Calendar Manager: An Open Source Extensible Web Application

Matej Refka

A Software Engineering Project

Submitted September 2021
for the degree of
MSc Advanced Computer Science

Supervisor
Dr Brian Mitchell

Inspector
Dr Mian Hamayun

UNIVERSITY OF
BIRMINGHAM

Abstract

This project implements an open source, extensible calendar manager web application. The application uses Google Calendar API, which populates the user's calendar with events. A booking system is appended to this calendar, enabling the user to construct booking slots. The booking slots are shareable with outsiders who can register onto any of these slots. Any booked slots are then written back to the owner's Google calendar. The calendar manager is perceived as a skeleton application because it is easily maintainable and extensible. To achieve these properties, software architecture and software design principles have been researched and implemented within the calendar manager application. Additionally, a thorough evaluation of these design decisions has been made, which is presented within this report. This report can be utilized as a complete documentation of the calendar manager application.

Keywords Software architecture, Software design, .Net 5, ASP.Net Core, Vue.js, School of Computer Science, University of Birmingham, UK, Calendar Manager: An Open Source Extensible Web Application

Table of contents

Abstract i

1 Introduction 1
1.1 Overview 1
1.2 Limitations 1
1.3 Definitions 1
1.4 Report structure 2

2 Background 3
2.1 PROMPT 3
2.2 Reference web applications 3
2.3 Calendar API 4
2.4 Server side framework 4
2.5 Software architecture reference 5
2.6 Application architecture 5
2.7 Server side solution structure 6
2.8 Web technologies 7
2.9 Web accessibility reference 8

3 High level system overview 8
3.1 Calendar skeleton 9
3.2 Booking system 10

4 System implementation 12
4.1 User authentication and authorization 12
4.2 API 15
4.3 Database 19
4.4 Routing 20

5 System architecture 21
5.1 Dependency injection 21
5.2 Revised project structure 21
5.3 Browser-side HTML design 23
5.4 Browser-side Vue.js design 25
5.5 Documentation 27

6 Results 28
6.1 User interfaces 28
6.2 Software architecture 31
6.3 Accessibility 31

7 Analysis 32
7.1 Encapsulation principle evaluation 32
7.2 Single responsibility principle evaluation 32
7.3 Don't Repeat Yourself (DRY) principle evaluation 34
7.4 Testing 35
7.5 Accessibility testing 36
7.6 Documentation revised 38

8 Conclusions 39
8.1 Summary of the project . . . 39
8.2 Future work . . . 39
8.3 Personal development . . . 40
8.4 Reflections . . . 40

A Test plan 42
A.1 Server-side testing . . . 42
A.2 Client-side use-case testing . . . 43
A.3 Usability testing . . . 45

B Source code 46

References 47

List of tables

1 Routing table within the routing middleware . . . 21
2 Predefined software design principles . . . 31
3 Contrast ratio of components within Calendar Manager . . . 31
4 Unit tests . . . 42
5 Integration tests . . . 42
6 Functional tests #1 . . . 43
7 Functional tests #2 . . . 44
8 Functional tests #3 . . . 45

List of figures

1 MVC implementation within ASP.NET . . . 6
2 Project solution . . . 7
3 Default calendar sketch . . . 10
4 Booking slot pop-up sketch . . . 10
5 Booking slots sketch . . . 11
6 Booking calendar sketch . . . 12
7 OAuth 2.0 authorization protocol . . . 13
8 Desired program flow . . . 18
9 Actual program flow . . . 19
10 Underlying database . . . 20
11 Circular dependency . . . 23
12 HTML semantic structure . . . 25
13 Vue.js MVVM design . . . 26
14 Vue.js life-cycle (simplified) . . . 27
15 Calendar UI . . . 29
16 Booking UI . . . 30
17 Single responsibility principle violation . . . 33
18 Single responsibility principle enforced . . . 34
19 Proposed calendar component system . . . 35
20 Desired testing pyramid (left) vs actual testing distribution (right) . . . 36
21 Proposed documentation coverage . . . 38

Acknowledgements

First and foremost, I would like to thank my supervisor, Brian Mitchell, for undertaking the role as my supervisor for this project. His guidance, support, and encouragement throughout my Master's year has been invaluable.

