FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

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.

Electronically approved by Ing. Jaroslav Kuchař, Ph.D. on 9 February 2023 in Prague.

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.

February 15, 2024

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

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, 2024. Also available from: ⟨`https://davidstraka.dev/masters-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 features an overview of select existing reservation systems, a requirements analysis, and an overview of state-of-the-art decentralized systems. The thesis 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, analýza požadavků a přehled moderních decentralizovaných systémů. 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

List of Figures

List of Listings

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 and select state-of-the-art decentralized systems.

Chapter **1**

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 Scheduling[1], Reservio[2], Square Appointments[3], and Wix[4].

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.

[1]Acuity Scheduling (`https://www.acuityscheduling.com`)
[2]Reservio (`https://www.reservio.com`)
[3]Square Appointments (`https://squareup.com/us/en/appointments`)
[4]Wix (`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 singing 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.

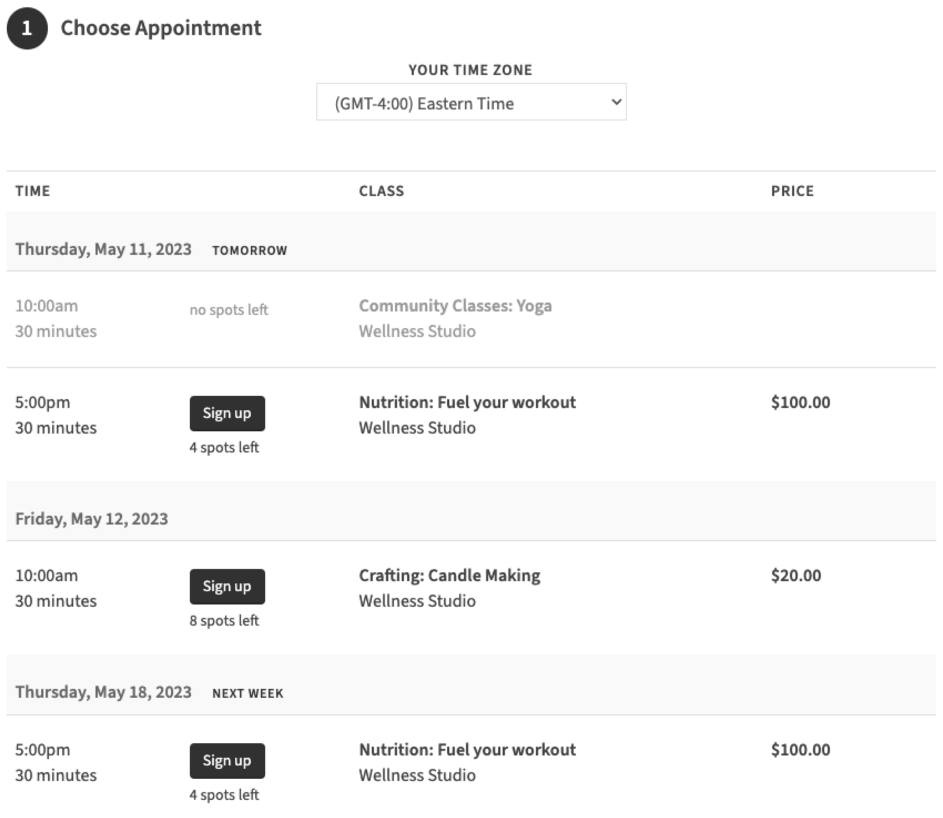
After the business user completes the sign-up process, customers can access a booking website that is created for the business. Upon accessing the website, customers see the business information (with only the sign-up completed, that is just the business name), followed by a list of appointments to choose 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 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	Sign up 4 spots left	Nutrition: Fuel your workout Wellness Studio	$100.00
Friday, May 12, 2023			
10:00am 30 minutes	Sign up 8 spots left	Crafting: Candle Making Wellness Studio	$20.00
Thursday, May 18, 2023 NEXT WEEK			
5:00pm 30 minutes	Sign up 4 spots left	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 online 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 subdomain (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.

