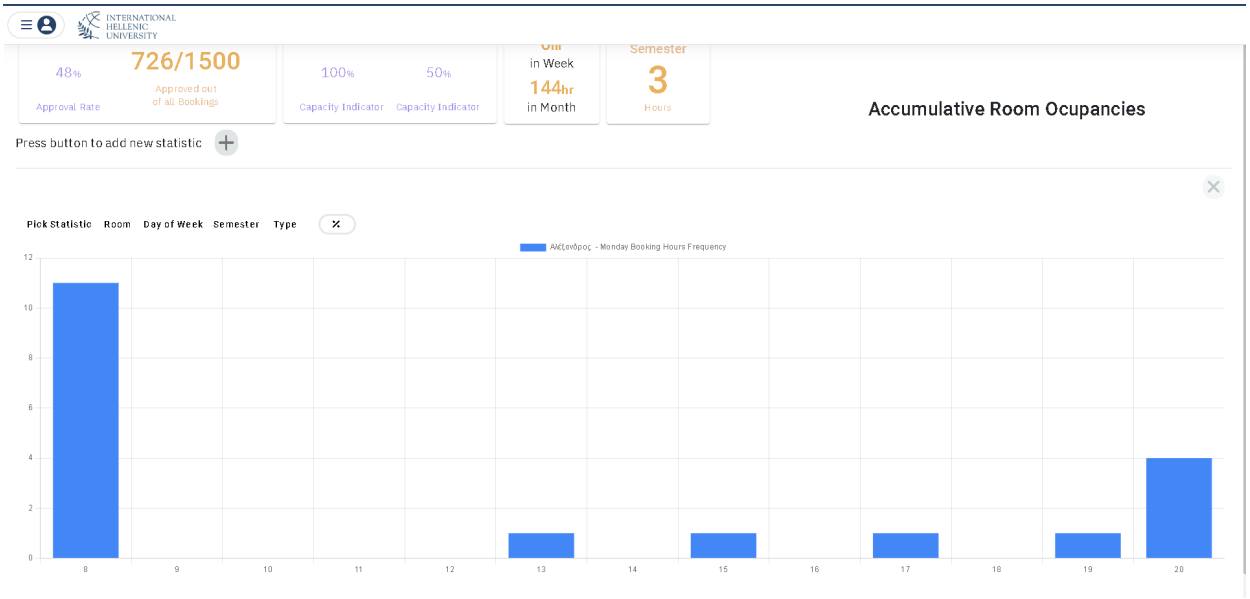
Pressing the highlighted button opens the menu containing all the statistic types.

B.4.4 Step 4: Pick from menu



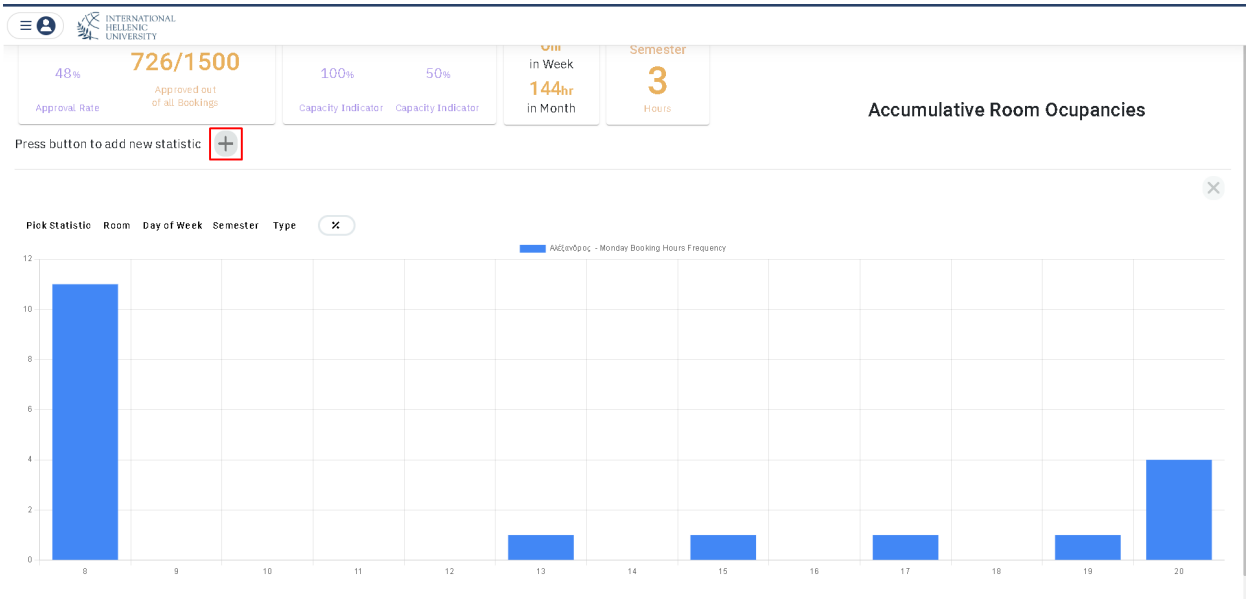
Picking any of these options will retrieve the associated data set for this particular statistic type.