I would also like to extend my thanks to my inspector, Mian Hamayun, who has been providing me with valuable feedback at different stages of this project.

Finally, I would like to thank my friends Ema Csutorova, Joseph Morphew, and my parents for their continued support throughout my academic life.

1 Introduction

1.1 Overview

This project implements an open source, extensible calendar manager web application. The calendar manager application consists of an integrated calendar and a booking system. The booking system implements a feature where any booked events are written back to the owner's calendar, which is the application's unique selling point. To the author's best knowledge, no calendar scheduling web application supports such feature, as of this writing.

A strict time limitation has been enforced upon this project, thus a full scale calendar manager web application cannot be implemented. Therefore, this project is made publicly available, inviting further extensions by outside parties. The incentive to build upon such project comes from the lack of similar open source projects. The amount of open source business or commercial software is far lesser than open source research or development software.

As a result, this project is split up into two main components, of equal importance. The first component is the calendar manager web application. The application comprises of a calendar, populated with events from a selected third-party calendar API, and an events booking system, implementing the aforementioned unique feature. The second component ensures that all code written for the first component is suitable to be publicly available, by addressing security concerns and code usability. This code is implemented and evaluated using appropriate software architecture, design, and principles to ensure maximum code maintainability and system extensibility.

1.2 Limitations

The implemented web application is a skeleton calendar manager, functioning as a structural framework for future extensions. This can be viewed as a minimization problem. All calendar and booking features are minimized, while the predefined functionality is fully implemented. The predefined functionality involves creating a calendar, populating this calendar with events pulled from a selected third-party API, allowing the addition of bookings, sharing these bookings with chosen users, and writing any booked slots back to the owner's calendar. Additional proposed features based on initial background research are presented within '8.2 Future work' section.

1.3 Definitions

The naming convention of web technologies can be confusing, because the labels describing hardware and software can be used interchangeably based on a scenario or a view point. For example, a client is connecting to a server doggo.com to view images of dogs. The client doggo.com is connecting to a database server to retrieve images of dogs. A clear definition of terms is needed:

Browser/user refers to client-side, front-end of Calendar Manager web application.

Client refers to server-side, back-end of Calendar Manager web application. The server-side of Calendar Manager often acts as a client to external servers within this project.

Server third-party, external servers, such as APIs and databases.

Application refers to the combination of client-side and server-side of Calendar Manager web application.

User refers to an end user utilizing Calendar Manager application service.

1.4 Report structure

§1 Introduction briefly describes the implemented web application and its functionality. It also provides motivation for the two predefined components of this project, the calendar manager application, and software architecture. This section also draws boundaries around these two components to eliminate any preconceptions about what they do not solve, and draw attention to what they do solve. Additionally, a list of definitions is supplied, which is vital and should be used as reference material throughout this report.

§2 Background collates a list of resources required for the two predefined components. The web application requires a list of technologies, whilst software architecture requires a list of software principles and references that the application should adhere to. Lastly, this section also covers preliminary design choices, required for the application setup.

§3 High level system overview gives a high level overview of the implemented calendar manager web application. The system overview is described against application sketches as they highlight the key functionality better than the implemented system.

§4 System implementation provides intricate explanations of how Calendar Manager functions. This covers back-end implementations, including data handling, routing and security. This section aims to cover the first component of this project.

§5 System architecture details the second component of this project. It describes the design choice made within both, the back-end, and the front-end of Calendar Manager, and provides justifications for these.

§6 Results contain the deliverables of the first component, the user interfaces. It also contains results of the second component; evaluating whether Calendar Manager adheres to the predefined design principles.

§7 Analysis evaluates the calendar manager application via unit testing, integration testing and functional testing. Software design and software principles are also evaluated within this section.

§8 Conclusions recaps the project and evaluates whether, and to what extent, the project aims were met. It also proposes additional features that could be appended onto the calendar manager application. Lastly, a personal reflection on the whole project is made.

