



Assignment of master's thesis

| | |
|---------------------------------|---|
| Title: | Decentralized and Open Architecture of a Reservation System |
| Student: | Bc. David Straka |
| Supervisor: | doc. Ing. Tomáš Vitvar, Ph.D. |
| Study program: | Informatics |
| Branch / specialization: | Web Engineering |
| Department: | Department of Software Engineering |
| Validity: | until the end of summer semester 2023/2024 |

Instructions

Despite the increase of centralized solutions on the Internet today, there are many novel standards and technologies that go back to the original idea of the Internet and the Web in particular and use decentralized architectures. For example, Blockchain, Fediverse, and the most recent efforts around blockchain-less Web 3.0 are only a few examples of novel approaches whose main goal is to promote data privacy and security and enable users to have more control of their data. The goal of the thesis is to contribute to such efforts by developing a decentralized architecture for a reservation system. The thesis will fulfill the following tasks.

- Design an open and decentralized architecture for a general-purpose reservation system which should include a decentralized client-server communication protocol based on HTTP, and it should be possible to integrate the system with state-of-the-art cloud-native architectures.
- Develop a reference implementation of the architecture, including a client and a backend.
- Develop and implement a use case for a selected type of reservation business and discuss extensions of the architecture for the selected use case.
- Test and evaluate the reference implementation and the solution for the use case.



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Master's thesis

Decentralized and Open Architecture of a Reservation System

Bc. David Straka

Department of Software Engineering
Supervisor: doc. Ing. Tomáš Vitvar, Ph.D.

January 9, 2025

Acknowledgements

I would like to start by thanking my supervisor, doc. Ing. Tomáš Vitvar, Ph.D., for his guidance and support throughout my work on this thesis, as well as for the time he dedicated to the consultations needed especially during the formative stages of the process. Furthermore, I would like to thank my family and friends for their support. Last but not least, I want to express my gratitude to the authors of the many tools and libraries that were used during my work on the thesis, as well as to the open source community as a whole.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on January 9, 2025

.....

Czech Technical University in Prague
Faculty of Information Technology
© 2025 David Straka. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Straka, David. *Decentralized and Open Architecture of a Reservation System*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2025. Also available from: (<https://davidstraka.dev/master-s-thesis>).

Abstract

This thesis deals with the design, implementation, and testing of an open and decentralized architecture of a general-purpose reservation system. The thesis first features an overview of select existing reservation systems and a requirements analysis. It then designs a suitable architecture, booking client application, and its back end, which is designed to support a common HTTP-based booking protocol. Lastly, an implementation of the design is presented, along with the results of its testing and evaluation.

Keywords decentralized architecture, reservation system, decentralization, reservations, bookings, appointments, Web

Abstrakt

Tato práce se zabývá návrhem, implementací a otestováním otevřené a decentralizované architektury pro univerzální rezervační systém. V práci je zahrnut přehled vybraných existujících rezervačních systémů a analýza požadavků. Tato práce dále navrhuje vhodnou architekturu, klientskou rezervační aplikaci a její back end, který je navržen tak, aby podporoval společný rezervační protokol založený na HTTP. Nakonec je představena implementace návrhu spolu s výsledky jeho testování a evaluace.

Klíčová slova decentralizovaná architektura, rezervační systém, decentralizace, rezervace, schůzky, Web

Contents

| | |
|---|-----------|
| Introduction | 1 |
| 1 Existing Reservation Systems | 3 |
| 1.1 Acuity Scheduling | 4 |
| 1.2 Reservio | 6 |
| 1.3 Square Appointments | 8 |
| 1.4 Wix | 11 |
| 1.5 Summary | 14 |
| 2 Analysis | 17 |
| 2.1 Functional Requirements | 17 |
| 2.2 Non-functional Requirements | 19 |
| 3 Design | 21 |
| 3.1 Architecture | 21 |
| 3.2 Booking Protocol | 30 |
| 3.3 Booking Service | 34 |
| 3.4 Client Application | 36 |
| 4 Implementation | 37 |
| 4.1 Booking Service | 38 |
| 4.2 API Gateway | 39 |
| 4.3 Client Application | 40 |
| 4.4 Testing and Evaluation | 53 |
| Conclusion | 59 |
| Bibliography | 61 |
| A List of Acronyms | 65 |

B Contents of the Digital Attachment

67

List of Figures

| | | |
|------|---|----|
| 1.1 | Acuity Scheduling | 6 |
| 1.2 | Reservio | 9 |
| 1.3 | Square Appointments | 11 |
| 1.4 | Wix Events | 14 |
| 1.5 | Wix Bookings | 15 |
| | | |
| 3.1 | System network diagram | 27 |
| 3.2 | System conceptual schema | 28 |
| 3.3 | Architecture diagram | 30 |
| 3.4 | Detailed architecture diagram | 31 |
| 3.5 | Conceptual schema of the booking service database | 35 |
| | | |
| 4.1 | Diagram of the booking service database | 39 |
| 4.2 | Authentication page | 42 |
| 4.3 | Home page – Creation of booking address: input and validation . . | 43 |
| 4.4 | Home page – Creation of booking address: confirmation and error | 44 |
| 4.5 | Home page – Bookings overview | 45 |
| 4.6 | Home page – Open drawer and profile dialog | 46 |
| 4.7 | Home page – Item booking detail | 47 |
| 4.8 | New booking page | 48 |
| 4.9 | Items for booking page | 50 |
| 4.10 | Items for booking page – Item details | 51 |
| 4.11 | Business page | 52 |
| 4.12 | Add item page | 54 |
| 4.13 | Inventory item detail page | 55 |

List of Listings

| | | |
|-----|---|----|
| 3.1 | Definition of booking service info JSON using a TypeScript interface | 32 |
| 3.2 | Definition of Inventory resource JSON using a TypeScript interface | 33 |
| 3.3 | Definition of Booking resource JSON using a TypeScript interface | 33 |
| 3.4 | Definition of BookingWithItem resource JSON using a TypeScript interface | 33 |

Introduction

The online reservation service Reservio [1] claims that 70% of people prefer to book online, and providing online bookings leads to a 30% profit increase for businesses. One of the reviews showcased on their website [1] from the University Hospital Brno cites that these services save the hospital over 10 hours a week of administrative work.

A 2019 United States healthcare report by KPMG [2], on the other hand, claims that “most consumers prefer to book appointments by phone,” but it also states that about 40% are unable to do so on the first try and that 58% of millennials and 64% of people belonging to the generation X “value online booking to the extent that they would switch providers in order to do so.”

According to a 2014 study on the adoption, use, and impact of electronic booking in private medical practices in Canada [3], both patients and physicians showed growing interest in such system, “great majority of patients said that they appreciated the system mainly because of the benefits they derived from it, namely, scheduling flexibility, time savings, and automated reminders that prevented forgotten appointments,” and the study’s findings “suggest that the system’s automated reminders help significantly reduce the number of missed appointments.”

Yet, in practice, one can see many businesses still opting not to offer online bookings, instead relying mainly on phone calls or sometimes emails for reservations.

When businesses do offer online bookings, they tend to use a wide variety of different systems, each with its own user interface (UI) and user experience (UX). This can be confusing for the customers and can lead to a suboptimal UX due to, for instance, having to learn to use a new UI and familiarizing oneself with the features available within the system, having to manage many different user accounts’ login credentials (oftentimes leading to password reuse and thus also posing a security risk), and keeping track of all the bookings spread across the different platforms. Smaller booking systems also tend to struggle with handling surges in traffic. Because of the lack of standardization,

migrating from one system to another can prove to be a challenge for both the business and the customers, risking vendor lock-in.

An alternative to having many different booking systems could be one or a few of them becoming dominant within its market. This does have the advantages of a unified UX and fewer login credentials; however, there are many disadvantages and risks associated with such dominant platforms (often dominant to the point of them becoming monopolies or oligopolies). Such risks can be seen in many other types of online services, such as social media, search engines, and e-commerce platforms. Once these dominant platforms form, they can be very difficult for users to escape, for instance, due to the network effect where [4] “increased numbers of people improve the value of a good or service,” and which [4] also says may lead to less innovation.

With the rise of these dominant platforms, there has also been a growing number of efforts to create decentralized alternatives. Such decentralized systems often provide many of the advantages of a dominant centralized platform, such as being able to interact with a large network of users from a single client application with a familiar UI/UX and a single user account, but without many of the risks associated with the system being run by a single legal entity. An example of such an emerging decentralized system is the Fediverse. Some great examples of older decentralized online systems that are nowadays ubiquitous are email, and even the World Wide Web itself.

The goal of this thesis is to explore the possibility of creating a decentralized, general-purpose online reservation system by designing an open architecture of such system, including a common HTTP-based client-server protocol, and developing a reference implementation including both a client application and a back end. The created solution should be easy to integrate with state-of-the-art cloud-native architectures and should support a use case for a selected type of reservation business. Possible extensions for the selected use case should be discussed. The reference implementation should be tested and evaluated for the selected use case. Additionally, the thesis will explore select existing online reservation systems.

Existing Reservation Systems

This chapter serves as an overview of select existing online reservation systems and can be taken as a form of market research for the booking system to be created.

The overview is meant to be qualitative rather than quantitative, focusing on the exploration of features provided by a select few systems and the user experience while using those systems. The systems chosen for the overview are Acuity Scheduling¹, Reservio², Square Appointments³, and Wix⁴.

The criteria used to choose the systems for the overview are the following:

- **Popularity** — The system must be popular with the public. Without an extensive quantitative survey, this is a difficult criterion to evaluate. As a rough estimate, the system's ranking in the Google search results for the queries "online reservation system," "online booking system," and "online scheduling system" was used. If the systems appeared on the first couple of pages of the results, they were considered popular enough. Some flaws of this approach are, for instance, results personalization based on geographic location, as well as search engine optimization (SEO) techniques used by the systems. Similarly, different systems could have been discovered by using other search queries and search engines. However, despite the flaws of this approach, it still seemed to yield good enough results.
- **Localization** — The system must have English localization.
- **Price** — The system must offer a free version of its service (at least as a limited-time trial). Additionally, it must not require payment information to sign up for the free version.

¹Acuity Scheduling (<https://www.acuityscheduling.com>)

²Reservio (<https://www.reservio.com>)

³Square Appointments (<https://squareup.com/us/en/appointments>)

⁴Wix (<https://www.wix.com>)

1.1 Acuity Scheduling

Acuity Scheduling [5] describes itself as “an online appointment-booking tool” that “is great for any business that needs clients to book appointments for services in advance.” It also says that “businesses including yoga studios, massage therapists, acupuncture, life coaches, photographers, hair and nail salons use Acuity with great success,” but that it “is not a great fit for businesses that want to book appointments that last more than a day, like car rentals, vacation rentals, or hotels.” Acuity Scheduling is a subsidiary of Squarespace – a company that primarily focuses on providing a website-building service (for some time after Squarespace acquired Acuity, the scheduling product was rebranded to Squarespace Scheduling, but that change has since been reverted).

The website offers a free limited-time trial, after which it cannot be used without paying for one of its tiered plans. After first signing up for a business account, the user is asked for the name of their business and the name of an Acuity Scheduling subdomain that the service will create for the booking website of the business (this is optional, and if not provided, the website will be available under a shared subdomain and a URL query parameter with a generated identifier).

The user then creates their first appointment type by filling out the name of the appointment, its duration, whether it is a one-on-one appointment or a group appointment, and optionally, its price. If the user has selected the group appointment type, they must set the number of open slots per class as well.

Moreover, the user sets up the availability of the created appointment type by selecting a date and a time, whether or not it is a recurring event, and, if it is, the frequency of the recurrence (such as every Monday, every other Tuesday, the first Wednesday of each month, or daily), and the number of recurrences.

Lastly, the user can optionally connect third-party payment processors (Stripe, Square, and PayPal) for collecting deposits or full payments for the appointment bookings. Long-term subscriptions are also supported. The service claims [5] that it does not take any commission on the payments and that the only fees are those charged by the payment processor.

After the business user completes the sign-up process, customers can access a booking website that is created for the business. Upon accessing the website, customers see the business information (with only the sign-up completed, that is just the business name), followed by a list of appointments to choose from, with each appointment featuring its name, time and date, duration, and a sign-up button. If the business user selected the group appointment type, there is also a quantity field above the appointment list and the number of free spots available under each appointment’s sign-up button. Signing up for a selected appointment is followed by filling out personal information (a full name, an email address, and optionally a phone number). If there was no payment

required, the customer completes the booking and receives an email with the details of their booking, a link to reschedule or cancel the appointment, and links to add the appointment to their calendar. The booking page can be seen in the figure 1.1. The customer can use an account to manage their booked appointments, but it is purely optional, and they can still manage their bookings through the email links. There is also no way for the business to limit their appointments to only signed-in customers (the business can, however, ban clients by their email address).

The scheduling functionality can be integrated into any website built with Squarespace (which can then use a custom domain). There are also simple booking components that can be embedded into another website, in addition to the option to embed a booking page iframe. The booking website can be customized by choosing between a couple of predefined layouts and can also use custom CSS. The platform also features webhooks and an extensive application programming interface (API), which can be used to integrate it with other applications.

Business users can see all their appointments in a calendar overview, as well as all their clients. Under each appointment in the calendar, the user can see all signed-up customers, reschedule or cancel their bookings (when the business user does this, the affected customer can be notified by email, but there is also an option not to send this notification), and add private notes and tags for each customer. The user can manually add new attendees to each appointment (which can be useful, for example, when a customer wants to create a booking over the phone, but the business wants to have everything tracked in the booking system). There is also an option to send an email to all attendees of a selected appointment, in addition to SMS reminders. The business user can be notified of new activity by email as well, and they can synchronize their calendar with other popular calendar applications.

Appointment types can be further customized by adding a color and a picture. Users can make appointments private, in which case they are not visible to the public and can only be accessed through a direct link. There is an option to require clients to sign up for every recurrence of a recurring appointment and an option to disallow clients from booking multiple spots. Appointment types can also have a form added for the customers to fill out before booking an appointment. The form can consist of questions with answers of several different types, including a text field, a drop-down list, a checkbox, and even a file upload. The business information shown on the booking website can also be customized a bit further by, for instance, adding a logo, filling out rich content instructions for the customers, and changing to one of a few different languages.

Other features available include CSV import and export of clients and appointments, setting limits on how long before an appointment it can be canceled or rescheduled, and adding Google Analytics to the booking web-

1. EXISTING RESERVATION SYSTEMS

1 Choose Appointment

YOUR TIME ZONE

(GMT-4:00) Eastern Time

| TIME | | CLASS | PRICE |
|----------------------------------|--|---|----------|
| Thursday, May 11, 2023 TOMORROW | | | |
| 10:00am 30 minutes | no spots left | Community Classes: Yoga Wellness Studio | |
| 5:00pm 30 minutes | <div>Sign up</div> <div>4 spots left</div> | Nutrition: Fuel your workout Wellness Studio | \$100.00 |
| Friday, May 12, 2023 | | | |
| 10:00am 30 minutes | <div>Sign up</div> <div>8 spots left</div> | Crafting: Candle Making Wellness Studio | \$20.00 |
| Thursday, May 18, 2023 NEXT WEEK | | | |
| 5:00pm 30 minutes | <div>Sign up</div> <div>4 spots left</div> | Nutrition: Fuel your workout Wellness Studio | \$100.00 |

Figure 1.1: Acuity Scheduling [5]

site. A business account may be used by multiple users with separate login credentials and role-based permissions.