B.4.5 Step 5: Statistic generated



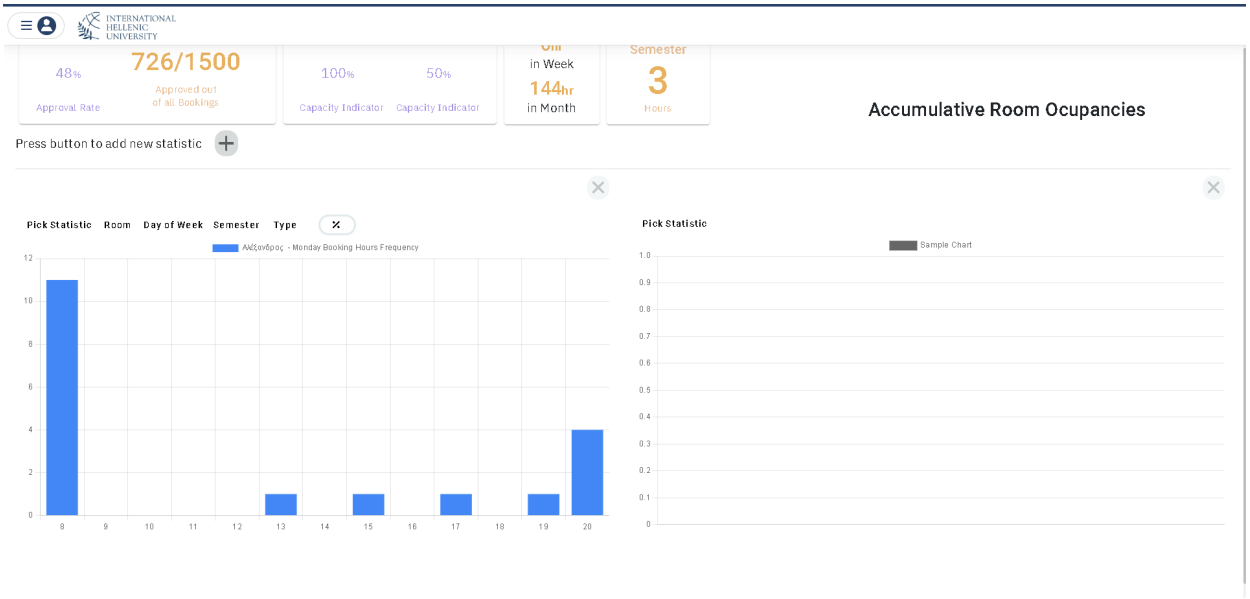
Now that the chart is generated additional options can be selected.