2 Background

2.1 PROMPT

The Internet is vast (de Kunder, 2016). It has become the largest medium of information, built upon its key concept of accessibility. It is extremely easy to contribute and access information on the Internet. An inherent flaw with this, is the lack of information filtering, leading to equivalently accessible misinformation. This project is strictly serving for educational purposes, with this report and all the code publicly available. The secondary aim is to make the system maintainable and extensible, enabling future work. It is therefore imperative that modern software practices are utilized, and all the information collected throughout this project is accurate. To achieve this, critical analysis, and evaluation of all sources of information must be conducted.

The chosen evaluation strategy is PROMPT (Huckle, 2019). The PROMPT system considers evaluating six aspects of a given source: presentation, relevance, objectivity, method, provenance, and timeliness. To avoid repetition, only the criticism is highlighted. The methodology of how the six aspects were derived is not included in this article.

To ensure sufficient entropy, a comparison between the chosen technology or methodology and the alternatives has been done. The alternatives considered were CRAAP (Myhre, 2012) and RADAR (Mandalios, 2013). All three methods cover the same aspects of information evaluation, only differing with the use of acronyms. CRAAP publication has 5 citations, whilst the RADAR publication has 54 citations. PROMPT is an article on The Open University. The presumption is made that this article has more exposure than the RADAR publication and therefore would be considered as more standardized and more appropriate for this project.

2.2 Reference web applications

This project implements a calendar interface and a simple booking system. Both of these components require reference calendar applications. These references will aid throughout the project with interface design and system implementation. The system implementation of these reference applications will assist when making Calendar Manager system design choices. For example, designing the functionality of a calendar grid.

The pool of reference applications for the calendar interface include Google Calendar, Microsoft Outlook Calendar, Calendar.com and Apple Calendar. These were chosen because they are the most popular calendars, with most calendar users using their interfaces. Following the design of these standardised interfaces ensures that the Calendar Manager interface is inherently easy to use. All four calendars have almost an identical interface design, therefore all four are viable for use, interchangeably, as the interface reference.

The initial references for the booking system are Calendar.com, Calendly and Doodle, as the most popular calendar manager web applications. The aim is to minimize the booking system functionality, while implementing a unique feature. The proposed feature that none of these booking applications implement, is the ability to write booked events back into the original user's calendar. This feature will be the unique selling point of Calendar Manager.

2.3 Calendar API

The default Calendar Manager calendar interface will be populated with events collected from a third-party calendar. The events will be pulled into Calendar Manager via the third-party API. The initial pool of third-party calendars consists of Google Calendar, Microsoft Outlook Calendar, Calendar.com and Apple Calendar, the most popular calendars. Targeting the most popular calendar will ensure that Calendar Manager has the largest possible user base.

Calendar.com does not provide an API for its events. Apple Calendar provides its EventKit Calendar API, Google Calendar provides its Google Calendar API and Outlook Calendar provides its Graph API. The EventKit Calendar API has a brief documentation, compared to the detailed documentation of the other two APIs. Additionally, the documentation shows the API requests in Swift and Objective-C languages, neither of which are considered to be used for Calendar Manager. The Google API calls are documented in Java, whereas the Graph API calls are documented in C#. Both of these languages are considered for the Calendar Manager implementation and both APIs are well documented. To select one of these two, a comparison of its total users is made. The best available metric is the App Store number of downloads. Outlook Calendar has over 500 million downloads, whilst Google Calendar has over one billion downloads. Therefore, the chosen third-party calendar is Google Calendar.

Since the chosen API is Google Calendar API, the selected interface reference web application is now Google Calendar, purely out of convenience throughout development.

2.4 Server side framework

The available options for the server side framework are Express which uses Node.js, Spring which uses Java, and ASP.NET which uses C#. This initial pool is comprised of languages that I have at least some experience with, and their corresponding frameworks. Calendar Manager will use the server side extensively for HTTP requests and responses, handling API response objects, customising endpoints for the dynamically generated booking calendar URL, as well as contain business logic. I am most confident with C# and have minor experience with ASP.NET. To ensure the deliverables are met on time, and at the highest possible standard, the chosen server side framework is ASP.NET.