1.2 Reservio

On the front page of its website [1], Reservio describes itself as a “free on-line scheduling software for gyms, fitness centers, hair studios, barbershops, nail salons, car repair, medical services, teachers and educational institutions, group events, and more.”

When a user first signs up for a business account, they are asked to provide the following information:

- the name of their business;
- the type of their business, which can be either “Single appointments” (described as “the client books a service or an appointment at a specific

time”) or “Group events” (described as “host multiple clients at the same event, lecture or course”); to change this business type later on, one must contact customer support;

- opening hours, which “determine the exact range of when your clients can make bookings” and can also be customized per employee;
- the physical address of their business;
- optionally the contact phone number of their business;
- and optionally other information about their business – a website, a slogan, a description, and additional information about the physical address.

Users are also asked to add the services they would like to provide to their clients. Each service must have a name and a duration, and can optionally have a description, a price, a color (to visually differentiate it from other provided services), and an additional question to ask the customers (which can also serve as an input for a voucher code).

Lastly, users can also add staff members of the business who provide the previously added services, each with a name, a checklist of the services they provide, and optionally a short biography.

After the user completes the sign-up process, customers can access a booking website that is created for the business on a Reservio subdomain. When a customer accesses the booking website, they immediately see all the information about the business that was provided during the sign up. They then select a service for booking, a staff member (or “anyone available”), a date from a calendar picker, and a free time slot for the selected date. The customer can complete their booking either as a guest or by creating a Reservio account (or logging into an existing account). If the customer chooses to create an account, they can also see their booking history and cancel their bookings (in the settings, a business can set how long before the event can the bookings be canceled). The business user can also restrict in the settings who can create bookings to for example only existing clients or only clients with an account. As the last step of the booking process, the customer can answer the additional question that was added by the business, and they may also add a note. Afterwards, the customer is sent an email with the details of their booking.

For group events, the process is very similar, with the business user additionally specifying the capacity of an event. The customers can then book multiple spots for an event, as long as there is sufficient space available. Group events can also be marked as private by the business user, in which case they are not visible to the public and can only be accessed through a direct link.

The booking page iframe or simple booking components can also be embedded into another website. The booking website can be set as indexable

by search engines and can use a custom domain instead of the Reservio sub-domain (for premium accounts only). The layout and theme of the booking website can be somewhat customized as well, though most of the customizations are limited to premium accounts. There is an extensive API which can be used to integrate Reservio with other applications.

Business users can see their existing bookings in a calendar (also available as per staff member and per service views) which can be seen in the figure 1.2, as well as an overview of their clients. Premium users can have their calendar be synchronized with other popular calendar applications. Users can edit and cancel the existing bookings, as well as manually add new bookings and clients. Unreliable clients can be blocked and there is a CSV file import of clients as well. The amount of bookings a business can have is limited by different premium subscription plans.

Other available features include the ability to receive email and SMS notifications about new and canceled bookings (though SMS is limited to premium accounts), client reminders a set number of days or hours before a booked event, the option to limit the time of user data retention, and adding analytics to the booking website (such as Google Analytics and Meta Pixel; limited to premium accounts). Businesses can opt in to have their profile listed in a service marketplace (which customers can access through a separate application). One business account may be used by more staff members with separate login credentials and role-based permissions.

The platform also features online payments (which can be set as mandatory or optional), for which it takes a commission: This feature includes the handling of refunds when a customer cancels their booking and the possibility of long term memberships. However, according to Reservio [1], this feature is currently only available in the Czech Republic where the company originates and for security reasons it does not work when a custom domain is set up for the booking website.

1.3 Square Appointments

Square Appointments [6] is a product by the company Square (not to be confused with Squarespace, which is a different company), which primarily focuses on providing financial services to businesses (some other reservation systems mentioned in this overview even offer Square as a payment processor). Square Appointments [6] describes itself as “the all-in-one point of sale for booking, payments, and more.”

After first signing up for a business account, the service asks the user to provide some basic information: their personal name, business name, business phone number, time zone, and the type of the business. This information is later shown to the customers on an online booking site, if the business chooses to have one.

1.3. Square Appointments

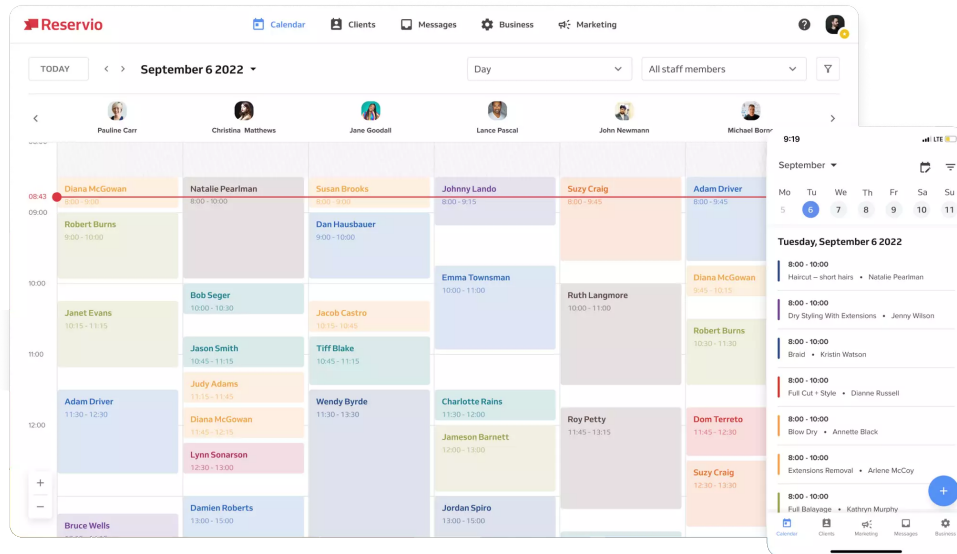


Figure 1.2: Reservio [1]

Square also asks for the number of staff members, business locations, and optionally services offered. The user also must fill out their estimated monthly revenue and optionally can input the average price per client. Then the user selects which features they are interested in, which can be any of the following: customizable online booking site, selling products, accepting payments, prepayment and no-show protection, and automated reminders and confirmations. For the purposes of this overview, the customizable online booking site, as well as the automated reminders/confirmations were chosen.

Square Appointments offers a feature-limited free plan (which is also limited to a single business location), as well as a limited-time trial of their lowest tier paid plan. The trial was used for this evaluation.

In the administrative dashboard, the user can edit their business location details, including its physical address, contact information, and social media links. They can add a short description of their business, a logo, and business hours.

The user then creates services that they offer, by filling out the service name, description, price, and duration. The service can have an image attached to it and its business location specified (when the business has multiple locations). A cancellation fee can be set up, and there can be extra time added to be blocked off after the service is done (for example for cleaning). The service can be categorized and team members can be assigned to it.

Finally, the user enables online bookings and can set up a customizable Square Online website with booking functionality built in. This website is

1. EXISTING RESERVATION SYSTEMS

published to a Square provided subdomain, and a custom domain can be set up as well (with a premium account). The booking functionality can also be embedded into an existing website using a button component or an iframe.

When a customer visits the booking website, they can see basic information about the business (such as its name and phone number), the services they provide, their staff, and locations including the opening hours. To create a booking, the customer selects the services they want to book, the date from a calendar picker, one of the available time slots from a list, then they fill out their personal information (phone number, email address, and full name) and optionally any note for the business. If the business sets up multiple staff members for a service, the customer may select a preferred staff member as well.

Upon booking, Square automatically creates an account for the customer, which they can sign in using the provided phone number (this cannot be skipped). The booking can be added to their personal calendar by one of several popular calendar services. The customer can reschedule or cancel their booking. The customer also receives an email with the booking details.

The business user can view their calendar, which includes the customers' bookings. They can view each booking's details, edit and cancel the booking. Bookings can be added manually as well. The business calendar can be synchronized with Google Calendar. The business calendar can be seen in the figure 1.3.

In the settings, the business user can also choose to require manually accepting all bookings. Time limits for scheduling can be set as well. The list of staff can be removed from the booking website. An interesting feature is the so-called "Fake-it Filter" which lets the business "remove some of their availability to give the appearance that their business is busier." Email and SMS reminders can be set up to be sent to the customers a given time before their appointment. There is no option to add a form to the booking process, but there is an option to create a form to send clients afterwards.

Other features include multiple team members using the business account with different permissions, a waiting list that automatically notifies clients of new availability, and an API. For some types of businesses, there is also a marketplace application from Square, where businesses can set up a profile for clients to discover. As square as a platform is primarily focused on offering financial services, there is a lot of detailed features related to this, including subscriptions management, thorough receipt customization, or various options regarding taxes, fees, and money transfers. Square also offers hardware for on location payments.

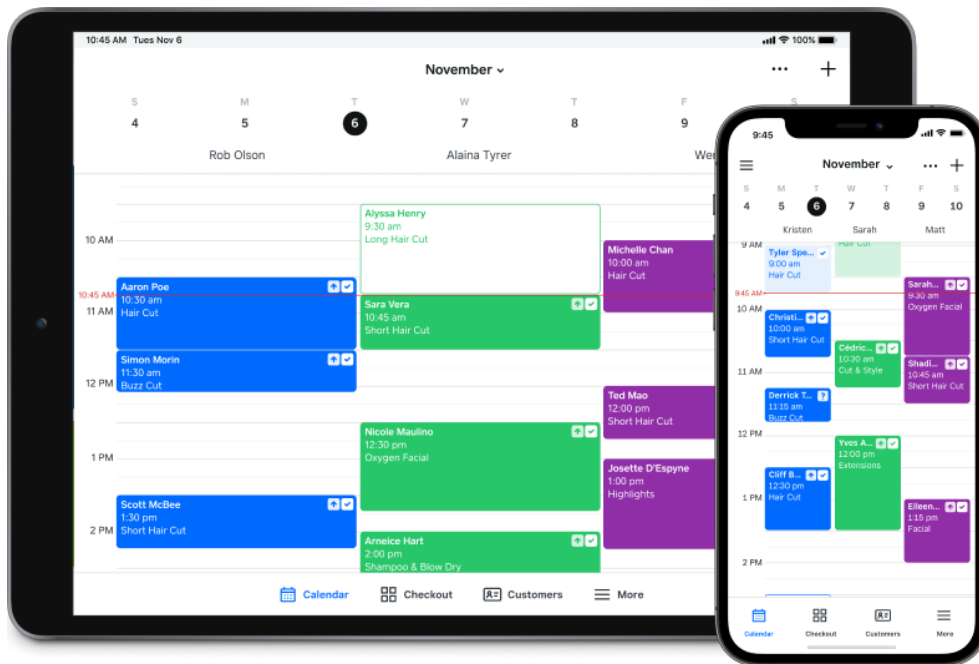


Figure 1.3: Square Appointments [6]

1.4 Wix

Wix [7], similarly to Squarespace (mentioned in a prior section of this overview), mainly offers a website building service which enables users to create their own websites using the Wix Website Builder UI. When building a website, users can make use of the so-called Wix Apps found in the Wix App Market. These can be thought of as plugins/extensions that the user can use to add functionality to their website or to the administrative dashboard. Wix Apps can be either official ones developed by Wix, or third-party ones (Wix provides an extensive API and even options to monetize the Apps published in the Wix App Market).

Wix offers three official Wix Apps which can in various ways be used to add reservation functionality to a Wix website: Wix Bookings, Wix Events, and Wix Restaurants. Wix Bookings seems to be meant for businesses offering frequently occurring appointments or classes. Wix Events appears more suited for those who organize events that only happen a few times or infrequently (Wix [7] lists “planning a wedding, hosting a convention, or selling tickets to a show” as examples). Wix Restaurants, as the name suggests, focuses purely on restaurant businesses, enabling them to for example showcase their menu, receive orders, but also to set up online reservations.

This overview focuses on Wix Events, which are available as part of Wix’s free plan (with some limitations), and briefly on Wix Bookings, which do not work with the website as part of the free plan, but do at least let users set up the administrative section.

After first signing up for an account, Wix asks the user a few questions about the purpose of their website to be built. Based on these answers it customizes the UI a bit, including adding suitable Wix Apps. When choosing the “Book Event” option as the primary focus of the website, the user is directed into the setup of the already installed Wix Events. They create their first event by entering the following:

- the name of the event,
- the type of the event – either a “Ticketed event” with options for pricing and availability or a “Registration only” event meant to collect confirmations of an invitation (RSVPs),
- the starting time and date of the event,
- optionally the ending time and date of the event,
- whether it is a recurring event (this lets the user add multiple dates/times for the event, but there does not appear to be an option to do this automatically),
- and optionally the location of the event (which can be a physical address or online).

The user can then add various tools to their website. These tools include for example a seating map which can “let guests choose their seats” or email marketing which can send invitations.

Lastly, the user can set up a custom domain to use for their website (though this is only available with a paid plan; otherwise the service provides a free Wix subdomain), design their website, and optionally optimize the website’s SEO. If the user selected the *Ticketed* event type, they also need to create ticket types for the created event to be bookable.

For designing the website, there are ready-made layouts which can be customized to a great extent, since that is Wix’s primary focus. The user can also let the service use artificial intelligence (AI) to generate the website layout for them. Since this overview’s goal is not to evaluate website builders, the AI option was chosen to save time. The resulting layout appeared fairly usable, though it seemed to have issues adjusting to different window sizes.

In the event details, the user can additionally provide a description and an image for the event. Events can be grouped into categories. Creating a ticket asks for the name of the ticket, optionally its description, the number of tickets

available, a pricing method, and a window of time when the ticket is available for purchase. Tickets can be limited to a maximum amount per single order.

The pricing method can be fixed to a certain amount, split into different categories (e.g. child, adult), “pay what you want,” and free. Wix lets users connect third-party payment processors to collect these payments (such as PayPal and Stripe, but the availability of the payment processors depends on the user’s country), and it charges a service fee (which is added on top of the payment processor’s fee).

Events can have forms attached, which can include fields of multiple different data types. With the seating map tool, the user can create a detailed map of the seats and areas available at the event venue. They can then assign the available tickets to specific seats or areas. This seating map from the customer’s perspective can be seen in the figure 1.4.

When all of this is set up, the user can publish the event on their website. Customers who access the website then navigate the event, see its details (such as the name, the time and date, and the location) and available tickets. If the window, during which tickets are up for sale, is open, they choose the ticket to buy by selecting the seat or admission area from the map (or, if no seating map was added, just choose which type of ticket to buy and its quantity), fill out any form the business user attached to the event, and check out. The customer can then download their tickets as a PDF document and add the event to their personal calendar. They also receive an email with the details of their booking.

The tickets that a customer receives include a QR code in them, which can then be scanned on location by the event organizer using a special-purpose mobile app from Wix. It does not appear that a customer can reschedule or cancel their booking, though the business user can do this for them (as well as cancel the event completely) through an overview of their orders and guests, though according to Wix [7], they will need to handle any refunds themselves. The business user can also add guests manually and later on manually check in any guest.