B.4.6 Step 6: Second Statistic



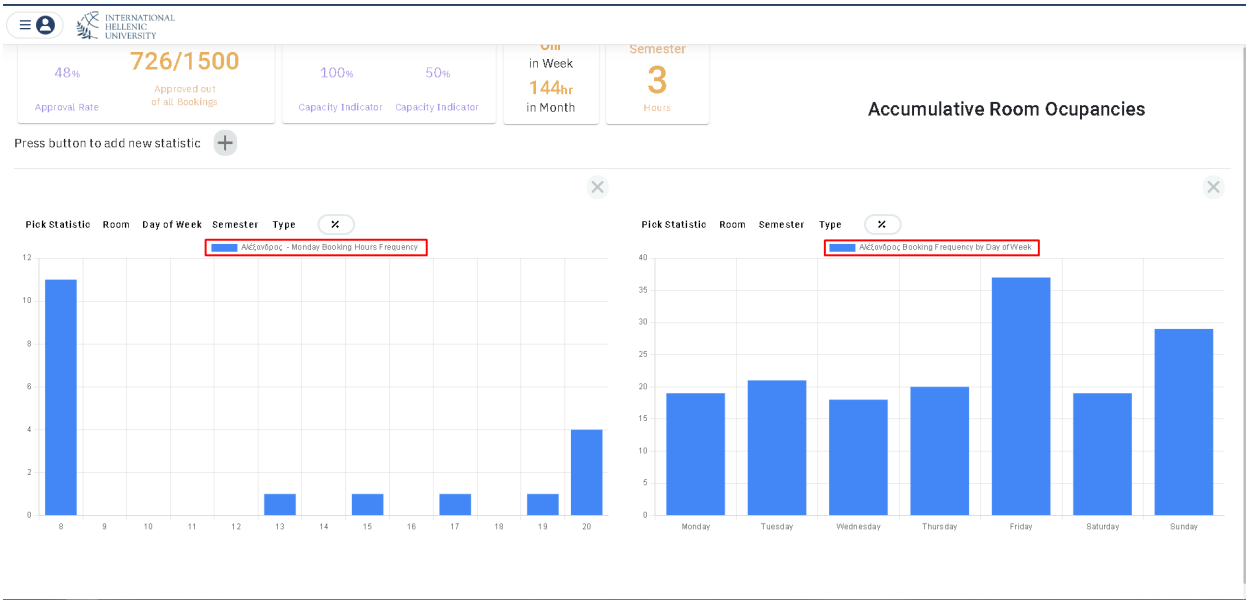
Now that a second statistic has been added the size of the first has been reduced. This is the general configuration for any number of generated statistics.

B.4.7 Step 7: Select second type



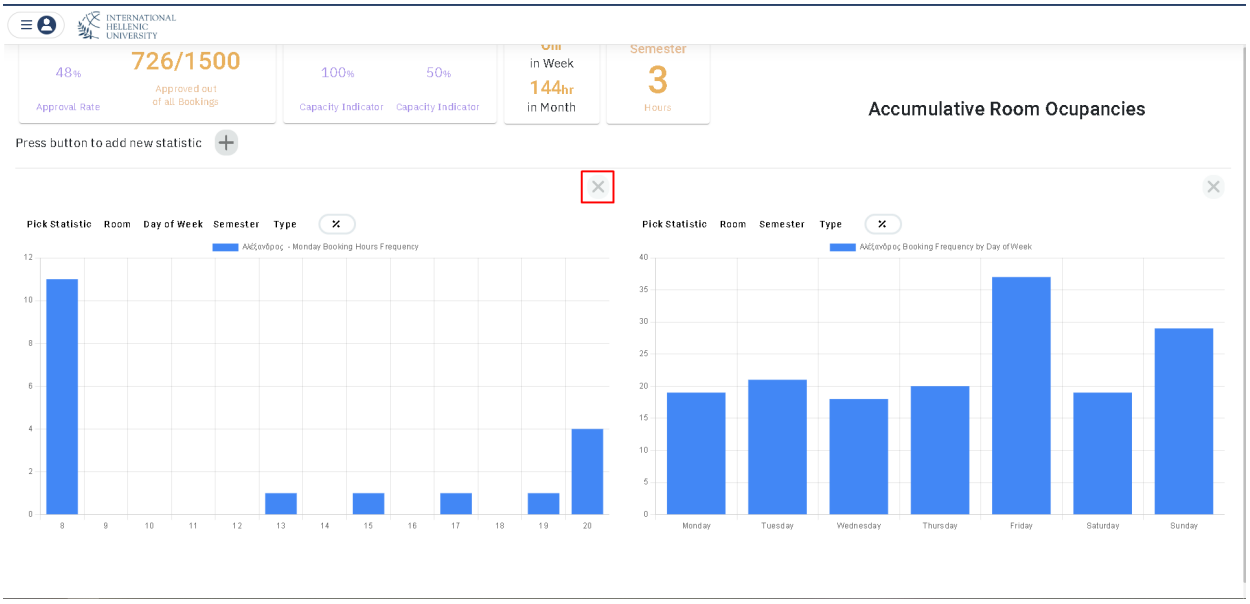
By selecting a second type and generating a new chart comparison between samples, options and rooms can be made much more easily.