2.5 Software architecture reference

A substantial part of Calendar Manager revolves around software architecture. This is to ensure that the application is maintainable, scalable and extensible. The written software should follow modern design principles and practices. Microsoft documentation provides a reference to an e-book ”Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure” (Smith, 2021). This e-book covers the current .NET developer platform, which will be used for the Calendar Manager project. An alternative to this e-book is an article recommended by the e-book, which covers recommended programming principles (Kappert et al., 2021). The official Microsoft e-book has been chosen because it is much more extensive and is specific to ASP.NET framework. This e-book, in combination with Microsoft's ASP.NET documentation (Anderson et al., 2021), will be used as a reference throughout the project's development cycle, starting from the next subsection to the evaluation of the project.

2.6 Application architecture

The two high-level approaches to web application architecture are Model-View-Controller (MVC) pattern, and Single Page Application (SPA) pattern (Smith, 2021).

MVC heavily relies on server-side the functionality. This includes business logic, accessing data, making requests and parsing responses. The only functionality handled by the browser-side is web page reactivity. The Controller handles requests and responses. The Model handles data through data objects. The View handles user interface. Within MVC, the models are passed between the controllers and the views, allowing the server to pass data to the browser and vice versa (Smith, 2021).

SPA involves the use of browser-side for all application business logic, routing, data retrieval, making requests, and processing responses. Because all application logic is executed within the browser, the benefit is a reduced network load, since there are no back and forth calls transferring data between the browser and the server. The biggest downside with SPA is the difficulty in securing sensitive information. Exposing confidential data to the browser is generally a bad practice and can lead to security vulnerabilities (Smith, 2021). Calendar Manager will include user authentication and authorisation, as well as making API calls. These functions will inherently involve confidential data which will need to be secured, such as API keys or authorization tokens. Additionally, I am much more confident in my server-side than browser-side abilities. Therefore, the chosen application design pattern is MVC.

The MVC implementation within ASP.NET is entangled with its routing system. ASP.NET allows the addition of middleware to the application pipeline (Smith, 2021). This middleware is executed on every request made by the web application. The routing middleware parses and routes all requests into an appropriate endpoint within the application (Anderson & Smith, 2020). An example below is added for clarification. The browser makes a request to the Calendar Manager host (CM) to get the home page, specified by the path '/Home/Index'. This request is passed into the application pipeline. First, the explicitly defined middleware parses this request. One of the defined middleware is the routing middleware which routes this request into an application endpoint. The endpoint is an 'Index' action (method) of the 'Home' controller. The Index action passes a Model into the View, which is then returned and served back to the browser.

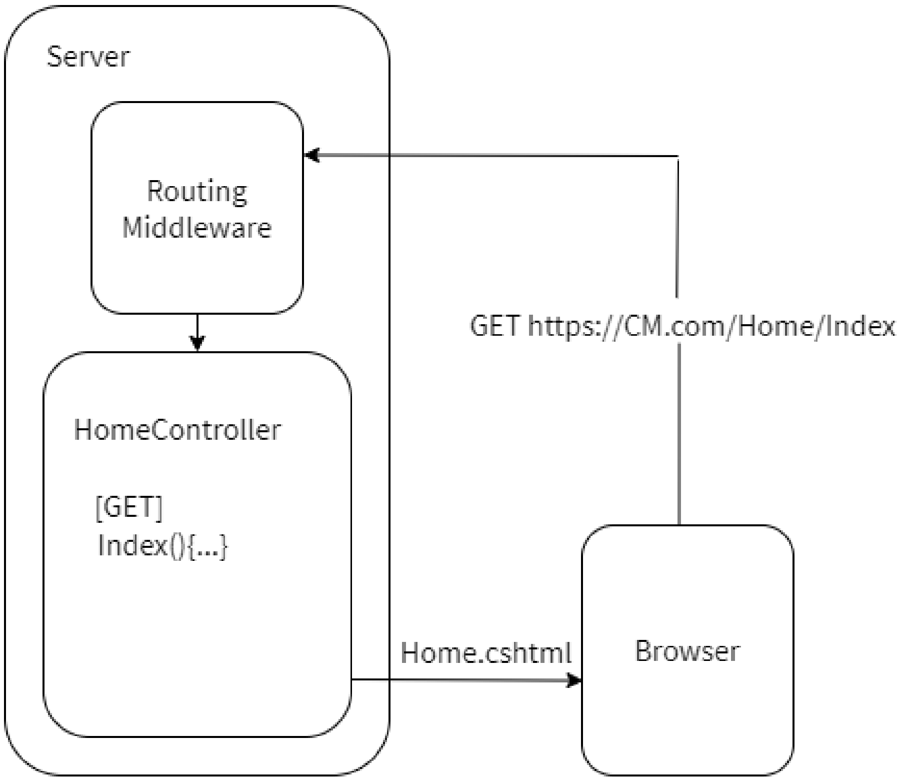


Figure 1 MVC implementation within ASP.NET.