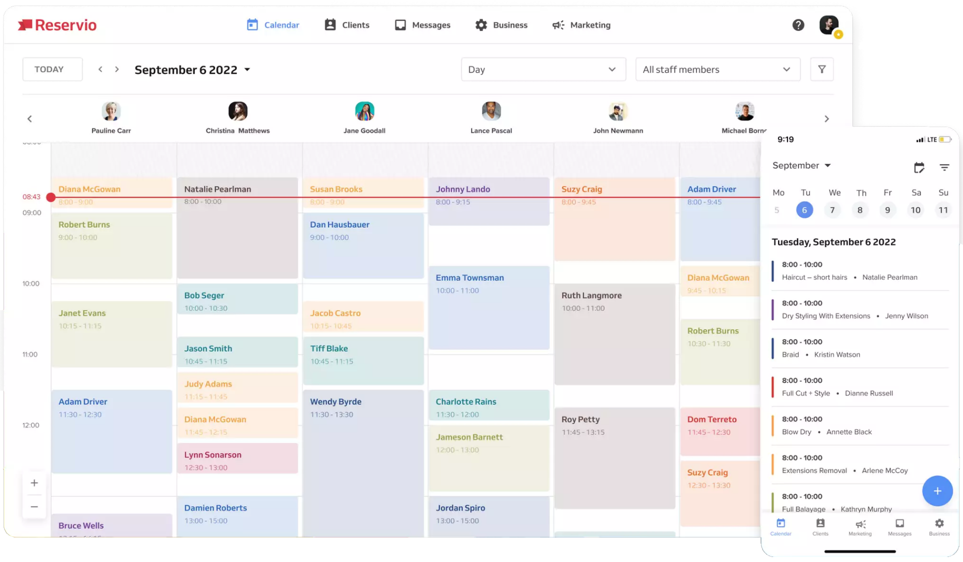

Reservio
Calendar
Clients
Messages
Business
Marketing
TODAY
September 6 2022
Day
All staff members
Pauline Carr
Christina Matthews
Jane Goodall
Lance Pascal
John Newmann
Michael Borne
September
Tuesday, September 6 2022
8:00 - 10:00 Haircut – short hairs • Natalie Pearlman
8:00 - 10:00 Dry Styling With Extensions • Jenny Wilson
8:00 - 10:00 Braid • Kristin Watson
8:00 - 10:00 Full Cut + Style • Dianne Russell
8:00 - 10:00 Blow Dry • Annette Black
8:00 - 10:00 Extensions Removal • Arlene McCoy
8:00 - 10:00 Full Balayage • Kathryn Murphy

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 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 waitlist 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.

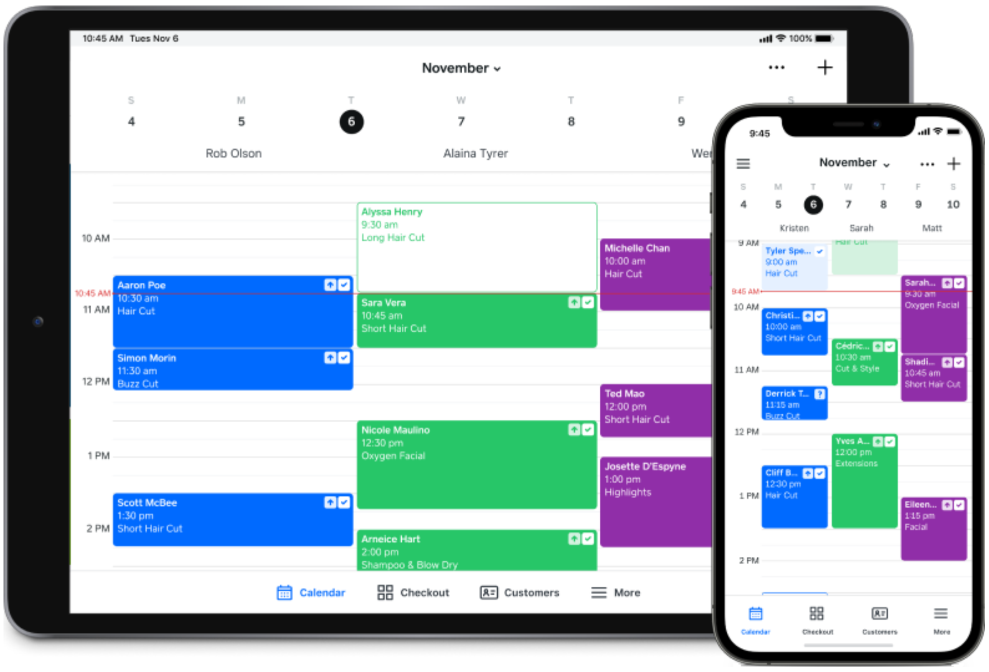


Figure 1.3: Square Appointments [6]

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).