B.4.8 Step 8: Inspect charts



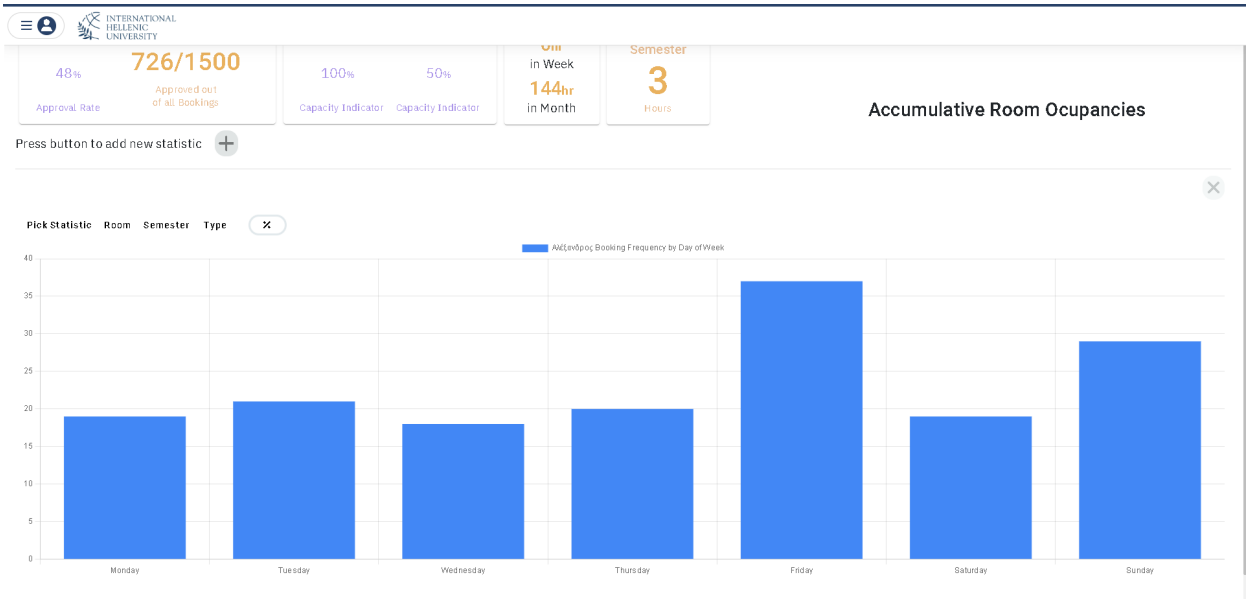
The type of the generated chart can be seen in the highlighted areas.

B.4.9 Step 9: Remove chart



By pressing the highlighted button the corresponding chart is removed.

B.4.10 Step 10: Original size configuration



Now that only one chart is in the row the original size configuration is restored.

B.5 Running Queries on the Database

In order to exercise full control over the state of the system, the option of direct manipulation of the database is available to the Administrator. This section provides the steps necessary to run any SQL query to the database as well as provide a direct example for creating additional departments and buildings as well as associating them using the pivot table buildingDepartment.

B.5.1 Prerequisites

In order to be able to follow the bellow steps, the Administrator should have access to the server that runs the system and knowledge of how to operate a bash environment.

B.5.2 Step 1: Connect to MySQL

Assuming the Administrator has logged into the server and is ready to execute the desired commands the first step they should follow is to connect to the database via MySql by typing the following.