2.7 Server side solution structure

The calendar manager application can be structured within a single layer, or as a combination of multiple layers. Layers split the application logic, where each layer contains the corresponding low-level functionalities (Smith, 2021). Calendar Manager will be split into three layers. The first layer will contain the Controllers and the Views. The second layer will hold all business logic required for the first layer. This layer will be implemented as a class library. The class library and its logic can be reused with other projects. The final layer will hold unit tests which will be used to test the business logic of Calendar Manager.

Layered architecture provides structure to the application. At a high-level, it also adheres to the single responsibility principle, which states that each component should only have responsibility over a single part of Calendar Manager's functionality (Smith, 2021).

Within ASP.NET, a project solution holds application layers which are referred to as 'projects'. The layered architecture introduces dependencies to link these projects together. The Calendar Manager project is dependant on the Calendar Manager class library and the Calendar Manager unit tests are dependant on both, the Calendar Manager project and the Calendar Manager class library.

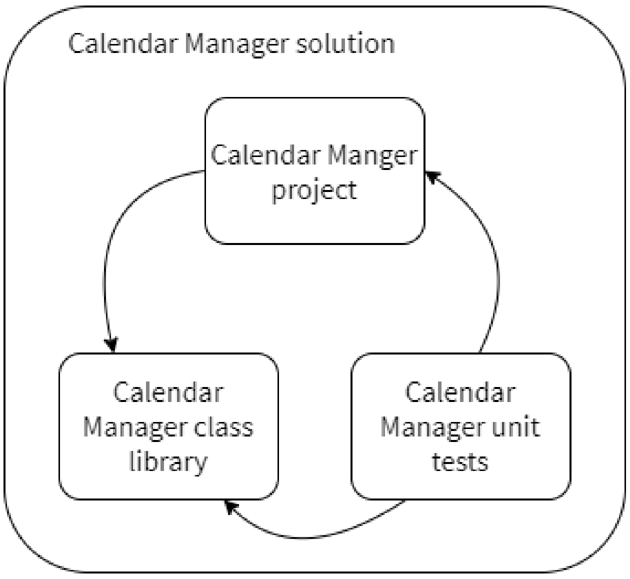

Calendar Manager solution
Calendar Manger project
Calendar Manager class library
Calendar Manager unit tests

Figure 2 Project solution.

2.8 Web technologies

The majority of modern web pages are constructed with the combination of HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript. HTML defines the structure, CSS defines the appearance and JavaScript defines the behaviour of a web page.

The standard for page structuring is HTML 5, which will be used. There are several technologies and options available for styling. Syntactically awesome style sheets (Sass) is a preprocessor to CSS. The Sass code is lower level, which is compiled into CSS language. This allows the user to utilize lower level constructs such as variables, nesting or functions. There are also CSS frameworks available, including Tailwind CSS, Bootstrap and Bulma. These frameworks provide predefined element classes. This is a convenient method to style elements, however the predefined classes tend to clutter HTML code, therefore CSS frameworks will be omitted. The use of Sass is preferable as it allows code re-usability. However, styling is the lowest priority within this project, and due to project time constraints, standard CSS will be used.

The browser-side functionality of Calendar Manager will be handled by a combination of Razor and JavaScript. Razor is used to load server-side models into the HTML Document Object Model (DOM). It is also used to post models from browser-side back to the server-side. JavaScript will be used to handle the browser-side functionality of the Calendar Manager interfaces. Calendar Manager is expected to grow in functionality by extending the system with more features. These extra calendar features will mostly be implemented in browser-side. Therefore a front-end framework for JavaScript is needed to enforce structure, enabling code reusability and better readability. The most popular JavaScript frameworks are Angular, React and Vue.js. These are assumed to have the best documentation and support. All three are well supported and viable for this project (Pekarsky, 2020). The chosen framework is Vue.js because it is assumed to be the easiest to learn, which is a beneficial attribute as this project has strict time constraints.