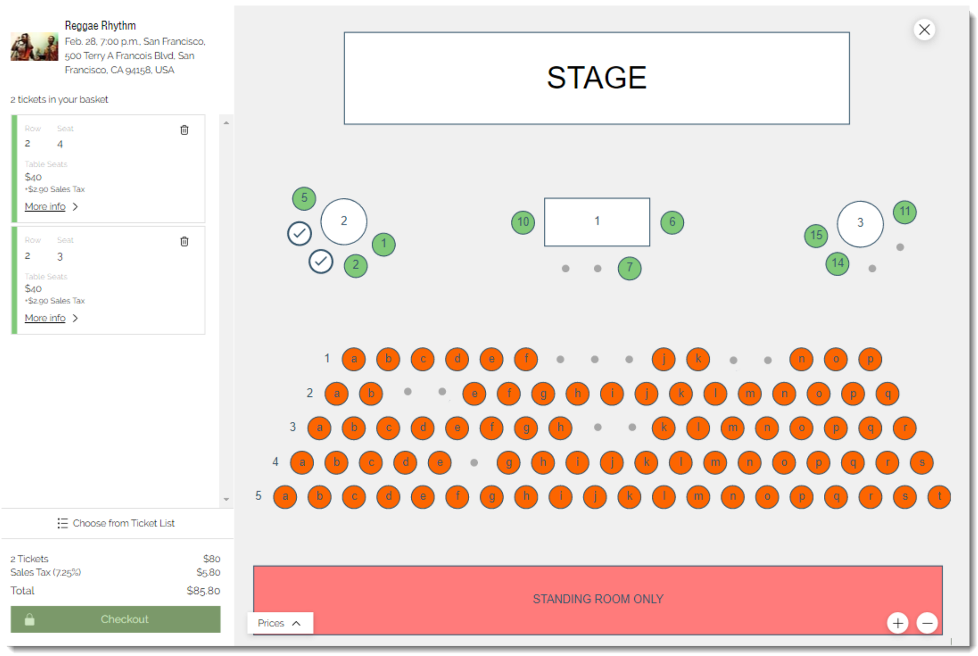
Events can have forms attached, which can include fields of multiple different data types. With the seating map tool, the user can create a detailed map of the seats and areas available at the event venue. They can then assign the available tickets to specific seats or areas. This seating map from the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



Reggae Rhythm
Feb. 28, 7:00 p.m., San Francisco, 500 Terry A Francois Blvd, San Francisco, CA 94158, USA

2 tickets in your basket

Row	Seat
2	4

Table Seats
$40
+$2.90 Sales Tax
More info >

Row	Seat
2	3

Table Seats
$40
+$2.90 Sales Tax
More info >

Choose from Ticket List

2 Tickets	$80
Sales Tax (7.25%)	$5.80
Total	$85.80

Checkout

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 waitlists 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.