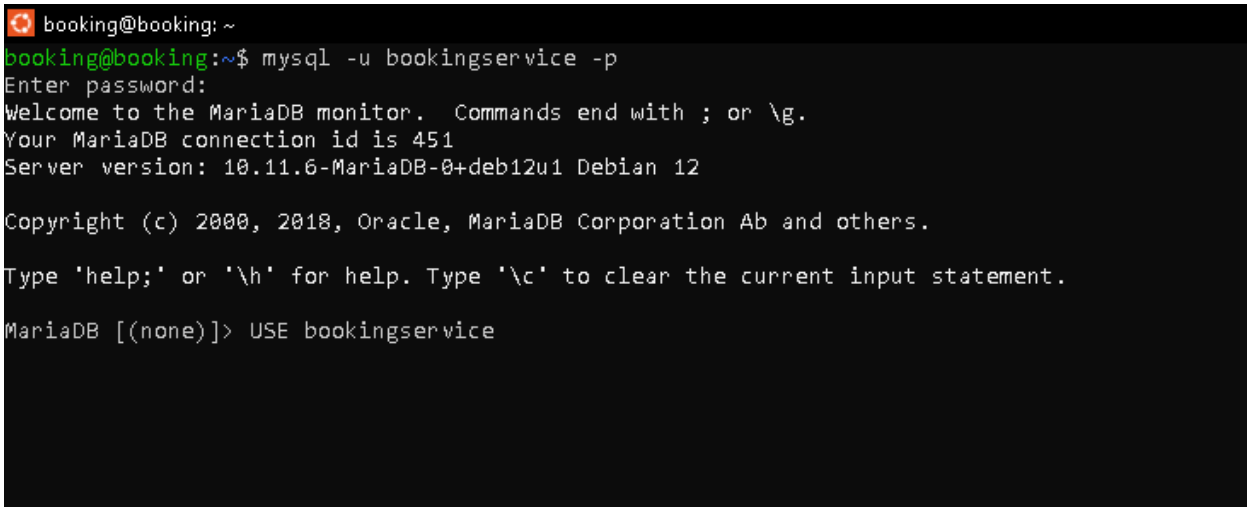
mysql -u bookingservice -p

Afterwards the Administrator will be prompted for the password to the database.

B.5.3 Step 2: Selecting the Database

After logging into the database, we need to specify on which database the sql queries will be run. We do this by typing the following.

USE bookingservice

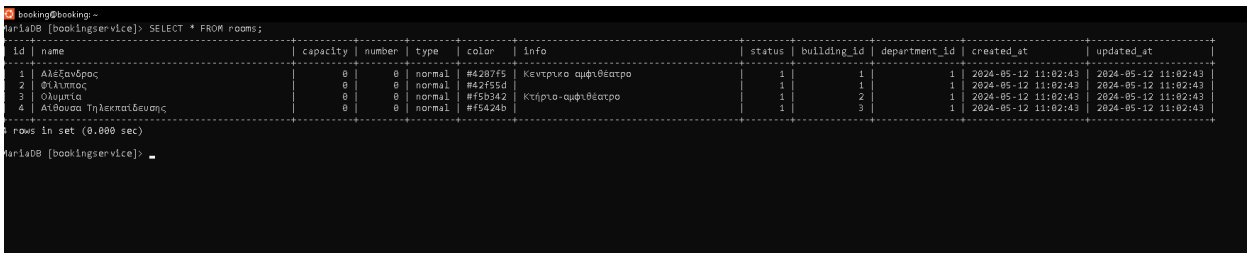


The system will then inform whether the selection has changed.

B.5.4 Step 3: Running a Query

Now we can run any SQL query we want and directly inspect or manipulate the database. For example we can see all the entries of the room table by typing the following.

SELECT * FROM rooms;



B.6 Creating Departments, Buildings and Associating them

This section provides steps for the insertion of new buildings and departments as those are the only models of the database that can't be managed through the interface of the system.

B.6.1 Step 1: Insert Building

To insert a building run the following query.

INSERT INTO buildings (name) VALUES ('Building A');

Replace 'Building A' with the appropriate name. Info can also be provided for the info column.