The official Vue.js (You et al., 2021) documentation will be referenced throughout the browser-side development process. This will aid with providing examples of how to use and utilize the framework, but it will also ensure that appropriate design within the framework is enforced. A reference to HTML and CSS is also required. The two considerations for this reference are w3schools (W3Schools-Contributors, 2021) and MDN Web Docs (MDN-Contributors, 2021). Both of these resources are considered as the standard for web references and web standards. In terms of provenance, the w3schools resource references the World Wide Web Consortium (W3C) in its own name, while it is not associated with them. Additionally, the .NET documentation references the MDN Web Docs rather than w3schools. The MDN Web Docs are preferred and chosen.

2.9 Web accessibility reference

Web accessibility is important within any website, as it allows equal access to all users, regardless of their disabilities. The accepted standard for web applications are Web Content Accessibility Guidelines 2.1 (WCAG 2.1) (Kirkpatrick et al., 2018). The WCAG 2.1 defines three conformance categories: A (lowest), AA (medium), and AAA (highest) for each accessibility component. It is difficult to achieve AAA conformance across all accessibility components. For this reason and the limited time frame of this project, the WCAG 2.1 will only be used to evaluate the final product. This evaluation will then act as the basis for further extension of Calendar Manager, in terms of accessibility.

3 High level system overview

The Calendar Manager functionality can be broken up into two main implementations. First, a skeleton calendar which is dynamically populated with user's Google events, and second, a booking system.

3.1 Calendar skeleton

The home page of Calendar Manager is a simple login page, where the user signs in with their Google account. Upon successful user authentication and authorization with sufficient permissions granted to access the user's Google calendar, the user's events are pulled from Google calendar API. The collected events are passed to the browser and the user is redirected to the calendar page. The skeleton calendar is constructed dynamically using the current date on page load. The current date is then used to compute and populate the seven dates in the current week, with their corresponding day names (Monday-Sunday). The current week and date are then used to compute and populate the current month or two overlapping current months. The current month is used to compute and populate the current year or two overlapping years. Navigating the calendar one week forward or backward adjusts the current day by seven days and executes the previous functions on this new date again. The 'previous' button is disabled upon reaching the year 2009, because Google calendar has been released to the public in July 2009. There are no constraints placed on the 'next' button.

The calendar grid is comprised of 288 buttons, each representing five minutes of the week. The buttons are also dynamically generated using Vue.js 'v-for' loop, where each button is assigned a unique reference. The decision has been made to have the smallest time unit five minutes rather one minute to avoid visual clutter. Additionally, both Google Calendar and Doodle use five minutes as the smallest visual interval for their calendars. Google calendar displays all events that span over three days outside of the main calendar to avoid clutter. Calendar Manager also filters out all events spanning over three or more days from the events list. The list of long events can be lengthy, depending on how long the user has been using Google calendar. Therefore, all past long events are deleted, and the remaining future long events are added to a list in the side panel, in an ascending order of the event's start date. Examples of long events include holidays or religious events. Having the long events side by side with the main calendar, without cluttering it, can aid the user with choosing appropriate booking slots. The remaining events list is processed further to capture all events corresponding to the current week. These events are then matched to their corresponding buttons in the calendar grid based on the button references. This allows the mapping of events to the calendar.