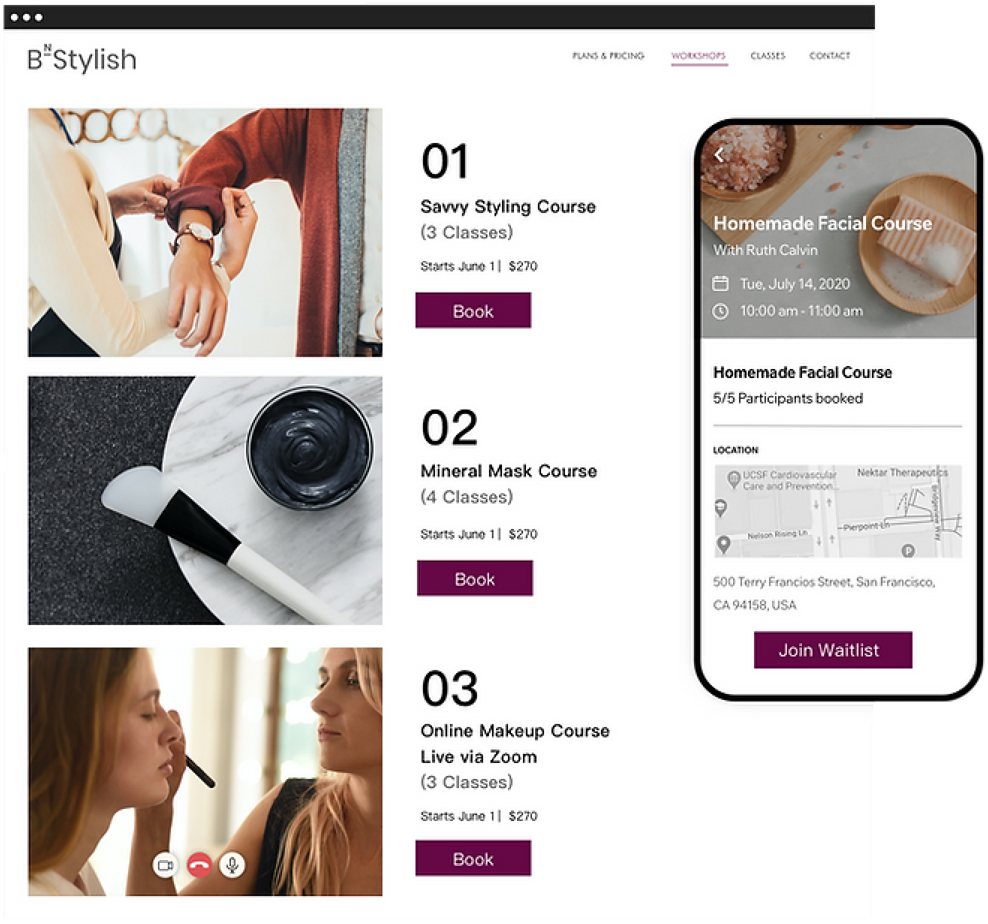


Figure 1.5: Wix Bookings [7]

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 synchronization 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.

Chapter 2

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 **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.

F6 **Booking edits** — A user can edit their existing bookings' form data.

F7 **Deadlines** — A user with an inventory can limit until when an existing booking of an inventory item can be booked or modified (which includes both changes to a booking and booking cancellations).

F8 **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.

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* **Waitlist** — A user can enter a waitlist 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 first user on the waitlist is automatically booked the item.

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 that 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 (*).

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.

Chapter 3

State-of-the-Art Decentralized Systems

This chapter describes select state-of-the-art online decentralized systems, which could either be directly used for creating the new system, or could serve as an inspiration in some way.

Chapter 4

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 Identifier (URI) 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.

4.1 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.

4.1.1 System architecture

Like some other decentralized systems, such as email or the Fediverse, described in chapter 3, 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 4.3.

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 4.1. The figure 4.2 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 `@vocab` directive for the JSON-LD context of the inventory's metadata (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*, *modifyDeadline*, 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.

  The attribute *modifyDeadline* is a timestamp of the date and time starting from when the item can no longer be modified (that is, edited or canceled), including the time zone, in the standardized ISO 8601 format. If the attribute is not set, the item can be modified 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.

- 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 items *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.