B.6.2 Step 2: Insert Department

To insert a department run the following query.

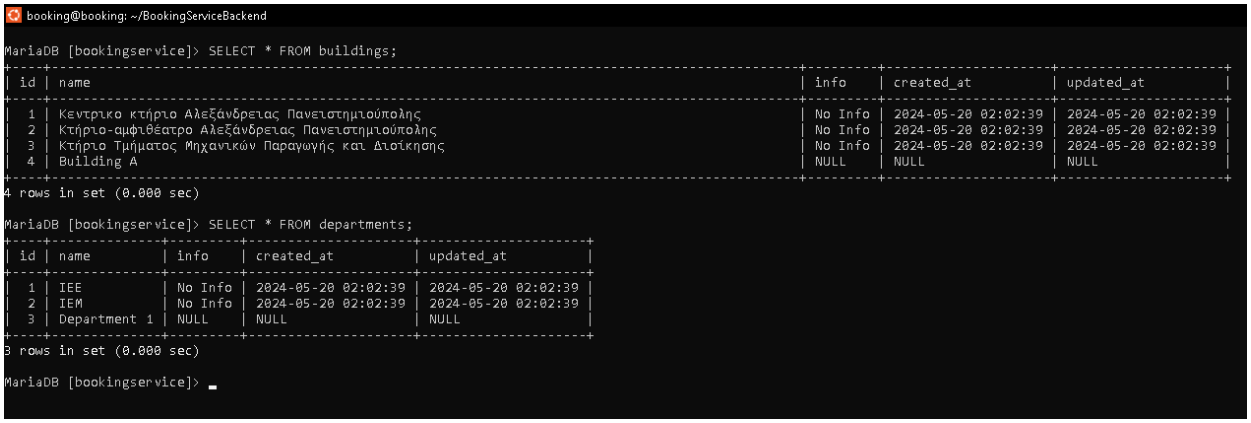
INSERT INTO departments (name) VALUES ('Department 1');

Replace 'Department 1' with the appropriate name. Info can also be provided for the info column.

B.6.3 Step 3: Find the required IDs

In order to connect any building and table we need to know their IDs. In order to do this we run the following.