The Wix Bookings App adds a section with a booking calendar to the administrative dashboard. The user can add a service they would like to provide to their customers, which can be either an appointment (described as “a private session that can be booked according to availability”), a class (described as “a group session that can recur” where “clients book any session they want to join”), or a course (described as “a set of group sessions” where “clients book them all up front”).

In addition to providing some basic details about the service, the user can add pricing or a long-term membership plan, as well as staff members who provide the service. A booking form can also be attached. Unlike with Wix Events, the booking can be canceled or rescheduled by the customers, since there are options to disable this and limit the period of time before the

1. EXISTING RESERVATION SYSTEMS

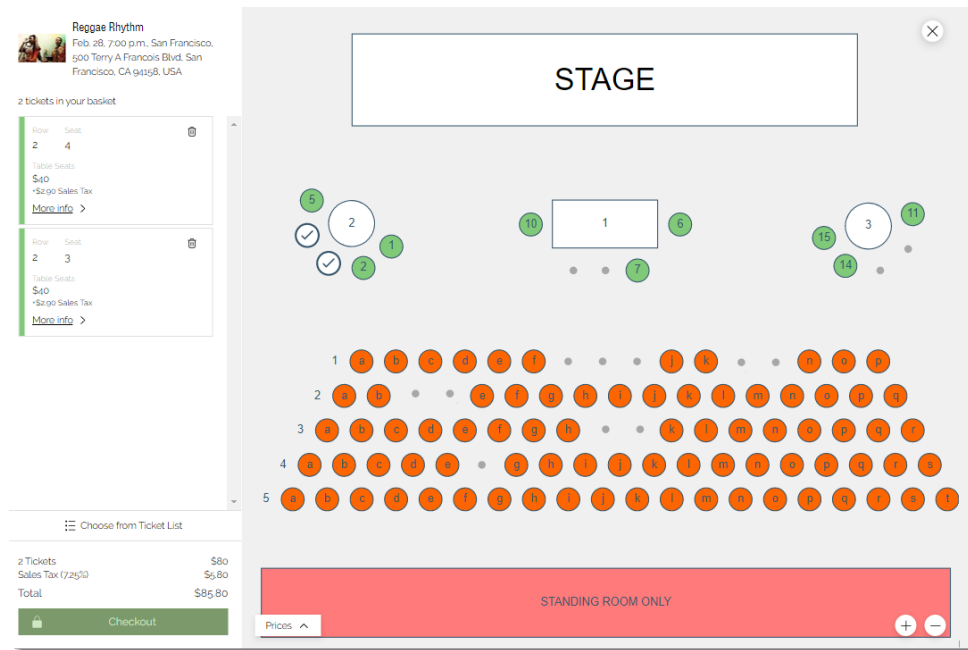


Figure 1.4: Wix Events [7]

booking when such changes can be made. Additionally, there is an option to require manually accepting all bookings.

The business user also sets up their opening hours, so that the service can offer recurring classes during these times (unlike with Wix Events, where recurring events had to have their times and dates added manually). A useful feature is also the option to schedule the appointment time slots either based on the service duration or every fixed amount of time. Booking reminders through email and SMS, calendar synchronization, and waiting lists are available as well. An example of a Wix website with the Wix Bookings functionality can be seen in the figure 1.5.

Other features of the platform include built-in website analytics, multiple staff members managing a single account with role-based permissions, business website localization, CSV export of orders, and a full-featured API.

1.5 Summary

To summarize this overview, among the four online reservation systems included in the overview – Acuity Scheduling, Reservio, Square Appointments, and Wix – there were many similarities and common features.

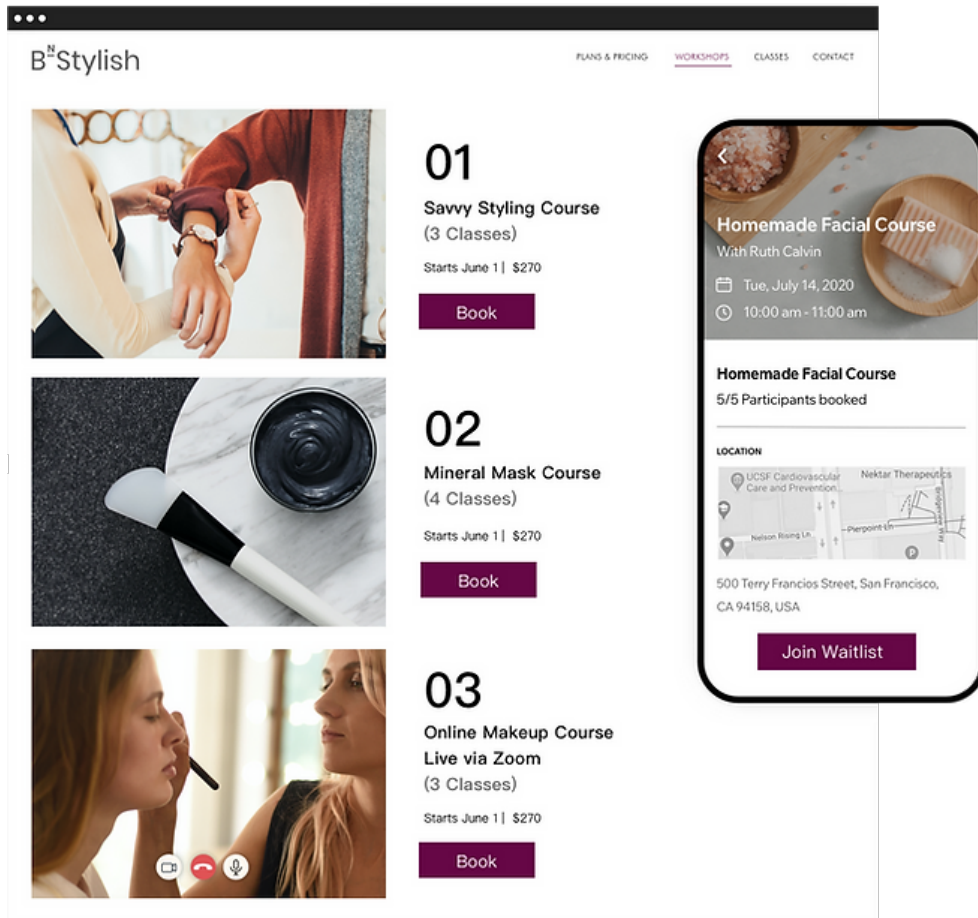


Figure 1.5: Wix Bookings [7]

One of the similarities was the overall process of setting up a business account, filling out basic information about the business and setting up certain services, classes, or events for the clients to book. After this, the system would let the users create a booking website under the subdomain of the service or under a custom domain (which was always a premium feature). Customers could then access the website, view the information about the business, the services or events provided, and create a booking.

All systems also featured some options to accept payments, in addition to sending out emails with booking details and reminders. There was always some possibility to embed components or iframes with booking functionality to other websites, and an API for developers to integrate the system with other software (the API was always platform specific though). Calendar syn-

1. EXISTING RESERVATION SYSTEMS

chronization and multi-user accounts (with adjustable permissions) were also common features.

The main differences (excluding pricing strategies and the UI) seemed to be the ability to add forms for users to fill out during booking (present in Acuity Scheduling and in Wix's solutions), the ability to create a custom venue map with seating and areas sold under different ticket types (present in Wix Events).

Those platforms that focus on building websites (Wix and – the owner of Acuity Scheduling – Squarespace) offer more features for extensive booking website customization. On the other hand, Square Appointments from Square who focuses on financial services, offers the most features for payment processing, customer subscriptions, and financial reporting. Finally, Reservio seemed to offer subjectively the simplest business account setup and UI.

Analysis

This chapter includes the analysis of functional and non-functional requirements for the system to be designed and implemented.

The requirements are based on both the assignment of the thesis, the overview of select existing reservation systems in the previous chapter 1, and on original ideas for the new system.

2.1 Functional Requirements

The functional requirements for the system are the following:

- F1 User account** — A user without an account must sign up for one to use the system. A user with a user account must be uniquely identified in the system. A user can log into their account and log out of their account.
- F2 Inventory** — A user can create an inventory of items for users to book. An inventory can have some basic information attached (such as a name, a description, and contact information). Items can also have information attached, and can be booked by multiple users up to a given maximum capacity.
- F3 Booking** — A user can view an inventory including its items of a user with a created inventory. A user can book an item from an inventory. A user can view their existing bookings.
- F4 Forms** — A user with an inventory can attach a form to the inventory. A user who wants to book an item from an inventory with a form must fill out the form.
- F5 Deadlines** — A user with an inventory can limit until when an existing booking of an inventory item can be booked.

- F6 Permissions** — A user with an inventory can limit who can book items from their inventory by creating either an allowlist or a denylist of users.
- F7* Booking cancellation** — A user can cancel their existing booking. A user with an inventory can cancel an existing booking of item from their inventory as well.
- F8* Booking edits** — A user can edit their existing bookings' form data.
- F9* Booking ownership verification** — A user can easily verify in person that another user is the owner of a booking (for instance by scanning a QR code).
- F10* Real-time item availability updates** — A user can see real-time availability updates of a user with an inventory of items.
- F11* Inventory filtering** — A user can request a user's inventory of items filtered by item attributes.
- F12* Custom domain** — A user can use a custom domain for their identification in the system without setting up their own server.
- F13* Data export and import** — A user can export and import their data in an open, standardized, machine-readable data format.
- F14* Inventory directory** — A user with an inventory set up can opt in to have their inventory listed in a publicly accessible directory of user inventories. A user can access the directory of user inventories and filter/sort it by inventory attributes, or search it using natural language queries.
- F15* Calendar synchronization** — A user can have their bookings with date and time attributes synchronized with their personal calendar. A user with an inventory set up can have their inventory items with date and time attributes synchronized with their business calendar.
- F16* Notifications** — A user can receive email (and possibly other types, such as SMS) notifications about their bookings and inventory items.
- F17* Waiting list** — A user can enter a waiting list for an item that is fully booked. When a booking of that item is cancelled, or the user who the item's inventory belongs to increases the capacity, the item is automatically booked for the first users on the waiting list up to the newly available capacity.
- F18* Payments** — A user with an inventory can require payments before or after a booking is made. A user can make a payment for a booking they made or are about to make, if that booking requires it.

- F19*** **Multi-user account** — A user account can be used by multiple users. The user that first creates a user account can manage who else can use the account, and can limit their permissions.
- F20*** **Internationalization** — A user can choose the locale (language, currencies, date and time formats, etc.) they want to use of the system under. A user with an inventory can set up their inventory with multiple different locales.
- F21*** **Contacts** — A user can add users with inventories to their contacts list. A user can view their contacts list and use it to quickly access the inventories of their contacts, as well as view the bookings made for the items of each individual contact.
- F22*** **Autofill** — A user can have their personal information automatically filled into forms during booking. This personal information can be manually entered by the user into a settings section, or automatically saved from previously filled-in forms.
- F23*** **Automatic reservations** — A user can instruct the system to automatically book selected items from selected inventories for them. Such instructions can for example be to book any appointment-type item at the beginning of every month from a given inventory, and that the event item must occur within a given time range.

This list also includes some requirements that are out of scope for the initial version of the system, but are included for the sake of completeness. These requirements will not be further designed or implemented, but may have some impact for instance on the design of the system, the choice of technologies used, or certain implementation details. Such requirements are marked with an asterisk (*). Additionally, the item **F6** will be only only designed, not implemented, to keep the scope of the project to a reasonable degree.

Most of the functional requirements stem from features already present some or all in the reservation systems previously explored. Optional requirement **F23** is an original idea, that would could make use of the strengths of the newly designed system.

2.2 Non-functional Requirements

The non-functional requirements for the system are the following:

- N1 Decentralized architecture** — Users of the system can book from users regardless of the exact server or client application they use. Users can also set up their own server and client application to use the system.

- N2 Cloud-native** — The designed and implemented solution is containerized for easy deployment to a cloud platform and can be scaled well. The part of the solution that is responsible for the booking logic can be easily integrated into another solution.
- N3 General-purpose** — The designed and implemented solution should try to be as general-purpose as possible, not rigidly built around a very specific use case.
- N4 Extensibility** — The designed and implemented solution can be extended both in terms of adding completely new functionality and extending existing functionality to better fit other use cases.
- N5 Backward compatibility** — Users of the system can book from users using servers that implement older or newer designs of the system (with the functionality limited to what both parties support).
- N6 Fast response times** — As reservations in general can oftentimes be very time-sensitive, the system should be designed in such a way that does not introduce unnecessary delays to the booking process.
- N7 Platform-agnostic** — While testing the designed and implemented solution on a single platform should be sufficient for the purposes of this thesis (especially considering it is supposed to be a prototype and not a production-ready product), the solution should not be limited by its design or implementation to certain processor architecture, operating system, or a web browser (within reason, taking into account the commonplace platforms). An exception can be made for new standardized web technologies that are not supported by all platforms yet, as long as those technologies only add additional features.
- N8 Open source** — The designed and implemented solution is open-source, enabling anyone to freely use and repurpose it for their own needs.

These requirements mostly originate from the assignment of this thesis, and in many cases present an improvement over existing reservation systems.

Design

This chapter describes the design of the proposed and implemented system. First it details the architecture, including the architecture of the decentralized booking system overall, the format of the booking addresses which are used to uniquely identify users within the system and the Uniform Resource Locator (URL) scheme which is to be used for distinguishing the booking addresses from other similar identifiers, and the architecture of the implementation in this thesis. Then the chapter describes the HTTP-based booking protocol which is used to communicate between different booking services. Lastly, the chapter describes the design of the individual back-end services and client application.

3.1 Architecture

This section first describes the architecture of the entire decentralized booking system, then later defines the format of booking addresses and their URI scheme, and lastly it focuses more on the architecture of the solution that is implemented in this thesis.

3.1.1 System architecture

Like many other decentralized systems, such as email, the Fediverse or Bluesky, the decentralized booking system shall use a federated architecture, which comprises of a network of several booking services operated by different booking service providers. These booking services shall communicate with each other using a common booking protocol, and interface with the end users through booking client applications provided by the booking service provider operating the booking service. Within the system, end users shall be uniquely identified by a booking address provided by their booking service provider. An end user can not only be a human, but also a bot, used to perform for

instance automatic bookings, though some booking service providers and their users may opt to take measures to prevent bot traffic.

The common booking protocol shall be an application layer (as defined by [8]) protocol built on top of the HTTP protocol (as per the thesis assignment). It shall be designed to be backwards-compatible, so that new features can be added without breaking compatibility with older versions of the protocol, and to be extensible, so that features useful to only some booking use cases can be added without affecting the rest of the protocol. The protocol's design is described in more detail in the section 3.3.

While the main intention is to design a system that is as interoperable as possible, some booking service providers may choose to ban traffic from other booking service providers if they act in malicious ways, and bookable users may choose to ban (or only allow) certain users as well. Booking service providers may even choose to completely isolate themselves from the rest of the system on the internet, and instead provide booking services on a private network only (this may be desirable for instance in some corporate environments).

A broad system network overview is shown in the figure 3.1. The figure 3.2 then shows a detailed conceptual schema modelling the entities of the system and their relations.