4.1.2 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

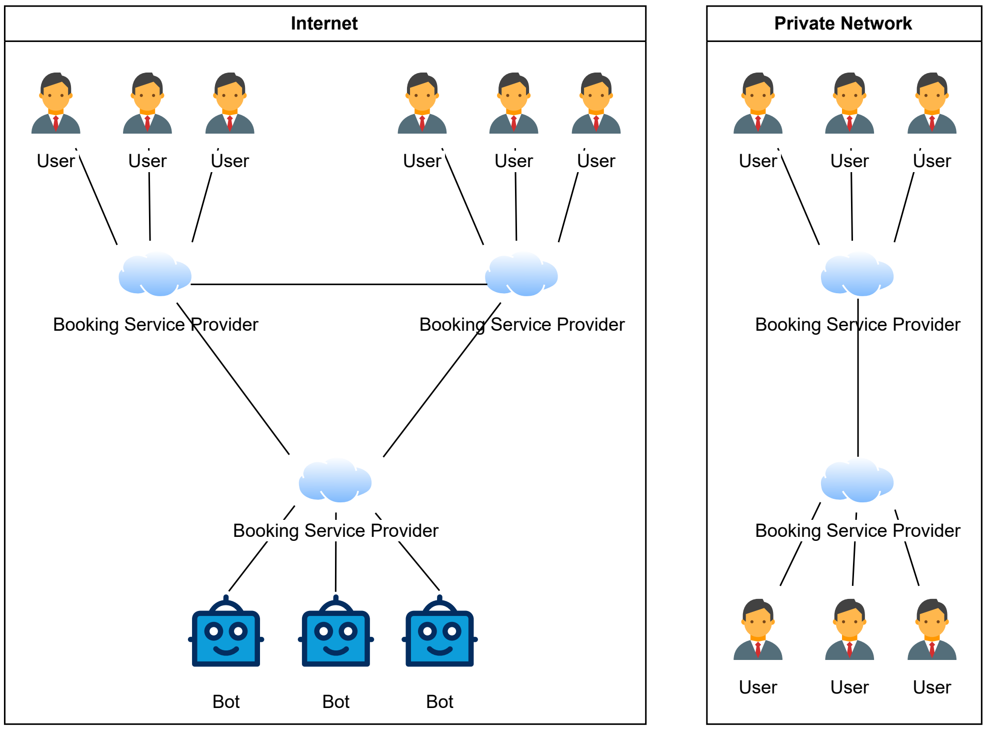


Figure 4.1: System network diagram

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 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 portion can contain the 26 letters of the English alphabet, numbers, and the following characters: the period (`.`), the minus sign (`-`), and an underscore (`_`). The username can be ex-