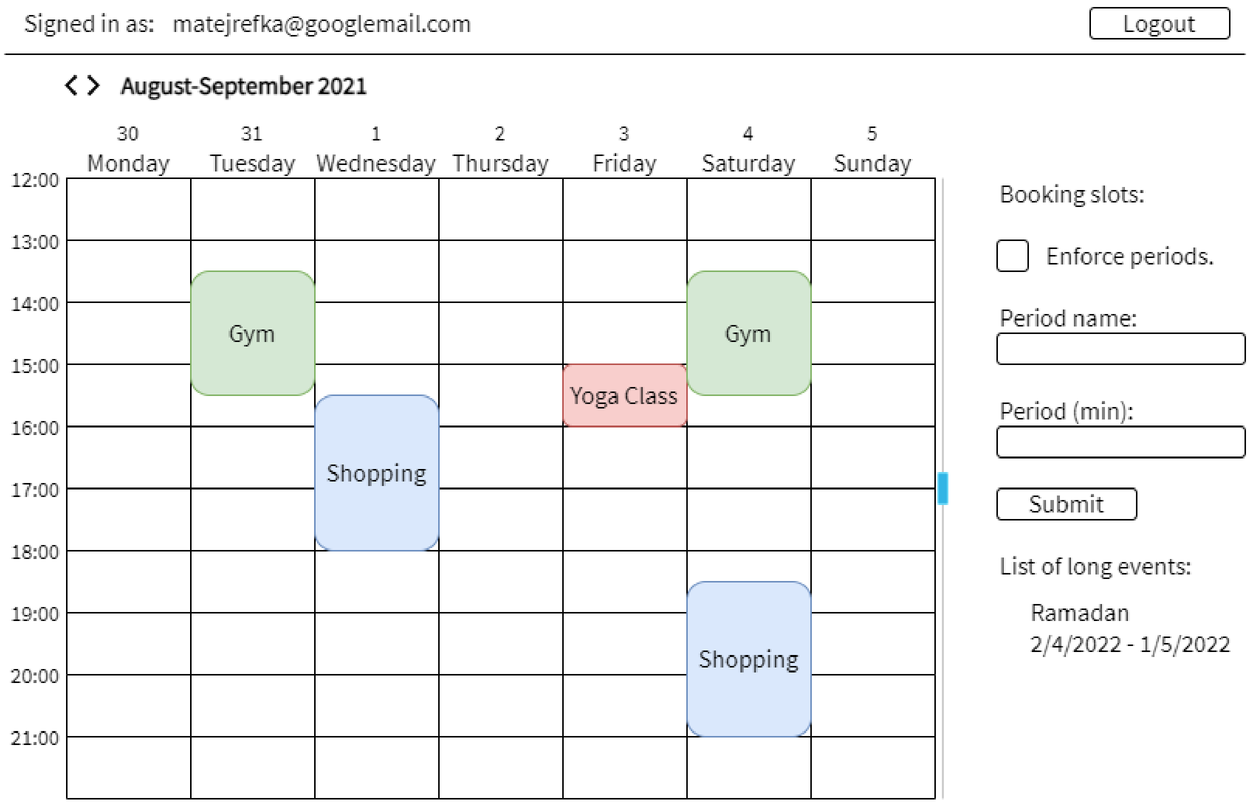


Figure 3 Default calendar sketch.

3.2 Booking system

The mapped events are baked into the calendar and are disabled. The rest of the calendar grid however, is still clickable, replicating Google calendar functionality. Instead of adding a new event by clicking on an empty space within the calendar grid, Calendar Manager adds a new booking slot. This is accomplished via a booking slot pop-up, where the user enters a time period for their selected slot.