The conceptual schema contains the following entities:

- ***BookingProtocol*** — This entity represents the booking protocol used for communication between booking services in the system. It has the attribute *version*, which is used to version the protocol in accordance with the principles of semantic versioning, described by [9]. Such version shall, as [9] says, be in the format *MAJOR.MINOR.PATCH*, where a *MAJOR* version shall be incremented “when you make incompatible API changes,” a *MINOR* version shall be incremented “when you add functionality in a backward compatible manner,” and a *PATCH* version shall be incremented “when you make backwards compatible bug fixes.”
- ***BookingProtocolExtension*** — This entity represents any extension created for the booking protocol. It has the attributes *id* and *version*. The identifier *id*, the entity's primary key, is used to uniquely identify the extension, and it shall be an Internationalized Resource Identifier (IRI), as defined by [10]. The *version* is used to version the protocol extension, again in accordance with the principles of semantic versioning.
- ***BookingService*** — This entity represents a booking service, that is a server used to facilitate booking activities with other booking services in the system, using the booking protocol.

- ***BookingServiceProvider*** — This entity represents a provider of a booking service. Such provider is any individual or a company, that operates a booking service and issues booking addresses to its users.
- ***BookingClientApp*** — This entity represents a booking client application, that the end users use to interact with the booking service. Such application can be for instance a web application, a mobile application, or a desktop application with a graphical user interface (GUI), or even just a simple command-line interface (CLI).
- ***User*** — This entity represents an end user of the system. That can be both a human or a bot (or even a legal entity). One single person can also be represented as multiple users in the system, for instance if they use services of multiple booking service providers. This entity has the attribute *bookingAddress* which is used to assign the user their system-wide unique booking address, and also serves as the entity's primary key. The format of the booking address is defined later on in this section.
- ***BookableUser*** — This entity extends the *User* entity and it represents a user who has an inventory of items, that other users of the system can book.
- ***Inventory*** — This entity represents the inventory of a bookable user – that is, again, a user who wants other users to be able to book items from them. The entity may have the attributes *permissionMode*, *metadata*, *form*, *itemType*, and *itemMetadataSchema*.

The attribute *permissionMode* is used when the inventory features either a list of banned users who may not book items of the inventory or a list of allowed users who are the only ones permitted to book the inventory's items (which one of these options is used is determined based on the value of the attribute).

The attribute *metadata* is used to store arbitrary metadata about the inventory, such as contact information, opening hours, or announcements and notes for customers. To provide machine-readable semantic meaning to the metadata as well as the ability to link the data to other datasets, the JSON-LD format shall be used, as defined by [11, 12].

The attribute *form* is used to store a JSON Schema (as specified by [13]) which defines the structure of the data that the bookable user wants users to fill out when booking an item from the inventory. According to [14], the defined JSON Schema can then be used to both validate the data that the user fills out and to generate UI of a form for the user to fill out. JSON Schema is used by many widespread production-ready systems and utilities, including for example OpenAPI ([15] says that

“OpenAPI 3.0 uses an extended subset of JSON Schema (...) to describe the data formats”) and in turn also Kubernetes (which according to [16] for instance uses OpenAPI 3.0 to allow the definition of custom resources).

The attribute *itemType* is used to define the type of the items in the inventory. This is done using an IRI, that will also be used as the `@type` directive for the JSON-LD metadata of the inventory items (thus reducing verbosity of the data). This information could then be also used by booking client applications to provide a better user experience, for instance by providing a more specific UI for booking items of a certain type.

Lastly, the attribute *itemMetadataSchema* is used to store the JSON Schema of the metadata of the individual items in the inventory. Since this is defined on the inventory level, booking services and booking client applications can know which metadata to expect from all items even when using pagination for large collections of items.

- **Item** — This entity represents an item in the inventory of a bookable user. Such item can for example be a one-time doctor appointment, a concert, a recurring language class, or even a physical item to be reserved in a store. The entity has the attributes *uuid*, *createdAt*, *capacity*, *bookDeadline*, and *metadata*.

The attribute *uuid* is a random version 4 universally unique identifier (UUID) as defined by [17]. It is used to uniquely identify the item in the system, and it is also the entity’s primary key.

The attribute *createdAt* is a timestamp of the date and time when the item was created, including the time zone, in the standardized ISO 8601 format, as defined by [18].

The attribute *capacity* is an integer optionally used to store the maximum number of bookings that can be made for the item. It is used to prevent overbooking. If the attribute is not set, the capacity is considered to be unlimited.

The attribute *bookDeadline* is a timestamp of the date and time starting from when the item can no longer be booked, including the time zone, in the standardized ISO 8601 format. If the attribute is not set, the item can be booked indefinitely.

Lastly, the attribute *metadata* is used to store arbitrary metadata about the item, such as a description, a price, a date and a time, or a location. To provide machine-readable semantic meaning to the metadata as well as the ability to link the data to other datasets, the JSON-LD format shall be used.

The item's metadata must adhere to the JSON Schema defined in the *itemMetadataSchema* attribute of the item's inventory. The JSON-LD context of the item's metadata has **implicitly** included the `@vocab` directive included in the inventory's *itemType*. Aside from the informational value of such item metadata for end users, it could also be used by booking client applications to filter and sort items.

- **Booking** — This entity represents a successful booking of an item held by a user. It has the attributes *uuid*, *createdAt*, and *formData*. The attribute *uuid* is a random version 4 universally unique identifier (UUID), used to uniquely identify the booking in the system, and it is also the entity's primary key. The *createdAt* is a timestamp of the date and time when the booking was created, including the time zone, in the standardized ISO 8601 format. The *formData* is used to store the data that the user filled out when booking the item, and it must adhere to the JSON Schema in the inventory's *form* attribute. If the attribute is not set, there was no form to fill out.

As for the relations between the entities, there are the following:

- A *BookingProtocol* entity can be implemented by any number of *BookingService* entities and a *BookingService* can implement one or more *BookingProtocol* entities, due to the possibility of supporting multiple protocol versions at once.
- A *BookingProtocol* entity can be extended by any number of *BookingProtocolExtension* entities and a *BookingProtocolExtension* can extend one or more *BookingProtocol* entities, again, due to possibly extending multiple protocol versions.
- A *BookingService* entity can support any number of *BookingProtocolExtension* entities and a *BookingProtocolExtension* can also be supported by any number of *BookingService* entities.
- A *BookingServiceProvider* entity provides one or more *BookingService* entities for their users and a *BookingService* is provided by one *BookingServiceProvider*.
- The relation between the *BookingServiceProvider* and *BookingClientApp* is analogous to the one before.
- A *BookingClientApp* serves as a UI to one *BookingService* and one *BookingService* can be served by any number of *BookingClientApp* entities. A typical example of a situation with multiple booking client applications per booking service is when a booking service provider provides, for instance, both a web application and native mobile applications for different platforms.

3. DESIGN

- Similarly, a *User* entity can use one or more *BookingClientApp* entities and a *BookingClientApp* can be used by any number of *User* entities.
- A *User* entity is registered with one *BookingServiceProvider* who gives them their booking address and one *BookingServiceProvider* can have any number of *User* entities registered with them. This includes such cases like a new provider who may not have any users yet, and a user with their own deployed booking infrastructure and a domain name being their own provider without providing services to anyone else. Note again that a *User* entity only represents a person, a bot, a legal entity, or a group under one account, so this does not mean that a person cannot have accounts with multiple providers.
- A *BookableUser* entity creates one *Inventory* entity and the *Inventory* is only created by that one *BookableUser*.
- An *Inventory* entity can ban/allow any number of *User* entities from/for booking its items and a *User* entity can be allowed or banned by any number of *Inventory* entities.
- An *Inventory* entity can contain any number of *Item* entities and an *Item* entity is contained only by that inventory.
- An *Item* can be in an exclusive (XOR) relation with any number of other *Item* entities and vice-versa. When two *Item* entities have the XOR relation between them, it means that booking one of the items also counts towards the capacity of the other item. This can be useful, for instance, when a business is offering multiple different services at a time and has only one staff member carry out the services.
- An *Item* entity can be booked by any number of *Booking* entities (up to its capacity) and a *Booking* entity books one *Item* entity.
- Lastly, a *User* can hold any number of *Booking* entities and a *Booking* entity is held by a single *User* entity.

3.1.2 Booking address format

To uniquely identify users in the system, a booking address shall be used. The booking address is a string that is unique within the system, and it is registered with the user's booking service provider (that being anyone who is an owner of a domain name and runs infrastructure implementing the booking protocol). This is a very familiar idea to that of email addresses and the format will look very similar.

The booking address shall comprise of a username and either a domain name or an Internet Protocol (IP) address, separated by the *at* (@) sign. At

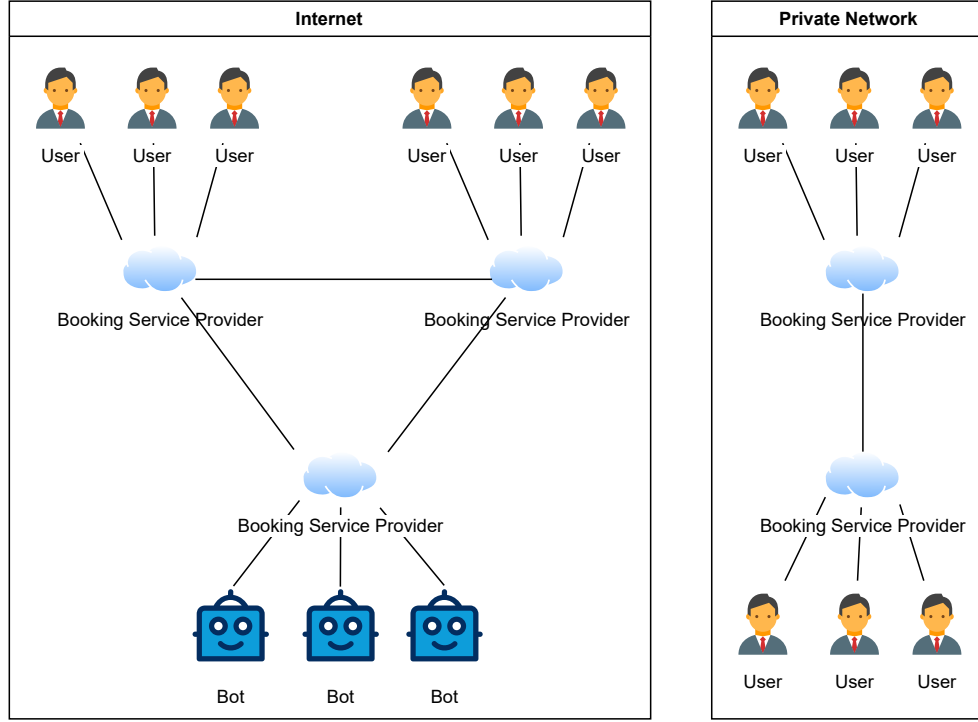


Figure 3.1: System network diagram

the end of the string, a port may also be specified, separated by a colon. The default port for the booking protocol shall be 3080, and it is not necessary to specify it in the booking address if it is the default.

The port 3080 is, according to [19], not restricted in browsers and, according to [20] is in the range of “User Ports” which range from 1024 to 49151, as opposed to “System Ports” which are below this range, and “Dynamic Ports” which are above the range and are never assigned by the Internet Assigned Numbers Authority (IANA). Searching for the port usage using search engines and using IANA’s Service Name and Transport Protocol Port Number Registry [21] has not shown too widespread use of the port number.

The booking address shall be case-insensitive, to prevent user errors. The username part may contain the 26 (case-insensitive) Latin letters, Arabic numerals, and the following special characters: the period (.), the hyphen/minus character (-), and an underscore (_). The username can be expressed by the regular expression `[0-9A-Za-z._-]+`. The entire booking address can then be expressed as `<username>@<ip_address>[:port]` and `<username>@<domain_name>[:port]`. Example valid booking addresses include `alice@127.0.0.1`, `bob@localhost:3080`, and `john.doe98@example.com`. Example booking addresses that are considered

3. DESIGN

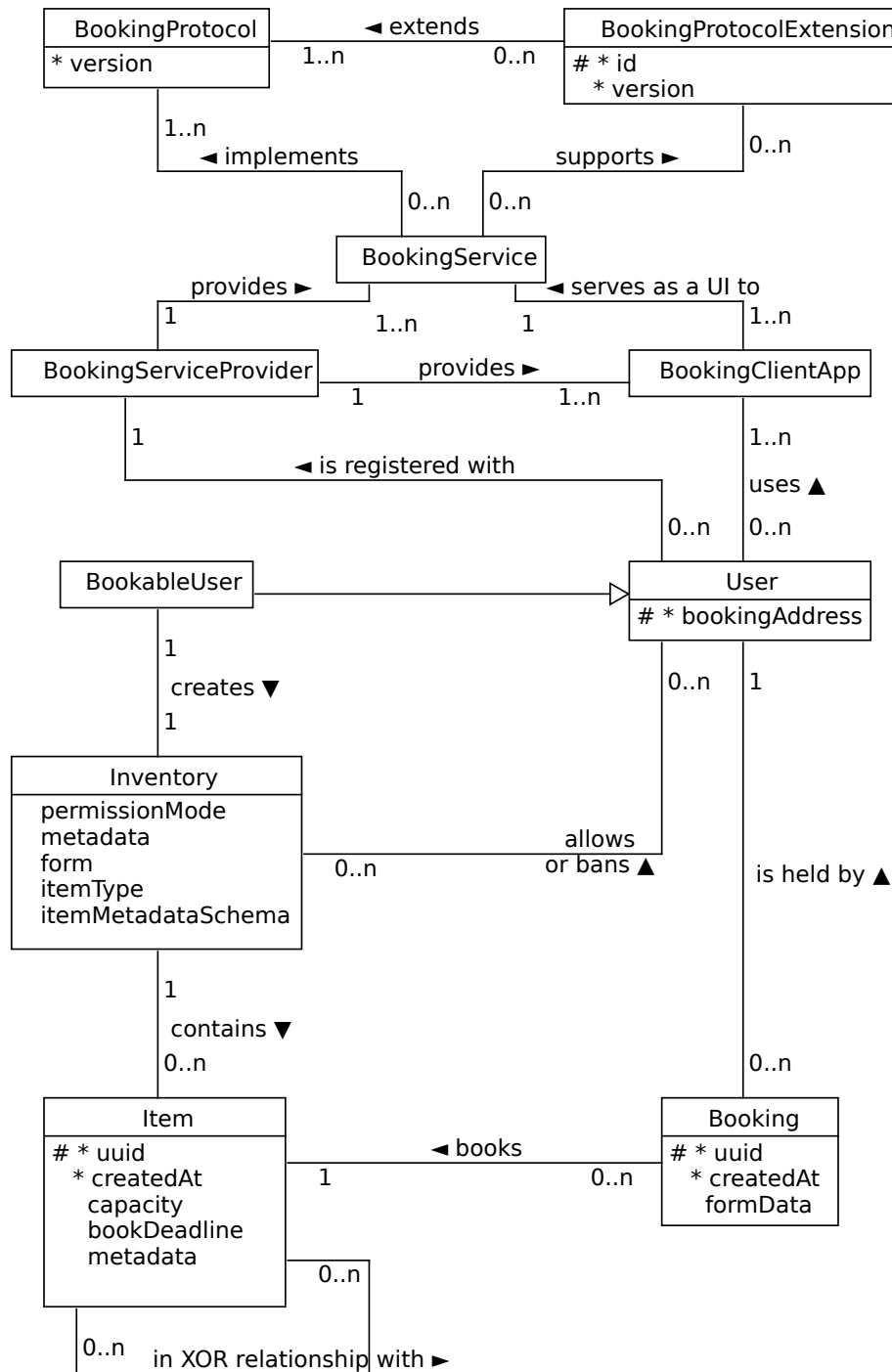


Figure 3.2: System conceptual schema

equivalent are `john.doe98@example.com`, `john.doe98@example.com:3080`, and `John.Doe98@example.com`. The maximum length of the username part shall be 64 characters (similarly to the limit of the local part of an email address, as defined by [22]).