SELECT * FROM buildings; and **SELECT * FROM departments;**

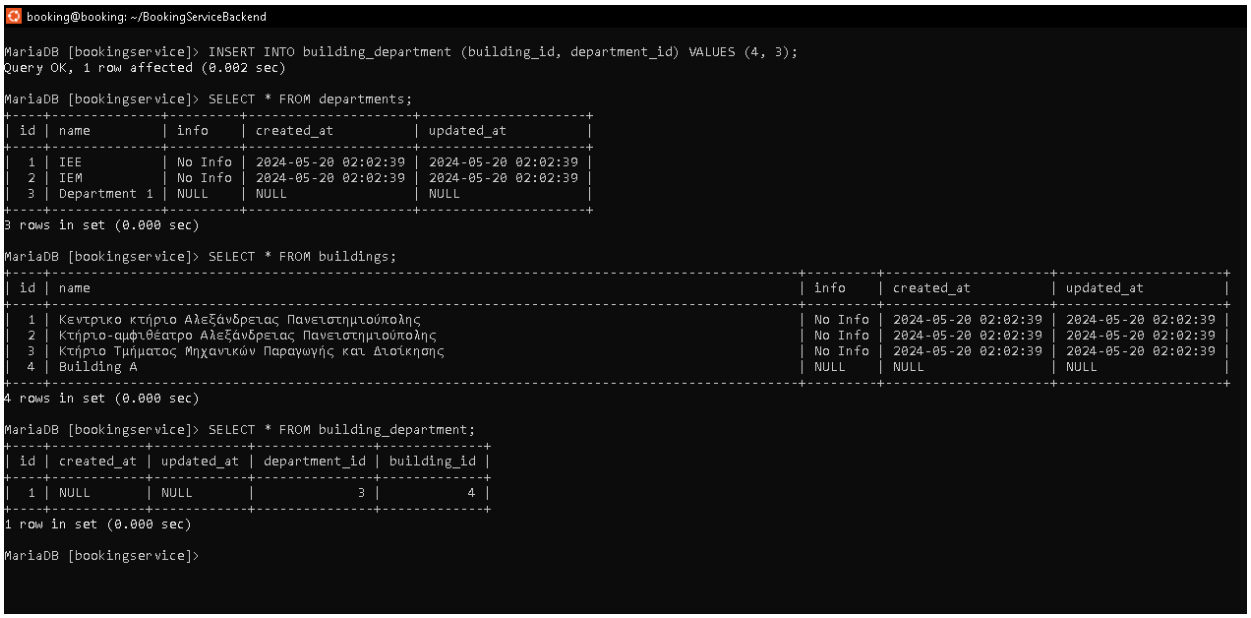


Then we can choose the entries we want to connect and continue to the next step.

B.6.4 Step 4: Insert Entry to Pivot Table

The actual connection is made using the pivot table. So we need to add an entry to this table using the following.

INSERT_INTO building_department (building_id, department_id) VALUES (4, 3);



Replace the values 4 and 3 with the IDs of the building and department that you want to connect.

Bibliography

[1] TatvaSoft, *Angular vs react vs vue: Which one to choose*, 2023. [Online]. Available: https://www.tatvasoft.com/blog/angular-vs-react-vs-vue/ (visited on 05/15/2024).

[2] Simplilearn, *What is angular?* 2024. [Online]. Available: https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular#:~:text=Angular%20enables%20users%20to%20build,plays%20well%20with%20web%20components.

[3] S. Daityari, *Angular vs react vs vue: Which framework to choose in 2023*, 2023. [Online]. Available: https://wpshout.com/angular-vs-vue-vs-react/ (visited on 05/15/2024).

[4] TatvaSoft, *Why use react?* 2024. [Online]. Available: https://www.tatvasoft.com/blog/why-use-react/#:~:text=It%20allows%20you%20to%20create,frequently%20used%20in%20web%20development. (visited on 05/15/2024).

[5] Altexsoft, *Pros and cons of vue.js*, 2024. [Online]. Available: https://www.altexsoft.com/blog/pros-and-cons-of-vue-js/ (visited on 05/15/2024).

[6] Curotec, *Laravel vs node.js - choosing the right framework*, 2024. [Online]. Available: https://www.curotec.com/insights/laravel-vs-node-js/ (visited on 05/15/2024).

[7] Laravel, *Laravel*, 2024. [Online]. Available: https://laravel.com/ (visited on 05/20/2024).

[8] GeeksforGeeks, *Why node.js?* 2024. [Online]. Available: https://www.geeksforgeeks.org/why-node-js/ (visited on 05/20/2024).

[9] Node.js, *About node.js*, 2024. [Online]. Available: https://nodejs.org/en/about (visited on 03/20/2024).

[10] B. Technology, *Why use laravel?* 2024. [Online]. Available: https://www.bacancytechnology.com/blog/why-use-laravel#:~:text=Laravel%20enables%20easy%20control%2Daccess,called%20the%20PHP%20authentication%20framework.

[11] Z. Powell, *Laravel vs node: A head-to-head comparison*, 2024. [Online]. Available: https://kinsta.com/blog/laravel-vs-node/ (visited on 05/15/2024).

[12] Microsoft, *Microsoft booking*, 2024. [Online]. Available: https://www.microsoft.com/el-gr/microsoft-365/business/scheduling-and-booking-app (visited on 02/20/2024).

[13] SuperSaaS, *Supersaas: Free appointment scheduling software*, 2024. [Online]. Available: https://www.supersaas.com/ (visited on 04/04/2024).

[14] Google, *Google calendar*, 2024. [Online]. Available: https://calendar.google.com/ (visited on 02/20/2024).

[15] L. Documentation, *Eloquent: Getting started*, 2024. [Online]. Available: https://laravel.com/docs/8.x/eloquent.

[16] L. Documentation, *Blade templates*, 2024. [Online]. Available: https://laravel.com/docs/8.x/blade.

[17] L. Documentation, *Http controllers*, 2024. [Online]. Available: https://laravel.com/docs/8.x/controllers.