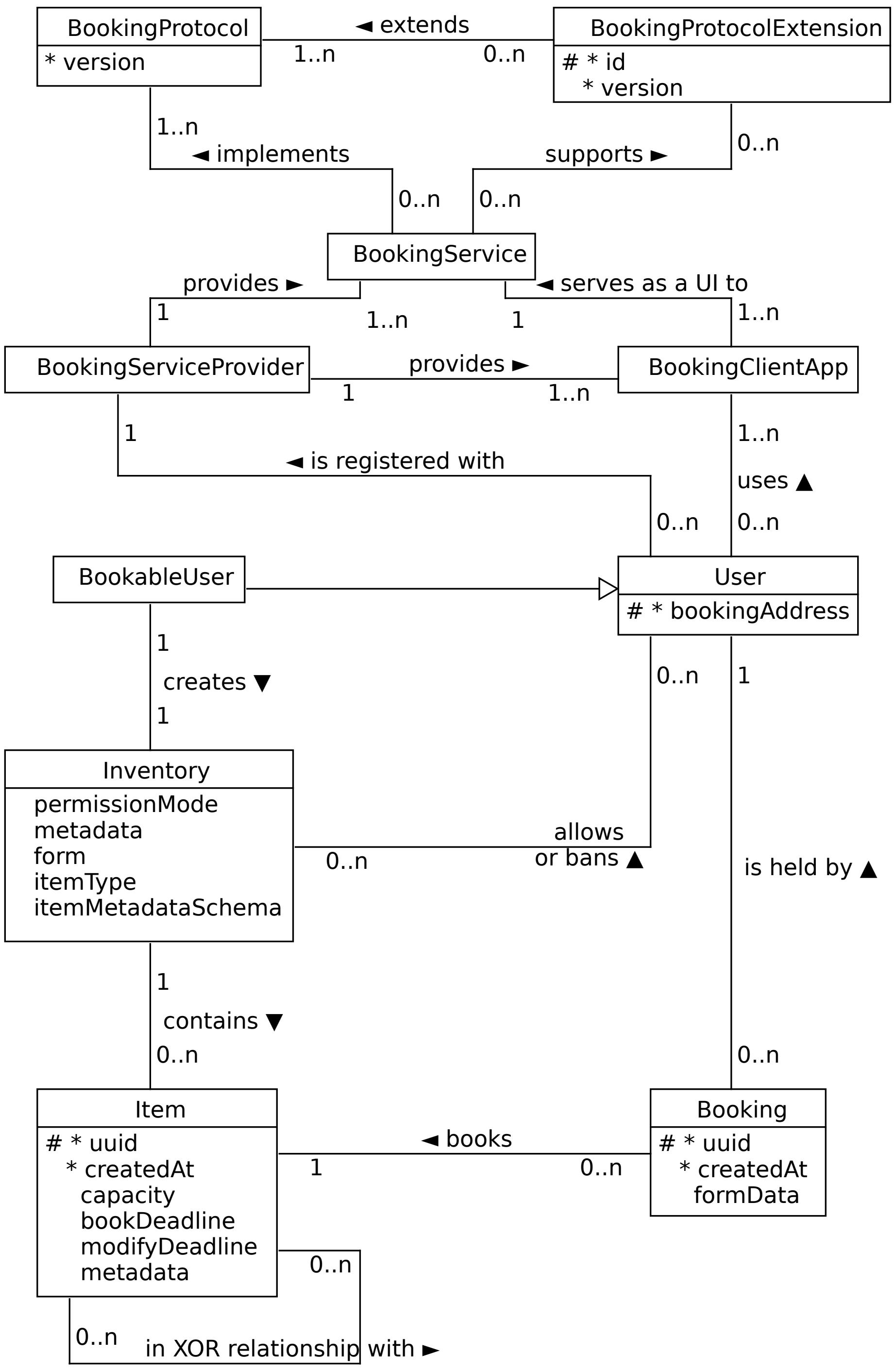


Figure 4.2: System conceptual schema

pressed by the regular expression `[a-zA-Z0-9._-]+`. The entire booking address can then be expressed as `<username>@<ip_address>[:port]` and `<username>@<domain_name>[:port]`. Example booking addresses are `alice@127.0.0.1`, `bob@localhost:3080`, and `john.doe98@example.com`.

4.1.3 Custom URI scheme

According to [22]: "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.

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 clicked links to a booking address, a custom URI scheme is also designed. [22] states that in order for a progressive web application (PWA) to register their ability to open or handle particular URI 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 feature-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, a path separated from the booking address by a forward slash (`/`), and possibly also query parameters separated from the booking address or path by a question mark (`?`), separated from each other by an ampersand (`&`), and separated from their values by an equals sign (`=`), as it is with the HTTP URLs.

The paths and query parameters will not be used in the initial version of the booking protocol, but extensions of the protocol as well as newer versions may utilize the path to direct the client application to a certain section and the parameters to provide additional functionality, such as form filling or item filtering and sorting.

The URI scheme can then be expressed as `web+booking://<booking_address>[/path][?param=1][&param=2][&...]`. Example URIs are `web+booking://alice@127.0.0.1`, `web+booking://bob@localhost:3080/a`, and `web+booking://john.doe98@example.com/a?b=1&c=2&d=3`.