3.1.3 Custom URL scheme

According to [23]: “A Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an abstract or physical resource. Each URI begins with a scheme name that refers to a specification for assigning identifiers within that scheme. As such, the URI syntax is a federated and extensible naming system wherein each scheme’s specification may further restrict the syntax and semantics of identifiers using that scheme. Schemes are also known as protocols.” Not to be confused with the actual communication protocol. URLs are a subset of URIs that specifies the location of a resource on the internet.

In order to differentiate booking addresses from other identifiers when shared around the Web and enable users to assign their booking client application of choice to handle activated booking address hyperlinks, a custom URL scheme is also designed. [23] states that in order for a progressive web application (PWA) to register their ability to open or handle particular URL schemes, the scheme must either be one of a few safelisted schemes or be prefixed with `web+`. Of course, this does not restrict non-PWA applications from handling the scheme.

In order to future-proof the system and allow the increasingly popular PWAs to handle the scheme, the booking system shall use the scheme `web+booking`. Searches made using search engines for the exact phrase have not revealed any existing use of the scheme. The scheme shall be followed by a booking address and, optionally, any of the following (in the order listed): a path separated from the booking address by a forward slash (/), query parameters separated from prior URI segments by a question mark (?), separated from each other by an ampersand (&) and separated from their own values by an equals sign (=), and lastly a fragment separated from prior URI segments by a number sign (#), much like in the case of URLs.

The path, query, and fragment segments will not be utilized in the initial version of the booking protocol, but extensions of the protocol as well as newer protocol versions may utilize them in the future. As an example of possible future utilization of these segments, path and fragment could be used to direct the client application to a certain section and the parameters could be used to provide additional functionality, such as form filling with predefined values or item filtering and sorting.

Because of limitations of URL format, unlike in URIs, the *at* (@) sign in cannot be used in the booking address in the URL, and using a percent encoding would mean double encoding. Therefore, specifically in

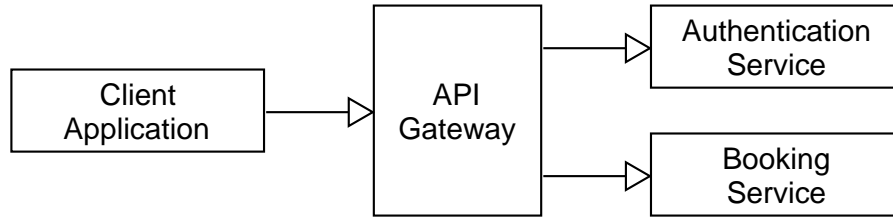


Figure 3.3: Architecture diagram

booking schema URLs, it shall be replaced with the *tilde* (~) character, which can be used and easily separated from the rest of the URL.

The URL scheme can then be expressed as `web+booking://<booking_address>[/path][?query][#fragment]`. Example URIs include the following:

- `web+booking://alice~127.0.0.1,`
- `web+booking://bob~localhost/a,`
- `web+booking://charlie~localhost:3080?q=val,`
- `web+booking://dan~localhost:8000#section,`
- and `web+booking://john.doe98~example.com/a?b=1&c=2&d=3#e.`

3.1.4 Implementation architecture

The implementation will use the microservice architecture API gateway pattern, with each service being containerized. A diagram illustrating the architecture is shown in the figure 3.3. The figure 3.4 then shows a more detailed diagram of the architecture featuring the technologies chosen for each service (which are later described in the section 4).

This will enable more efficient scaling of the individual services, as well as easier replacement of one individual service, if for instance a booking service provider decides to use their own implementation of only one of the services. It is also in accordance with the non-functional requirement **N2**.

3.2 Booking Protocol

The booking protocol shall build upon the HTTP protocol. Because the booking protocol shall handle transfers of lots of personal data, outside of localhost

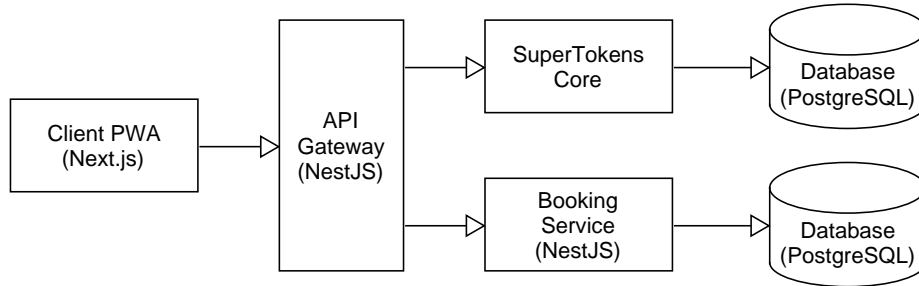


Figure 3.4: Detailed architecture diagram

the secured HTTP variant HTTPS shall be used (which is also enforced by some web browsers). The booking protocol shall use JSON as the data format for all requests and responses. In the headers of every request and response, two header items shall be present: **Booking-address** and **Booking-Service-Info**.

The **Booking-Address** header shall contain the booking address of the user who (or on whose behalf was) made the request. The receiving booking service shall then verify, that the request is coming from a server, whose IP address belongs to the domain of the booking address (if it uses a domain and not an IP address) according to public DNS records. This can be done using reverse DNS queries. If the booking service is unable to verify the booking address, it shall throw an HTTP 401 error with an informative message.

The **Booking-Service-Info** header shall contain base64 encoded JSON string, which contains information about the booking service, namely its supported protocol version and protocol extensions (and their versions). The receiving booking service shall throw an HTTP 400 error with an informative message, if it is not compatible with the requesting booking service (or if the request is missing the header). The decoded and parsed object can be described by a TypeScript interface as shown in the listing 3.1.

Furthermore, each request (not response) shall contain the **Host** header, with the host of the target booking service, so that the booking service, which can be hosted on multiple domains under one IP address, can determine the target booking address.

The booking protocol shall use the following two endpoints:

- **GET /users/:username** – Get a bookable user’s inventory with items.
 - The path parameter **:username** shall contain username part of the booking address of the user whose inventory is requested.

3. DESIGN

```
interface BookingServiceInfo {  
  version: string; // Resource SemVer version  
  extensions?: {  
    [key: string]: string; // Pairs of extension ID  
                          // and its SemVer version  
  };  
}
```

Listing 3.1: Definition of booking service info JSON using a TypeScript interface

- Respond with HTTP 200 and the **Inventory** resource with its items included and item occupancy (number of bookings of the item) calculated. The resource can be described by a TypeScript interface as shown in the listing 3.2.
- Respond with HTTP 404 if the booking address requested by the combination of the username from the path and host from the Host header does not exist.
- **POST /users/:username/bookings** – Create a booking for an item from a bookable user’s inventory.
 - The path parameter **:username** shall contain username part of the booking address of the user whose item is being booked.
 - The request body shall contain the **Booking** resource with the booked item’s id. The resource can be described by a TypeScript interface as shown in the listing 3.3.
 - Respond with HTTP 201 and the **BookingWithItem** resource with its item included and item occupancy (number of bookings of the item) calculated. The resource can be described by a TypeScript interface as shown in the listing 3.4.
 - Respond with HTTP 400 if the booking is after book deadline, fully booked, or the given form data does not match the JSON schema in the item.
 - Respond with HTTP 404 if the booking address requested by the combination of the username from the path and host from the Host header does not exist, or if the item requested for booking does not exist.
 - Respond with HTTP 409 if the booking already exists.
 - Note that it is up to the requesting user’s booking service to store the booking (and item or inventory) data if it wants to enable its users to view their bookings and items later.


```
interface Inventory {
  id: string;
  meta?: object; // JSON-LD
  form?: object; // JSON Schema
  itemType?: string; // JSON-LD context for the item meta
  itemMetaSchema?: object; // JSON Schema
  items?: Array<{
    id: string;
    xor?: string[]; // IDs of mutually exclusive items
    capacity?: number;
    bookDeadline?: string; // ISO 8601
    meta?: object; // JSON-LD
    occupancy?: number; // Number of bookings of the item
  }>;
}
```

Listing 3.2: Definition of Inventory resource JSON using a TypeScript interface

```
interface Booking {
  formData?: object;
  item: {
    id: string;
  };
}
```

Listing 3.3: Definition of Booking resource JSON using a TypeScript interface

```
interface BookingWithItem {
  id: string;
  createdAt?: string;
  formData?: object;
  item: {
    id: string;
    capacity?: number;
    bookDeadline?: string;
    meta?: object;
  };
}
```

Listing 3.4: Definition of BookingWithItem resource JSON using a TypeScript interface

3.2.1 Use case

The booking protocol will be used for a simple use case of a business that occasionally holds one-time events and it wants to know names, emails, and optionally phones of the users who are interested in attending. The business will use the booking protocol to allow users to book a spot at the event. The business will have a booking service, which will be used by the users to book a spot at the event.

The business also wants to let the users know its name, description, website, and contact information including email, phone, physical location, and opening hours. Each event (item of the inventory) will have a deadline for booking, a maximum number of bookings, duration, physical location, website, and a form schema, which the users must fill in when booking a spot at the event. The form schema will be a JSON schema, which will be used to validate the form data sent by the users when booking a spot at the event.

3.3 Booking Service

The booking service shall implement the booking protocol designed in 3.2. Furthermore, it shall offer internal APIs for use by the booking client application.

3.3.1 Database

This section describes the design decisions behind the structure of the database and the choice of the database engine.

3.3.1.1 Conceptual Schema

The data that will need to be stored in the database can be described by the conceptual schema in figure 3.5. The database will need to store information about users, inventories, items, bookings, and the relationships between them. Most of the data is highly relational, with some properties being JSON documents. Either one database with support for both relational and document data types, or two separate databases, one for relational data and one for document data, could be used.

3.3.1.2 OLAP vs. OLTP

When determining which database engine to use and how to structure the database, one aspect to examine is whether the system will be used for online analytical processing (OLAP) or online transaction processing (OLTP). This system is a prime example of OLAP

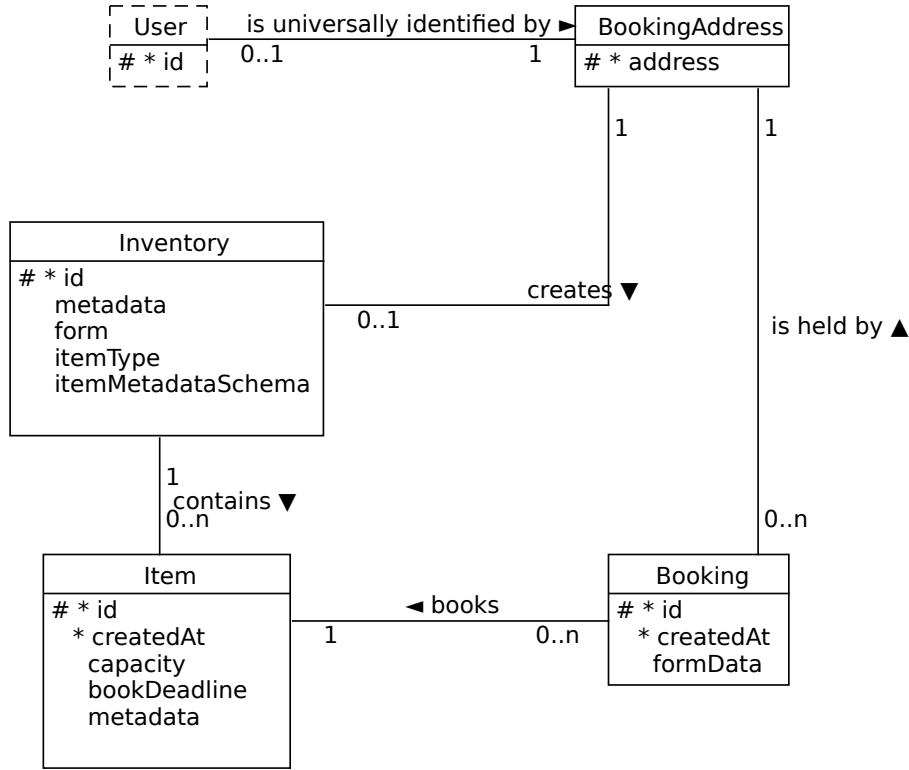


Figure 3.5: Conceptual schema of the booking service database

3.3.1.3 ACID vs. BASE

According to [24, 25], consistency, availability, and partition tolerance are the three properties that are impossible to achieve simultaneously in a distributed system. This is known as the CAP theorem. In the context of databases, the CAP theorem is often discussed in terms of ACID (Atomicity, Consistency, Isolation, Durability) and BASE (Basically Available, Soft state, Eventually consistent) properties.

In the domain of booking, strong consistency and availability is absolutely essential. For example, an item available for booking must never be booked by more users than its capacity. Bookings are can also be very time sensitive. For those reasons, the system must be designed with ACID properties in mind.

3.4 Client Application

The booking client application shall use a single page application (SPA) architecture. The SPA architecture is a design pattern that allows the client application to load a single HTML page and dynamically update the page as the user interacts with the application. This design pattern allows the client application to be more responsive and provide a better user experience.

The client application will be a progressive web application (PWA) that can be installed on the user's device and possibly even work offline (though that will not be implemented as of now). This will also enable the use of the custom booking URL schema (on supported platforms). And as any web application, it will be accessible on many different devices and platforms.

Lastly, it will be a mobile-first design to ensure that the application is usable on mobile devices.

Initially, the Figma design tool was used to create wireframes of the UI, however, the UI has changed by a not significant amount since then, so the original design is left out of the thesis.

Implementation

This chapter describes the prototype implementation of the designed system, as well as its testing and evaluation.

For the entire implementation, TypeScript, Node.js, and npm were used. The reason behind this choice is both prior experience with this ecosystem of the author of this thesis, as well as its popularity for full-stack web development, ensuring high amount of useful supporting resources.

TypeScript [26] is a popular “strongly typed programming language that builds on JavaScript.” It “adds additional syntax to JavaScript to support a tighter integration with (...) an editor” to “catch errors early.” The code “converts to JavaScript, which runs anywhere JavaScript runs: in a browser, on Node.js, Deno, Bun.” TypeScript is a static type checker, meaning that it performs type checking only at compile time, not during the runtime.

Node.js [27] is a “cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts.” It is “built on the V8 JavaScript engine” (which is also used [28] by Chromium-based web browsers, such as Google Chrome). The Node.js documentation [27] further states that “npm is the standard package manager for Node.js,” with npm [29] claiming that it is “the largest software registry in the world.”

Each application that is part of the implementation is kept in a separate directory: *booking-service* for the booking service, *booking-api-gateway* for the API gateway, and *booking-client-app* for the client application. These directories with the source code are included in the content of the digital attachment B.

Every application directory contains a *package.json* file and an *npm-shrinkwrap.json* file. These files in the JavaScript Object Notation (JSON) data format are both related to npm, with *package.json* [30] describing many things, including general metadata about the project (package name, version adhering to semantic versioning, license information, author information, important URLs, etc.), npm scripts’ definitions, lists of npm dependencies

(of various types) along with their version, application entrypoints, and information about supported environments. The file *npm-shrinkwrap.json* is then generated by npm when installing dependencies (using the `npm install` command) and it is a publishable variant of the *package-lock.json* file used by default to describe “the exact (dependency) tree that was generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.” This ensures consistently reproducible dependency installations. Dependencies are installed into the *node_modules* directory of each project (this directory is not included in the source code).

To fulfill non-functional requirement **N8**, each application directory also contains a *LICENSE.txt* file with the popular permissive open-source MIT license (which is also correctly denoted in the *package.json* files) and each application only uses dependencies with compatible permissive open-source licenses.

Furthermore, the file *Containerfile* is included in each application directory. *Containerfile* [31] (which is an alternative name for *Dockerfile*) is “a configuration file that automates the steps of creating a container image.” Application container images can then be created using tools like Docker, Podman, and Buildah, either in the Docker image format or in the Open Container Initiative (OCI) image format (which is largely similar). These container images can then be used by orchestration tools (such as Kubernetes) for [16] “automating deployment, scaling, and management of (...) applications”. Additionally, the *compose.prod.yaml* Compose file is attached to enable easily building and running all the service containers in production mode using tools like Docker Compose, and the *booking-service* directory contains a *compose.yaml* file with the dependant services for the booking service to run (and thus is useful for development). This fulfills non-functional requirement **N2**.

Lastly, each application directory contains a *README.md* file in the Markdown format, describing how to run it, and a *.env.template* file with a template for the environment variables that need to be set for the application to run.

4.1 Booking Service

The booking service was built around the framework NestJS (Nest). Nest [32] “is built with and fully supports TypeScript” and is used “for building (...) Node.js server-side applications.” “Under the hood, Nest makes use of robust HTTP server frameworks like Express (the default) and optionally can be configured to use Fastify as well.” As Express is the default and (judging by download numbers from npm) the much more popular choice, and the Nest documentation [32] occasionally mentions unsupported functionality when using Fastify, Nest was configured to use Express than the alternative. Nest “provides an out-of-the-box application architecture which allows devel-

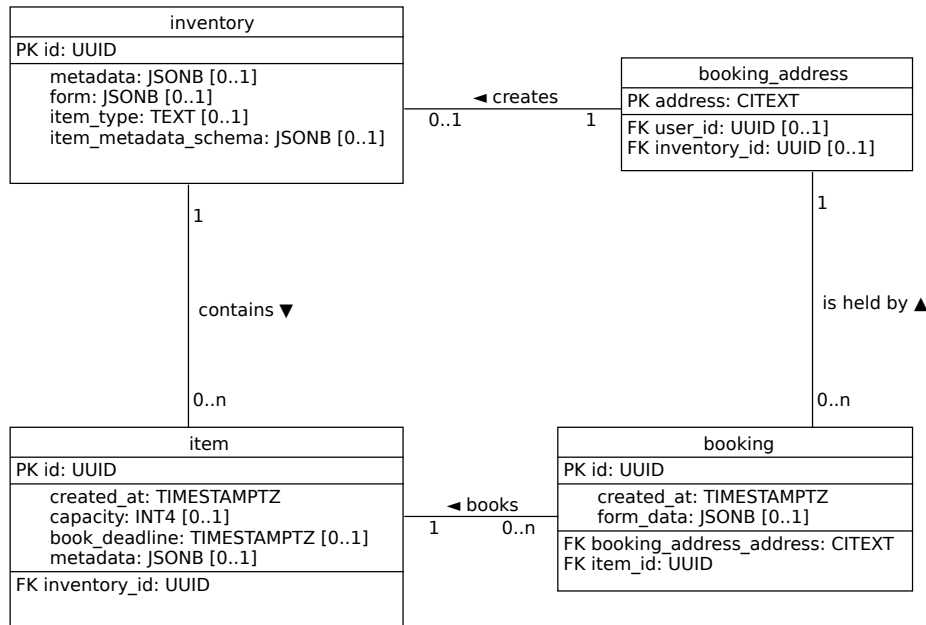


Figure 4.1: Diagram of the booking service database

opers and teams to create highly testable, scalable, loosely coupled, and easily maintainable applications.”

The framework makes frequent use of the dependency injection pattern. This pattern is used to inject services into controllers, which are used to handle incoming requests. The controllers are responsible for handling incoming requests and returning responses. The services are responsible for handling the business logic of the application. The services are also responsible for handling the communication with the database.

For Object Relational Management (ORM), MikroORM was used. For the database, PostgreSQL was used, which is an ACID compliant relational database management server, with support for JSON data types. The diagram of the PostgreSQL database can be seen in the figure 4.1.

The booking service implements the booking protocol described in 3.2. Furthermore it offers internal APIs for use by the client.

4.2 API Gateway

The API gateway behaves to the outside worlds as a booking service. It lets anybody access the booking protocol APIs of the booking service, but only authenticated users to use to internal APIs meant for the booking client app.

Much like the booking service, the API gateway was built around the framework NestJS (Nest), with Express enabled.

For user authentication, the library SuperTokens was used. For proxying requests to booking service, the library http-proxy-middleware was used.

The API gateway also provides an internal endpoint `POST /forward-booking-request` for use by the booking service only, which is used to forward booking requests from the booking service to remote booking services.

4.3 Client Application

The client application was built around the framework Next.js (not to be confused with NestJS, a different framework, used in the implementation of previously described services). Next.js [33] “is a React framework for building full-stack web applications.” “Under the hood, Next.js also abstracts and automatically configures tooling needed for React, like bundling, compiling, and more.” Some of its main features include:

- **Routing** — “A file-system based router built on top of Server Components that supports layouts, nested routing, loading states, error handling, and more.”
- **Rendering** — “Client-side and Server-side Rendering with Client and Server Components. Further optimized with Static and Dynamic Rendering on the server with Next.js. Streaming on Edge and Node.js runtimes.”
- **Data Fetching** — “Simplified data fetching with `async/await` in Server Components, and an extended `fetch` API for request memoization, data caching and revalidation.”
- **Styling** — “Support for your preferred styling methods, including CSS Modules, Tailwind CSS, and CSS-in-JS.”
- **Optimizations** — “Image, Fonts, and Script Optimizations to improve your application’s Core Web Vitals and User Experience.”
- **TypeScript** — “Improved support for TypeScript, with better type checking and more efficient compilation, as well as custom TypeScript Plugin and type checker.”

The MaterialUI component library was used during the creation of the pages.

4.3.1 Pages

4.3.1.1 Authentication

When a user accesses any part of the client application without being signed into an account, they are redirected to the authentication page, available under the path `/auth`. This redirection and validation of user sessions is handled by the `SuperTokensWrapper` and `SessionAuth` components from the SuperTokens SDK, which surround the contents of a page that requires authentication (in this case all pages).

The authentication page can be seen in the figure 4.2 and contains another component from the SuperTokens SDK, that is configured based on the desired authentication methods. The methods chosen are email with password, and third-party social login using a GitHub account. The component features a button for the social sign-in alongside two input fields and a confirmation button for the email with password sign-in method. The default state for the component is sign in (log in with an existing account). This state can be changed to sign up (create a new account) with a button present at the top of the component. Furthermore, there is also a “forgot password” button which, upon activation, switches the component to another state where the user inputs their account email address and confirms that they wish to reset their password. The user is then sent an email with a password reset link which, when opened, allows the user to choose their new password.

4.3.1.2 Home

The home page, available under the path `/`, is divided into three main states: booking address creation, an overview of user’s bookings, and an item booking detail.

If the user has not yet created a booking address, they are prompted to do so. The page then contains a one-field form, where the user fills in their desired booking address username. This username, along with the booking service provider’s provided hostname, then forms the user’s new booking address. During the process of choosing their username, the user gets immediate feedback if the username is invalid. The user then confirms their username selection, upon which either their new booking address is created, moving them to the home page state with an overview of their bookings or, if their booking address was already used by another user, an alert is displayed with an informative error (the alert can be hidden by clicking on it). The username input and validation can be seen in the figure 4.3, while the confirmation and error can be seen in the figure 4.4. This, along with the authentication page, fulfills functional requirement **F1**.


The overview of user’s bookings contains a vertical list of the items, that the user has booked. This overview can be seen in the figure 4.5 Each item lists the inventory owner’s booking address at the top, and item metadata at the

4. IMPLEMENTATION

The image displays two distinct authentication forms side-by-side. The left form, titled 'Sign In', includes a link for new users ('Sign Up'), a GitHub login option, an email field, a password field, a 'Forgot password?' link, and a 'SIGN IN' button. The right form, titled 'Change your password', prompts the user to enter a new password and confirm it, with a 'CHANGE PASSWORD' button. Both forms are powered by SuperTokens, as indicated by the footer text.

Sign In

Not registered yet? [Sign Up](#)

 Continue with GitHub

or

Email

Password [Forgot password?](#)

SIGN IN

Powered by SuperTokens

Change your password

Enter a new password below to change your password

New password

Confirm password

CHANGE PASSWORD

Figure 4.2: Authentication page with a SuperTokens component in the sign-in state seen left, password reset seen right.

Figure 4.3 displays two side-by-side screenshots of a web form titled "Your new booking address".

The left screenshot shows the form with the username input field containing the text "bob". The input field is labeled "username@localhost" and "@localhost". A blue button with a right arrow icon and the text "CREATE" is visible at the bottom right.

The right screenshot shows the form with the username input field containing a tilde "~". The input field is labeled "username@localhost" and "@localhost". A red validation error message is displayed below the input field: "Username can contain only Arabic numerals (0-9), 26 English letters regardless of case (A-Z, a-z), and the following special characters: dot ".", underscore "_", and hyphen '-'". A greyed-out button with a right arrow icon and the text "CREATE" is visible at the bottom right.

Figure 4.3: Home page in the create booking address state with booking address username input seen left and its validation to the right

4. IMPLEMENTATION

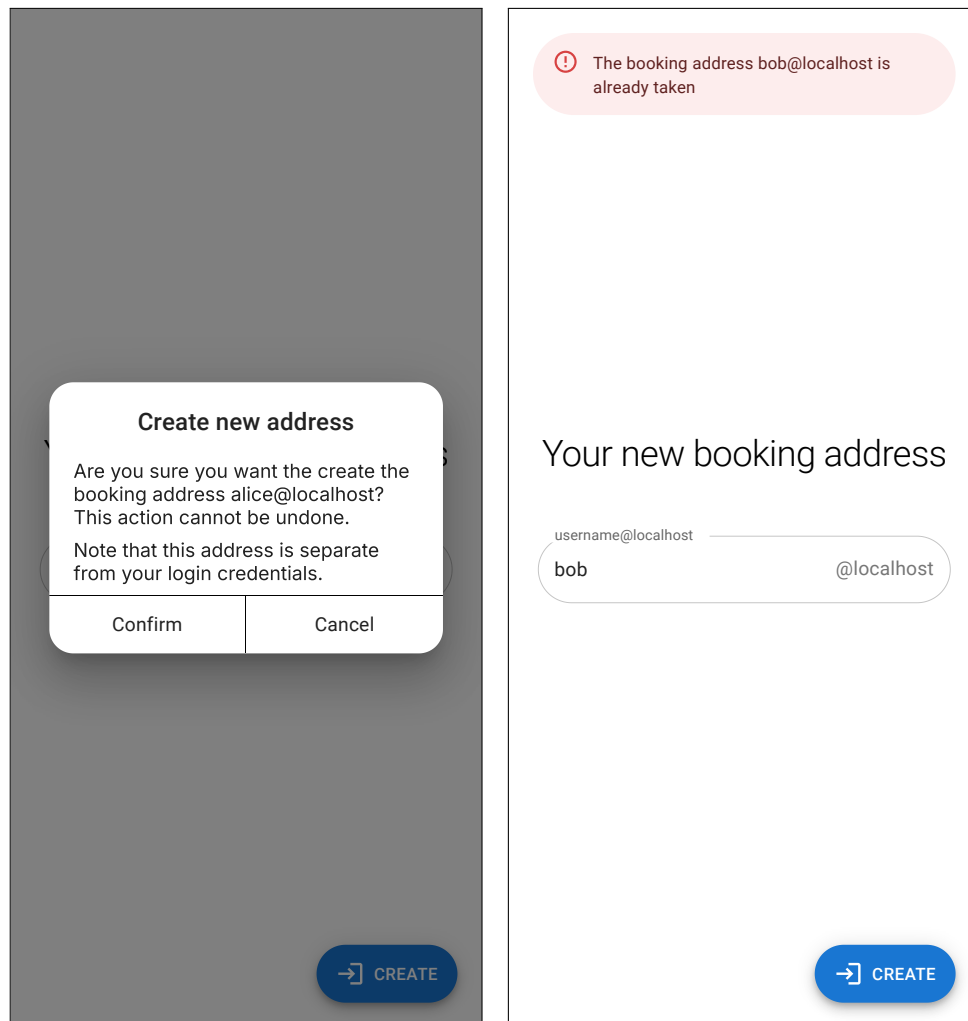


Figure 4.4: Home page in the create booking address state with confirmation before address creation seen left and an error due to the address being taken seen to the right

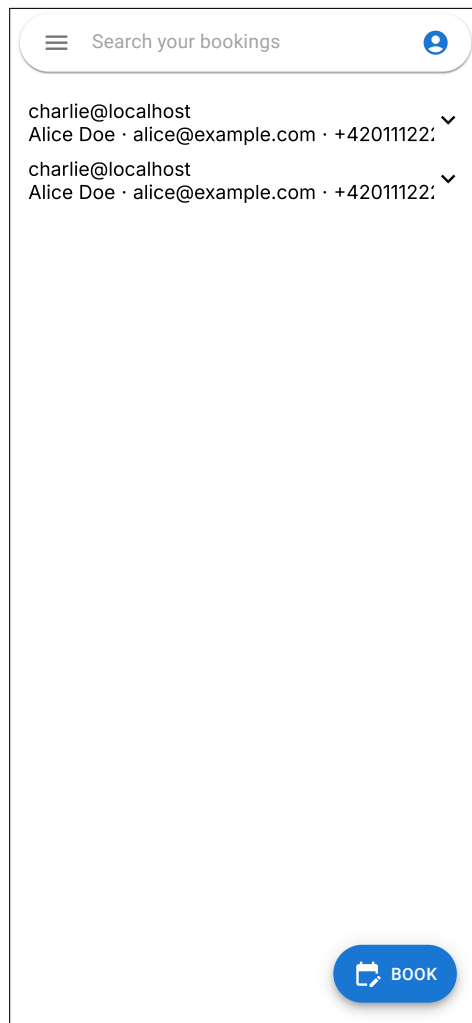


Figure 4.5: Home page in the bookings overview state

bottom, as many as can fit onto one line. Upon clicking an item from the list, the user is moved to the third state of the home page, the item booking detail. A menubar is present at the top of the bookings list, with a “burger” button that opens a drawer with links to the main client application sections (the currently active home page, and the business page), a search bar for searching within the bookings (currently not implemented), and a button with a profile icon that opens a dialog with the user’s booking address and a log out button. The open drawer and profile dialog are displayed in the figure 4.6. To the bottom-right corner of the page, a “book” button is present, that redirects the user to the new booking page.