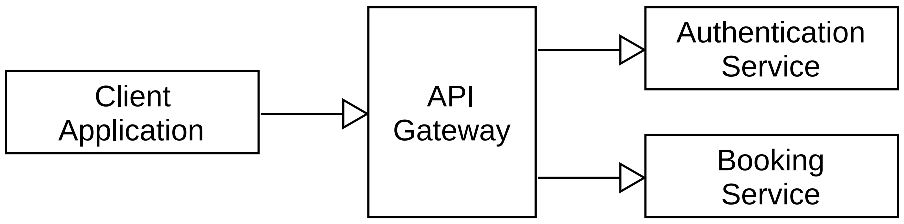


Figure 4.3: Architecture diagram

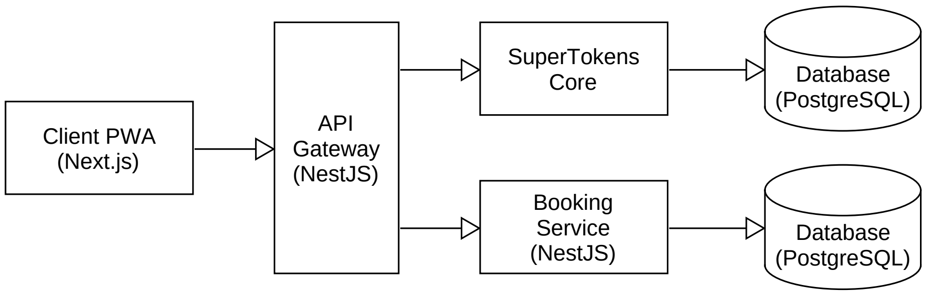


Figure 4.4: Detailed architecture diagram

4.1.4 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 4.3. The figure 4.4 then shows a more detailed diagram of the architecture featuring the technologies chosen for each service.

4.2 Booking Protocol

4.3 Booking Service

4.3.1 Database

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

```
interface Inventory {
    version: string; // Resource SemVer version
    extensions?: {
        [key: string]: string; // Pairs of extension ID
                               // and its SemVer version
    };
    meta?: object; // JSON-LD
    form?: object; // JSON Schema
    itemType?: string; // JSON-LD context
    itemMetaSchema?: object; // JSON Schema
    items?: Array<{
        id: string;
        xor?: string[]; // IDs of mutually exclusive items
        availability?: number;
        capacity?: number;
        bookDeadline?: string; // ISO 8601
        modifyDeadline?: string; // ISO 8601
        meta?: object; // JSON-LD
    }>;
}
```

Listing 4.1: Simplified definition of Schedule JSON using a TypeScript interface

4.3.1.1 Conceptual Schema

4.3.1.2 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).

4.3.1.3 ACID vs. BASE

According to [23, 24], 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 is absolutely essential. For example, an item available for booking must never be booked by more users than its capacity. Availability is also very important. For those reasons, the system must be designed with ACID properties in mind.

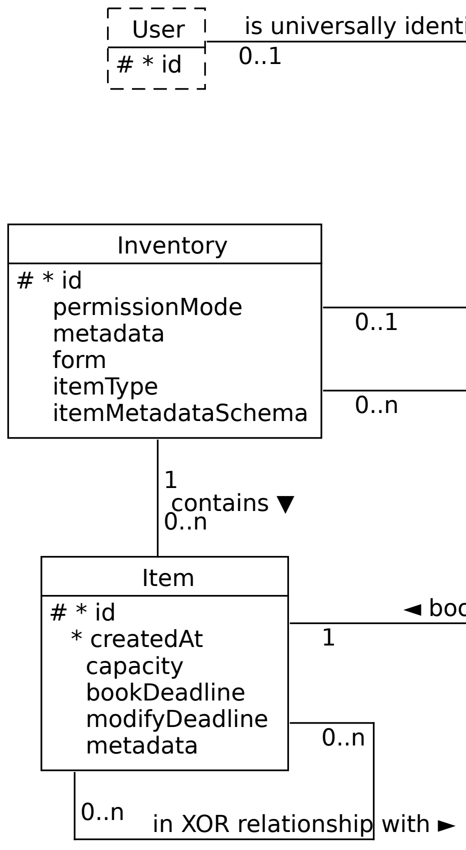


Figure 4.5: Conceptual schema of the booking service database

4.4 Client Application

A low fidelity prototype of the UI for the client application was created using Figma. The client application will be a single page progressive web application.

Chapter 5 Implementation

5.1 Booking Service

The diagram of the PostgreSQL database can be seen in the figure 5.1.

The booking service was implemented using the framework NestJS.