The item booking detail, which is shown in the figure 4.7, contains a list of information about the booked item, including its inventory’s owner’s booking

4. IMPLEMENTATION

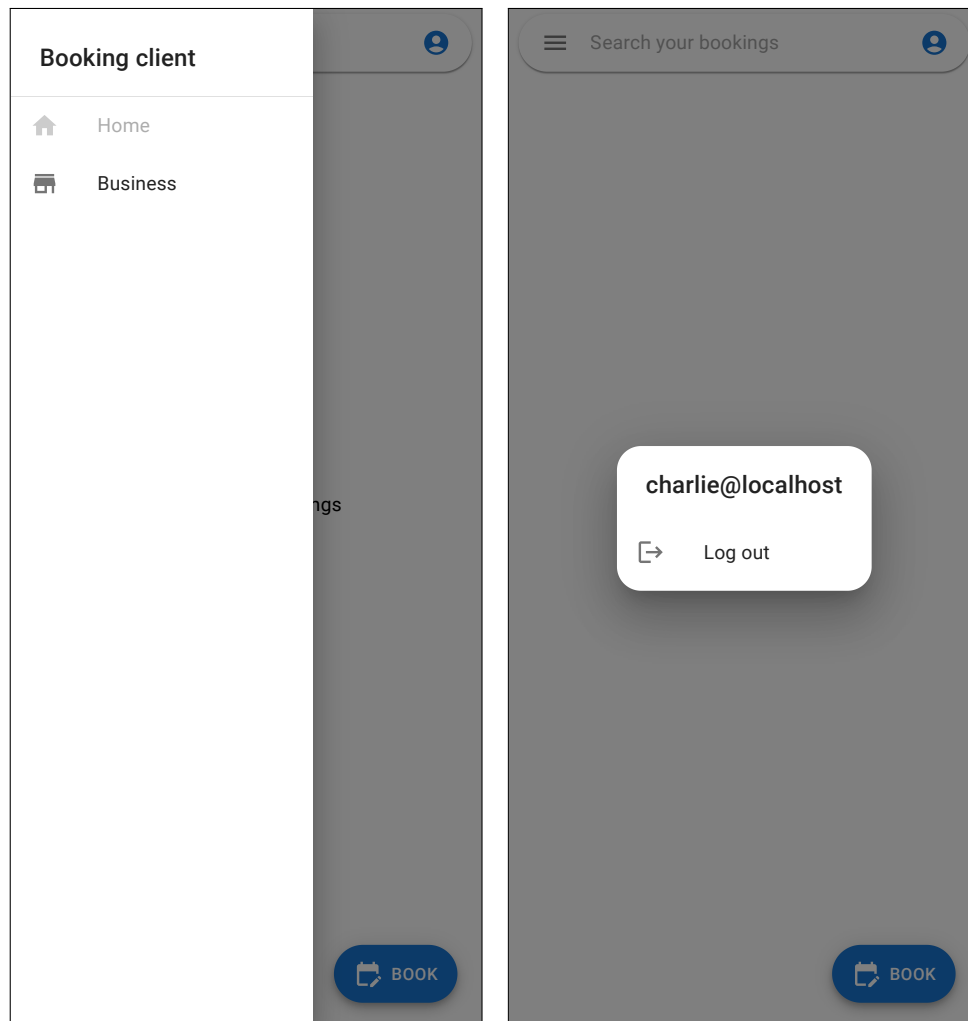


Figure 4.6: Open drawer with main website section links seen left, dialog with profile information and log out button seen right

[←](#) Booking details

Booking address: charlie@localhost

Name: Tuesday class

Duration: PT90M

Location: Thákurova 9, 160 00 Praha 6

StartDate: 4. 3. 2025 19:00:00

Form data:

Name: Alice Doe

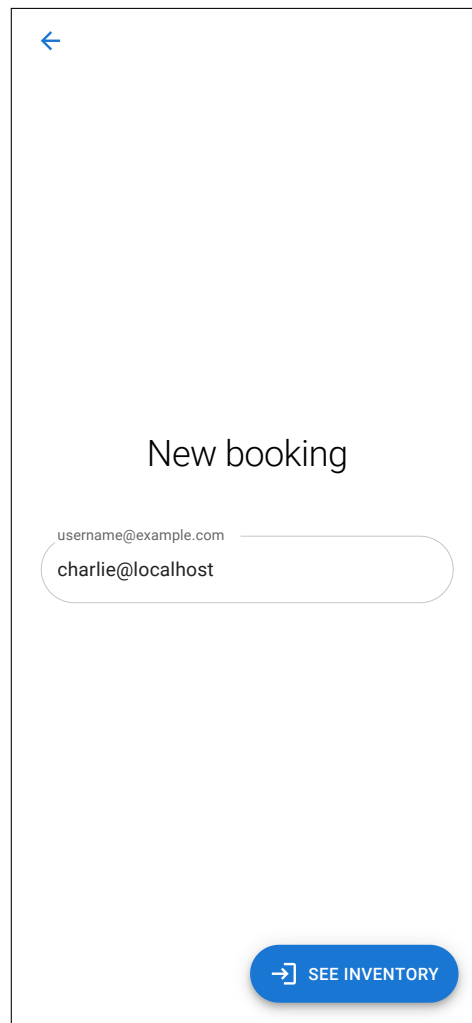
Email: alice@example.com

Telephone: +420111222333

Figure 4.7: Home page in the item booking detail state

address and the item’s metadata. This list is followed by listed form data, that the user filled in during booking. At the top, there is a title of the page state along with a back button that allows the user to move back to the bookings overview state.

The reason for keeping the item booking detail a state of the home page, instead of its own separate page, is that the home page already has all the data needed, therefore preventing unnecessary data fetching from the booking service.



The image shows a mobile app screen for a 'New booking' page. At the top left is a blue back arrow. The title 'New booking' is centered. Below it is a text input field with a placeholder 'username@example.com' and the text 'charlie@localhost'. At the bottom right is a blue button with a right arrow icon and the text 'SEE INVENTORY'.

Figure 4.8: New booking page

4.3.1.3 New booking

The new booking page, available under the path `/book`, contains a form similar to the one for creating a new booking address. Here, the user inputs an entire booking address (rather than only a username). After filling in the booking address (which is again validated as the user types) and clicking on the “see inventory” button at the bottom-right of the page, the user is redirected to an items for booking page. At the top, there is also a title of the page along with a back button that redirects the user to the home page. The new booking page can be seen in the figure 4.8.

4.3.1.4 Items for booking

The items for booking page, presented in the figure 4.9, available under the path `/book/[address]`, where `[address]` is the URI-encoded booking address, whose inventory is to be fetched and displayed. Additionally, `[address]` can be an entire URI-encoded booking URL, which is used when opening an installed booking client PWA application using a booking URL such as `web+booking://username~example.com`, under supported platforms.

After the page is opened, the inventory of the requested booking address is fetched and in case of an error, an alert is displayed to the user. Then, the page displays the inventory details and metadata, and the items available for booking, along with a snippet of their metadata and occupancy. When the user clicks an item, the page switches to a second state, showing item details. As with the previous page, there is a header with the page title and a button to go back to the home page.

In the item details state, the page first shows the item data and metadata, followed by a booking form for the user to fill out. The booking form is generated dynamically, based on the JSON Schema present in the item data. This is done using the JSON Forms library. There is also a header with a title and a button to go back to the basic items for booking state. After the user fills out the form and clicks the “create booking” button on the bottom-right, a request for booking the selected items with the filled-out form attached is sent (first through the API gateway to the user’s own booking service, then from the user’s booking service to the item inventory’s owner’s booking service). If there has been an error in the process, an alert is shown, otherwise, the user is redirected to the home page. Item details with a filled-out form, along with an alert after unsuccessful booking, can be seen in the figure 4.11. This, along with the home page, fulfills functional requirement **F3**.

4.3.1.5 Business

The business page, shown in the figure 4.11 and available under the path `/business`, has two main states, create an inventory, and view inventory items.

The create an inventory state is used when a user has no inventory. The user is then presented a form to fill out inventory details and metadata. This form is dynamically generated using the JSON Forms library from a JSON Schema, but the JSON Schema is currently static for the entire client application based on the designed use case. In the future, there could be multiple pre-defined schemas for different use cases, as well as, for instance, an option to upload a custom schema. When the user fills out the form and submits using the “create” button at the bottom-right of the page and confirms a dialog that pops out, their inventory is created and they become a bookable user.

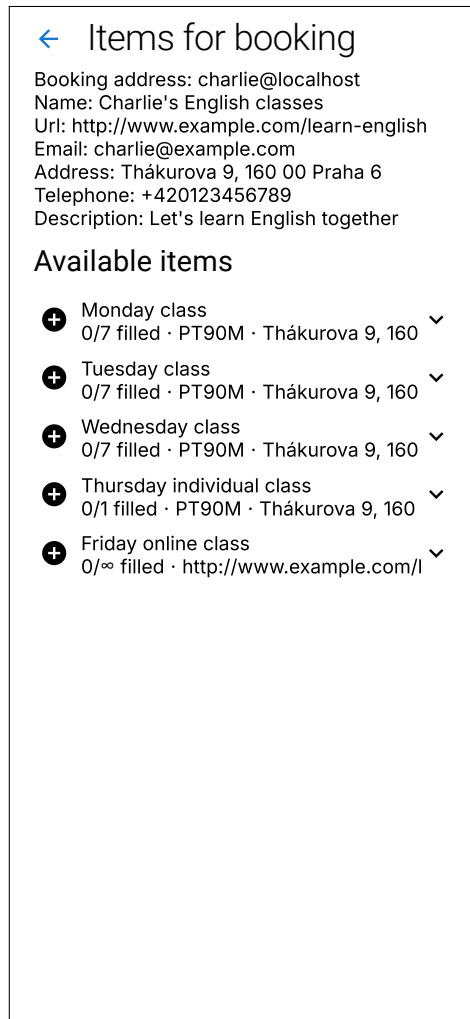


Figure 4.9: Items for booking page with the requested booking address inventory and its items

←

Item details

Book deadline: 4. 3. 2025 19:00:00

Capacity: 7

Occupancy: 0

Name: Tuesday class

Duration: PT90M

Location: Thákurova 9, 160 00 Praha 6

StartDate: 4. 3. 2025 19:00:00

Name *

Alice Doe

Email *

alice@example.com

×

Phone

+420111222333

+ CREATE BOOKING

←

Item details

Error 400: The item is full (its capacity has been reached)

C.

Occupancy: 1

Name: Thursday individual class

Duration: PT90M

Location: Thákurova 9, 160 00 Praha 6

StartDate: 6. 3. 2025 19:00:00

Name *

Bob Deer

Email *

bob@example.com

×

Phone

+420333222111

+ CREATE BOOKING

Figure 4.10: Item details state of the items for booking page, showing item information, metadata, and filled-out form to the left, and an alert upon unsuccessful booking to the right

4. IMPLEMENTATION

The figure displays two side-by-side mockups of a business page, representing different states of the application.

Left Mockup (Create New Inventory State):

- Header:** A search bar labeled "Search your inventory" and a user profile icon.
- Title:** "Your new inventory"
- Text:** "You do not currently have an inventory set up. If you wish to create one, you may do so here."
- Form Fields:**
 - Name ***: Charlie's English classes
 - Description**: Let's learn English together
 - Website**: <http://www.example.com/learn-english>
 - Contact**
 - Email**: charlie@example.com
 - Phone**: +420123456789
 - Location**: Thákurova 9, 160 00 Praha 6
 - Opening Hours**
- Action:** A blue button labeled "CREATE" with a right arrow icon.

Right Mockup (Inventory List State):

- Header:** A search bar labeled "Search your inventory" and a user profile icon.
- Inventory List:**
 - Monday class**: 0/7 filled · PT90M · Thákurova 9, 160 00 Pra^l ✓
 - Tuesday class**: 2/7 filled · PT90M · Thákurova 9, 160 00 Pra^l ✓
 - Wednesday class**: 0/7 filled · PT90M · Thákurova 9, 160 00 Pra^l ✓
 - Thursday individual class**: 1/1 filled · PT90M · Thákurova 9, 160 00 Pra^h ✓
 - Friday online class**: 0/∞ filled · <http://www.example.com/learn-ei> ✓
- Action:** A blue button labeled "ADD ITEM" with a plus icon.

Figure 4.11: Business page in the create new inventory state seen left, business page with a list of inventory items to the right

The view inventory items state is shown to bookable users. It contains a list of the items in the user's inventory, including snippets of the item's information (such as occupancy) and metadata. Upon clicking an item from the list, the user is redirected to the inventory item detail page. There is also a "add item" button at the bottom-right of the page, which redirects the user to the add item page.

Both states of this page share a menubar at the top of the page, which works analogically to the one on the home page.

4.3.1.6 Add item page

The add item page, available under the path `/business/add-item`, features a dynamically generated form, that the bookable user uses to fill out item information and metadata. As with the inventory creation form, the JSON Schema used is currently static for the client application, but adding new ones in the future should be simple. After filling out the form, the user clicks the “add” button, confirm a dialog and creates a new item. In case of an error during the item creation, an alert is shown. After successful item creation, the user is redirected back to the business page. The page is displayed in the figure 4.12. This, together with the business page, fulfills functional requirements **F2** and **F4**. Moreover, together with the booking pages and mechanisms, it fulfills functional requirement **F5**.

4.3.1.7 Inventory item detail page

The inventory item detail page, available under the path `/business/items/[itemId]`, where `[itemId]` is the id of the item that should be shown, has two main states. Its first state – item details – shows information about the selected item along with a list of its bookings and snippets of the bookings’ information. After clicking on a booking, the page switches to its second state, where the user is shown details of the selected booking, including its form data. Both states feature a page heading at the top, along with a button to go back to the business page, respectively to the item detail state in the case. The page can be seen in the figure 4.13.

4.4 Testing and Evaluation

First, every part of the implementation was tested manually by the author of this thesis. For the client application, this meant clicking through every page and state, testing edge-case inputs (invalid usernames, booking addresses, form data) and even setting up booking service failure scenarios by temporarily modifying the booking service to respond with invalid data. For the API gateway and the booking service, the command line utility `cURL` was used to send both valid and invalid requests, including different HTTP header combinations.

Afterwards, when the implementation appeared stable, user usability testing was performed.