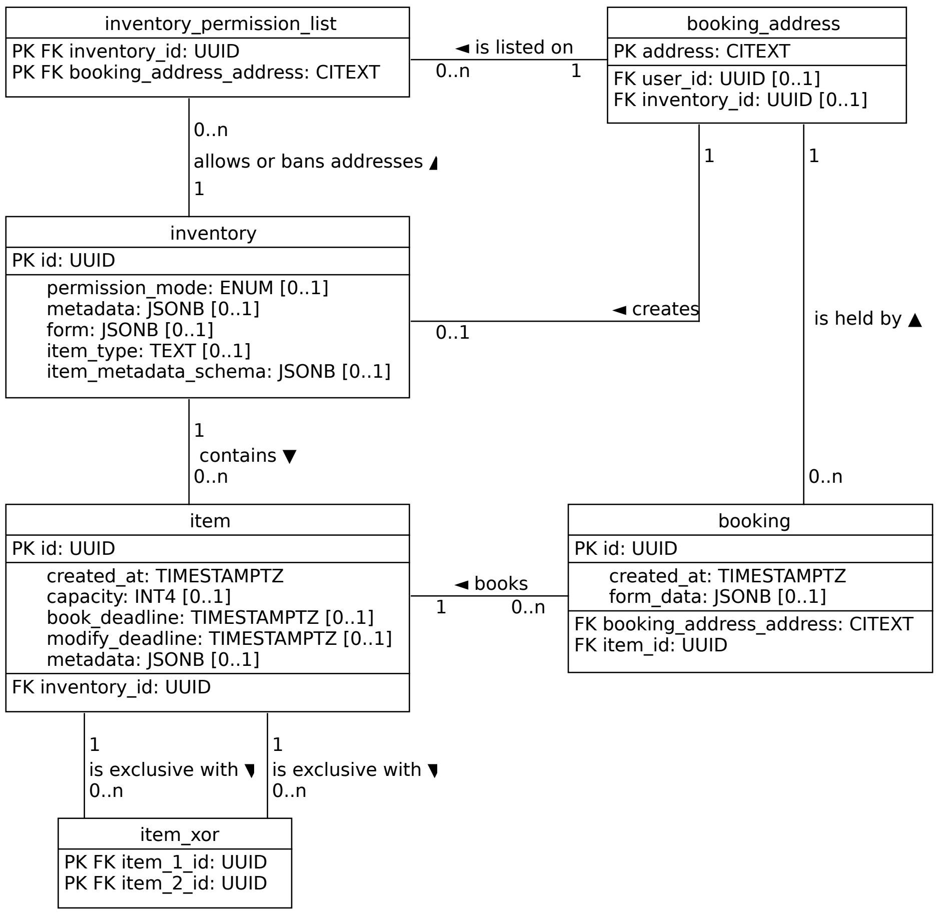


Figure 5.1: Diagram of the booking service database

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. Then, select state-of-the-art decentralized systems were briefly explored in chapter 3 to find out whether one of them could perhaps be used as part of the solution and to learn from their design. Chapter 4 describes the design of the newly created system, including its architecture, the common protocol, the back-end services, and the client application. Finally, chapter 5 describes the implementation of the system based on the created design, as well as its testing and evaluation.

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 overview of select state-of-the-art decentralized systems can introduce those systems to new people and help popularize the idea of decentralized systems as a whole. 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.

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*. Investopedia, 2023. Available also 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 Documentation* [online]. The Kubernetes Authors & The Linux Foundation, 2023 [visited on 2024-02-08]. Available from: `https://kubernetes.io/docs/home/`.
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. 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`.

23. BREWER, Eric. *Towards Robust Distributed Systems.* 2000. Available also from: `https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf`.

24. *What is the CAP theorem?* [online]. IBM, 2024 [visited on 2024-01-27]. Available from: `https://www.ibm.com/topics/cap-theorem`.

Appendix A

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

OLAP Online analytical processing

OLTP Online transaction processing

ORM Object relational mapping

PWA	Progressive web application
SEO	Search engine optimization
UI	User interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universal Unique Identifier
UX	User experience

Appendix **B**

Contents of the Digital Attachment

- `design`
  - `figma.fig` .... Figma design file with a lo-fi client app prototype
- `implementation` .... the source code of the implementation
  - `booking-api-gateway` .... booking API gateway
  - `booking-client-app` .... booking client application
  - `booking-service` .... booking service
  - `booking-service` .... booking service
  - `compose.prod.yaml` .... a production-mode Compose file
  - `README.md` .... Markdown instructions to the implementation code
- `thesis`
  - `MT_Straka_David_2024.pdf` .... thesis text PDF
  - `MT_Straka_David_2024` .... LaTeX source of the thesis text