4.4.1 Usability testing

Usability testing was performed with three subjects. **Subject A** is a 26 year old male software engineer with a university degree, who lives in the suburbs of Prague. **Subject B** is a 31 year old male army soldier, who lives in Prague.

4. IMPLEMENTATION

← New inventory item

Name *

Monday class

Description

Start Date *

2025-04-07 19:00

Duration

PT90M

Website

Location

Thákurova 9, 160 00 Praha 6

Capacity

7

Book Deadline

→ ADD

Figure 4.12: Add item page

Subject C is a 52 year old female business manager, who lives in the suburbs of Prague and is not too technologically savvy.

The following scenario was chosen based on the designed use case and given to testing subjects: “You are a manager at a mid-sized business (a couple hundred employees). Your company wants to organize a conference and needs to keep track of how many and who can partake. You know that you yourself will be at the conference.”

Then, the following tasks were given to each subject:

1. “Create a new account and booking address.”
2. “Create a new inventory with basic information about yourself as a manager.”

| ← Item details | ← Booking details |
|---|---|
| Book deadline: 4. 3. 2025 19:00:00 Capacity: 7 Occupancy: 2 Name: Tuesday class Duration: PT90M Location: Thákurova 9, 160 00 Praha 6 StartDate: 4. 3. 2025 19:00:00 Bookings: alice@localhost Alice Doe · alice@example.com · +420111; ✓ bob@localhost Bob Deer · bob@example.com · +420333 ✓ | Booking address: alice@localhost Name: Alice Doe Email: alice@example.com Telephone: +420111222333 |

Figure 4.13: Inventory item detail page, with its item detail state to the left, and booking detail to the right

4. IMPLEMENTATION

3. “Create a new inventory item for the conference.”
4. “Make a new booking of the conference item, since you know that you will be participating.”
5. “Check if you can find your booking in the inventory.”

The testing was done in person, on a Windows 11 laptop that was brought to the testing setting, with all the required services set up and the booking client application open in the latest Google Chrome browser.

The testing subjects performed the tasks as follows:

1. Account and address creation:
 - **Subject A** signed in using their GitHub account, input a valid username and the first try and created their booking address.
 - **Subject B** took a moment to find out how to switch the sign-in component to the sign up state, then created an account with email and password. They input a username and created their booking address, though at first try they put in an invalid character for the username, which was quickly corrected thanks to the instant UI feedback.
 - **Subject C** tried to fill in the email and password for their new account into the sign in form. When that attempt was unsuccessful, they realized the presence of the button to switch the component to the sign up state and created their new account successfully. Then they put in a simple username and created a booking address without an issue.
2. Inventory creation:
 - **Subject A** immediately spotted the burger icon button that opens the drawer with a link to the business section, where they transitioned, filled out the new inventory form and created their inventory.
 - **Subject B** first opened the profile dialog, but their second choice was the burger icon button. The subject created their inventory without issue.
 - **Subject C** took a little while to look around the page, but their first choice was the burger icon button. The rest of the process was flawless.
3. Item creation:

- **Subject A** navigated to the add item page and filled out the form. They were experimenting with the duration field, which they said has a too exotic format, though it was not needed for this task. Then they proceeded to create the item.
- **Subject B** navigated to the add item page, filled out the form and added the item to the inventory.
- **Subject C** also navigated to the add item page, filled out the form and added the item to the inventory without anything to mention.

4. Item booking:

- **Subject A** first clicked the newly created item, then realized that they need to navigate back to the home page and then to the new booking page. They at first did not realize, that they need to input the whole booking address, but thanks to the instant feedback of the UI, they managed to get to the items for booking page and complete the booking.
- **Subject B** experienced the same issues as **Subject A**, but managed to complete the booking.
- **Subject C** also struggled to get to the correct page and then to find out what to input as the booking address. Once they got over this step, the process was straightforward.

5. Booking overview:

- **Subject A** finished the task successfully without anything to mention.
- **Subject B** accessed the booking through their home page and thought that they completed the task. They had to be told, that this is not the correct place, otherwise they would not complete the task successfully.
- **Subject C** first opened the booking through their home page, then re-read the instructions and went correctly to their inventory to access the booking.

4.4.2 Evaluation

Through the conducted usability testing, some UI/UX blind spots were found. The issues some users faced with the sign in form could be attributed to the fact, that the form was in English and neither were native English speakers, so the two similar terms “sign in” and “sign up” can be confusing. Improvements in the client application could be made to better parse user values of certain metadata (enabling for example more human readable duration or opening hours format). Some sections of the client app are very similar, which can

4. IMPLEMENTATION

be also confusing. Likely the biggest hurdle is the exotic nature of booking addresses. This issue could be improved by displaying the users' booking addresses more prominently in the application.

These issues, however, are relatively minor and the implementation is fully functional and fulfills the designed use case.

Conclusion

The goal of this thesis was to design an open and decentralized, general-purpose online reservation system architecture, including a common HTTP-based client-server protocol, and developing a reference implementation including both a client application and a back end. The created solution was to be easy to integrate with state-of-the-art cloud-native architectures, and was to support a use case for a selected type of reservation business. Possible extensions for the selected use case were to be discussed. The reference implementation was to be tested and evaluated for the selected use case.

First, a research/overview of select existing online reservation systems was conducted in chapter 1. Based on the results of the research, the assignment of the thesis, as well as original ideas, a requirements analysis was made in chapter 2, including both functional and non-functional requirements for the system to be designed and implemented. Chapter 3 describes the design of the newly created system, including its architecture, the common protocol, a use case for the system, the back-end services, and the client application. Finally, chapter 4 describes the implementation of the system based on the created design, as well as its testing and evaluation, fulfilling the selected use case.

The research on the existing online reservation systems could additionally be useful to those who are in the near future looking for a reservation system. Furthermore, together with the requirements analysis, this information could be useful to those who are looking into developing a reservation system of their own. The approaches in the design and implementation chapters can, in part, be used as a reference for those who are looking into developing other applications with some of the technologies or techniques used here. The designed architecture and protocol can be used by others to create new interoperable implementations. And, lastly, the implemented prototype can be further developed into a production-ready system, or integrated into existing infrastructures, as it is released under a permissive open-source license.

CONCLUSION

One can dream of a future where, if the ideas proposed in this thesis ever became a commonplace reality, after searching for a hairdresser nearby using a search engine, users would be presented with a list of results including not only the basic information about hair salons within close proximity, but also with real-time information about the hairdressers' availability (based on which the results could also be sorted), and a direct link to book an appointment, which would open a user's booking client application of choice, that would in turn automatically fill out the user's personal data if required by the business offering the appointment, and let the user just confirm the booking. Or a future where users could have their booking service automatically book an appointment with their dentist for regular check-ups twice a year, with the exact time and date chosen based on the availability of both parties and an optional notification about the newly created booking.

Bibliography

1. *Reservio* [online]. Reservio, s.r.o., 2024 [visited on 2024-01-05]. Available from: <https://www.reservio.com>.
2. BEATY, Mike; KOCOT, Larry; SHEIDY, Dion; COAKLEY, Monica; ROBBINS, Lori; WALSH, Liam; GINIAT, Ed. *Healthcare 2030*. KPMG LLP, 2019. Available also from: <https://kpmg.com/kpmg-us/content/dam/kpmg/pdf/2023/kpmg-healthcare-2030.pdf>.
3. PARÉ, Guy; TRUDEL, Marie-Claude; FORGET, Pascal. Adoption, use, and impact of e-booking in private medical practices: Mixed-methods evaluation of a two-year showcase project in Canada. *JMIR Medical Informatics*. 2014, vol. 2, no. 2. Available from DOI: 10.2196/medinform.3669.
4. BANTON, Caroline. *Network Effect: What It Is, How It Works, Pros and Cons* [online]. Investopedia, 2023 [visited on 2024-02-14]. Available from: <https://www.investopedia.com/terms/n/network-effect.asp>.
5. *Acuity Scheduling* [online]. Acuity Scheduling, 2024 [visited on 2024-01-07]. Available from: <https://www.acuityscheduling.com>.
6. *Square* [online]. Block, Inc., 2024 [visited on 2024-01-27]. Available from: <https://squareup.com>.
7. *Wix* [online]. Wix.com, Inc, 2024 [visited on 2024-02-02]. Available from: <https://www.wix.com>.
8. ISO CENTRAL SECRETARY. *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model – Part 1*. Geneva, CH, 1994. Standard, ISO/IEC 7498-1:1994. International Organization for Standardization. Available also from: <https://www.iso.org/standard/20269.html>.
9. PRESTON-WERNER, Tom. *Semantic Versioning 2.0.0* [online]. 2023. [visited on 2024-02-06]. Available from: <https://semver.org>.

10. DÜRST, Martin J.; SUIGNARD, Michel. *Internationalized Resource Identifiers (IRIs)* [RFC 3987]. RFC Editor, 2005. Request for Comments, no. 3987. Available from DOI: 10.17487/RFC3987.
11. SPORNY, Manu; LONGLEY, Dave; KELLOGG, Gregg; LANTHALER, Markus; LINDSTRÖM, Niklas. *JSON-LD 1.0*. 2014-01. W3C Recommendation. W3C. <https://www.w3.org/TR/2014/REC-json-ld-20140116/>.
12. SPORNY, Manu; LONGLEY, Dave; KELLOGG, Gregg; LANTHALER, Markus; CHAMPIN, Pierre-Antoine; LINDSTRÖM, Niklas. *JSON-LD 1.1*. 2020-07. W3C Recommendation. W3C. <https://www.w3.org/TR/json-ld11/>.
13. WRIGHT, Austin; ANDREWS, Henry; HUTTON, Ben; DENNIS, Greg. *JSON Schema: A Media Type for Describing JSON Documents* [online]. Internet Engineering Task Force, 2022 [visited on 2024-02-08]. Available from: <https://json-schema.org/draft/2020-12/json-schema-core>.
14. WRIGHT, Austin; ANDREWS, Henry; HUTTON, Ben; DENNIS, Greg. *JSON Schema* [online]. 2024. [visited on 2024-02-08]. Available from: <https://json-schema.org>.
15. *OpenAPI Guide* [online]. SmartBear Software, 2024 [visited on 2024-02-08]. Available from: <https://swagger.io/docs/specification/about/>.
16. *Kubernetes* [online]. The Kubernetes Authors & The Linux Foundation, 2023 [visited on 2024-02-08]. Available from: <https://kubernetes.io>.
17. LEACH, Paul J.; SALZ, Rich; MEALLING, Michael H. *A Universally Unique Identifier (UUID) URN Namespace* [RFC 4122]. RFC Editor, 2005. Request for Comments, no. 4122. Available from DOI: 10.17487/RFC4122.
18. ISO CENTRAL SECRETARY. *Date and time — Representations for information interchange — Part 1: Basic rules*. Geneva, CH, 2019. Standard, ISO/IEC 8601-1:2019. International Organization for Standardization. Available also from: <https://www.iso.org/standard/70907.html>.
19. *List of restricted ports in browsers – Knowledge Base* [online]. Neo4j, Inc., 2020 [visited on 2024-02-14]. Available from: <https://neo4j.com/developer/kb/list-of-restricted-ports-in-browsers/>.
20. COTTON, Michelle; EGGERT, Lars; TOUCH, Dr. Joseph D.; WESTERLUND, Magnus; CHESHIRE, Stuart. *Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry* [RFC 6335]. RFC Editor, 2011. Request for Comments, no. 6335. Available from DOI: 10.17487/RFC6335.

-
21. *Service Name and Transport Protocol Port Number Registry* [online]. IANA, 2024 [visited on 2024-02-14]. Available from: <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>.
 22. KLENSIN, Dr. John C. *Simple Mail Transfer Protocol* [RFC 5321]. RFC Editor, 2008. Request for Comments, no. 5321. Available from DOI: 10.17487/RFC5321.
 23. STEINER, Thomas. *URL protocol handler registration for PWAs / Chrome for Developers* [online]. Google, 2021 [visited on 2024-02-15]. Available from: <https://developer.chrome.com/docs/web-platform/best-practices/url-protocol-handler>.
 24. BREWER, Eric. *Towards Robust Distributed Systems*. 2000. Available also from: <https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf>.
 25. *What is the CAP theorem?* [online]. IBM, 2024 [visited on 2024-01-27]. Available from: <https://www.ibm.com/topics/cap-theorem>.
 26. *TypeScript: JavaScript With Syntax For Types* [online]. Microsoft Corporation, 2025 [visited on 2025-01-03]. Available from: <https://www.typescriptlang.org/>.
 27. *Node.js — Run JavaScript Everywhere* [online]. Node.js contributors & OpenJS Foundation, 2025 [visited on 2025-01-03]. Available from: <https://nodejs.org/>.
 28. *V8 JavaScript engine* [online]. The V8 project authors, 2025 [visited on 2025-01-03]. Available from: <https://v8.dev/>.
 29. *npm* [online]. npm, Inc., 2025 [visited on 2025-01-06]. Available from: <https://www.npmjs.com/>.
 30. *npm Docs* [online]. npm, Inc., 2025 [visited on 2025-01-06]. Available from: <https://docs.npmjs.com/>.
 31. *GitHub - containers/common: Location for shared common files in github.com/containers repos* [online]. The Podman project, 2025 [visited on 2025-01-05]. Available from: <https://github.com/containers/common>.
 32. MYSLIWIEC, Kamil. *Documentation / NestJS - A progressive Node.js framework* [online]. 2025. [visited on 2025-01-06]. Available from: <https://docs.nestjs.com/>.
 33. *Next.js by Vercel - The React Framework* [online]. Vercel, Inc., 2025 [visited on 2025-01-06]. Available from: <https://nextjs.org/>.

List of Acronyms

| | |
|----------------|--|
| ACID | Atomicity, consistency, isolation, durability |
| AI | Artificial intelligence |
| API | Application programming interface |
| BASE | Basically available, soft state, eventually consistent |
| CAP | Consistency, availability, partition tolerance |
| CLI | Command-line interface |
| CSS | Cascading Style Sheets |
| CSV | Comma-separated values |
| GUI | Graphical user interface |
| HTTP | Hypertext Transfer Protocol |
| IP | Internet Protocol |
| IANA | Internet Assigned Numbers Authority |
| IRI | Internationalized Resource Identifier |
| JSON | JavaScript Object Notation |
| JSON-LD | JavaScript Object Notation for Linked Data |
| OCI | Open Container Initiative |
| OLAP | Online analytical processing |
| OLTP | Online transaction processing |

A. LIST OF ACRONYMS

| | |
|-------------|-----------------------------|
| ORM | Object relational mapping |
| PWA | Progressive web application |
| SDK | Software development kit |
| SEO | Search engine optimization |
| UI | User interface |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| UUID | Universal Unique Identifier |
| UX | User experience |

Contents of the Digital Attachment

| | |
|----------------------------------|---|
| implementation..... | the source code of the implementation |
| └ booking-api-gateway..... | booking API gateway |
| └ booking-client-app | booking client application |
| └ booking-service..... | booking service |
| └ compose.prod.yaml | a production-mode Compose file |
| └ README.md | Markdown instructions to the implementation source code |
| thesis | |
| └ MT_Straka_David_2025.pdf | thesis text PDF |
| └ MT_Straka_David_2025..... | \LaTeX source of the thesis text |