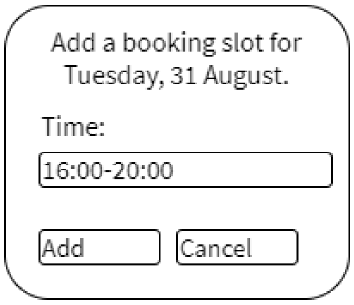
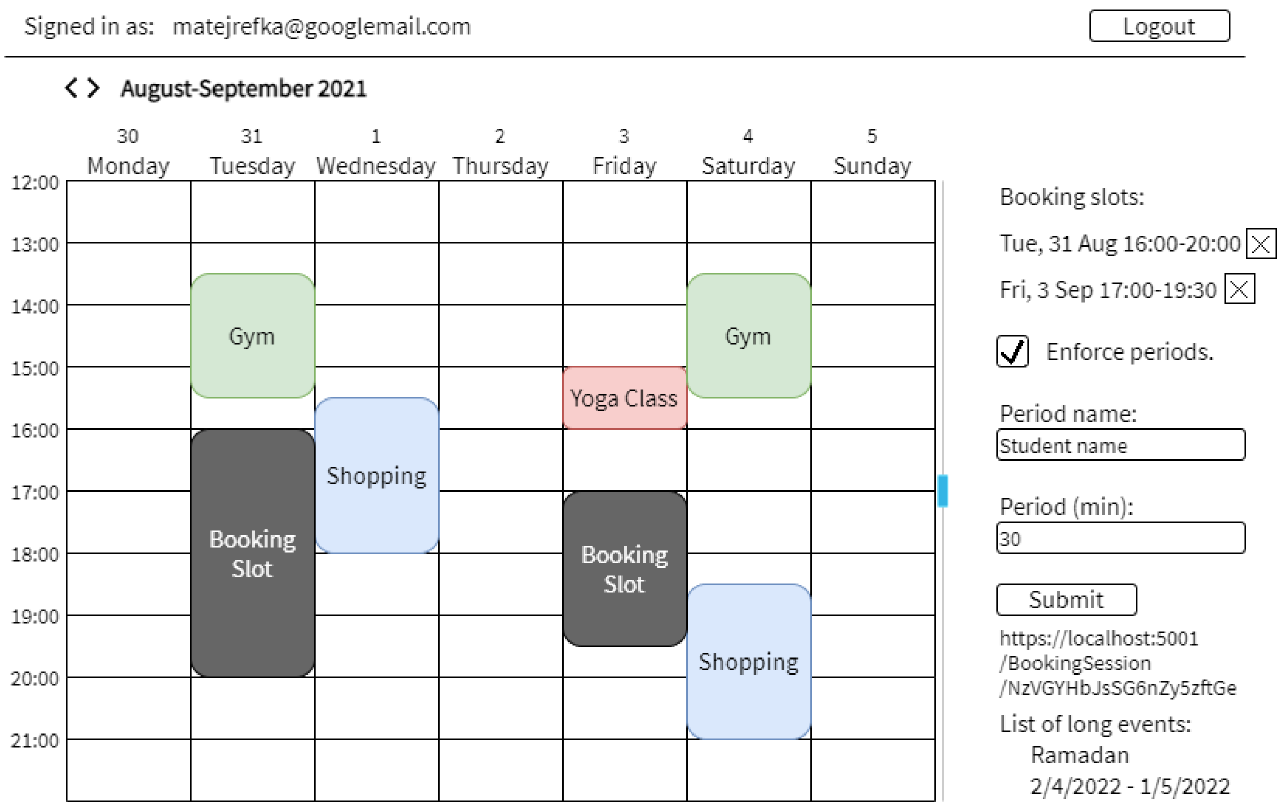


Figure 4 Booking slot pop-up sketch.

Upon the addition of a new booking slot, a custom booking slot event is added to the original calendar. The user is able to add multiple booking slots. The booking slots are displayed in a side panel list, in an ascending order. Each booking slot in the list is accompanied by a corresponding ‘remove’ button. On ‘remove’, the booking slot is removed from the list, and the booking slot event is removed from the calendar. ‘Enforce period’ checkbox allows the user to divide their chosen booking slots into smaller periods. These periods are assigned a user-defined period name and period length. An example below demonstrates a teacher creating two booking slots, which are then split into 13 periods for 13 students. On ‘submit’, a dynamically generated URL for this specific booking session is generated. Within the given example, the teacher then shares this link with the 13 students.



Signed in as: matejrefka@googlemail.com

Booking slots:

Tue, 31 Aug 16:00-20:00 ☒

Fri, 3 Sep 17:00-19:30 ☒

✓ Enforce periods.

Period name: Student name

Period (min): 30

https://localhost:5001/BookingSession/NzVGYHbJsSG6nZy5zftGe

List of long events:
Ramadan
2/4/2022 - 1/5/2022

Figure 5 Booking slots sketch.

Within the given example, the students open the shareable link. The booking calendar displays the appropriate booking session based on the provided URL. Clicking on a booking slot opens a popup. The popup provides a field labeled with the aforementioned period name. On submit, the chosen booking slot is written into the calendar owner’s Google Calendar as an event. The event name is the input given in the booking popup. The selected booking slot is then deleted from the booking calendar. Within the given example, the teacher’s Google Calendar is populated with events, labelled with the student names.

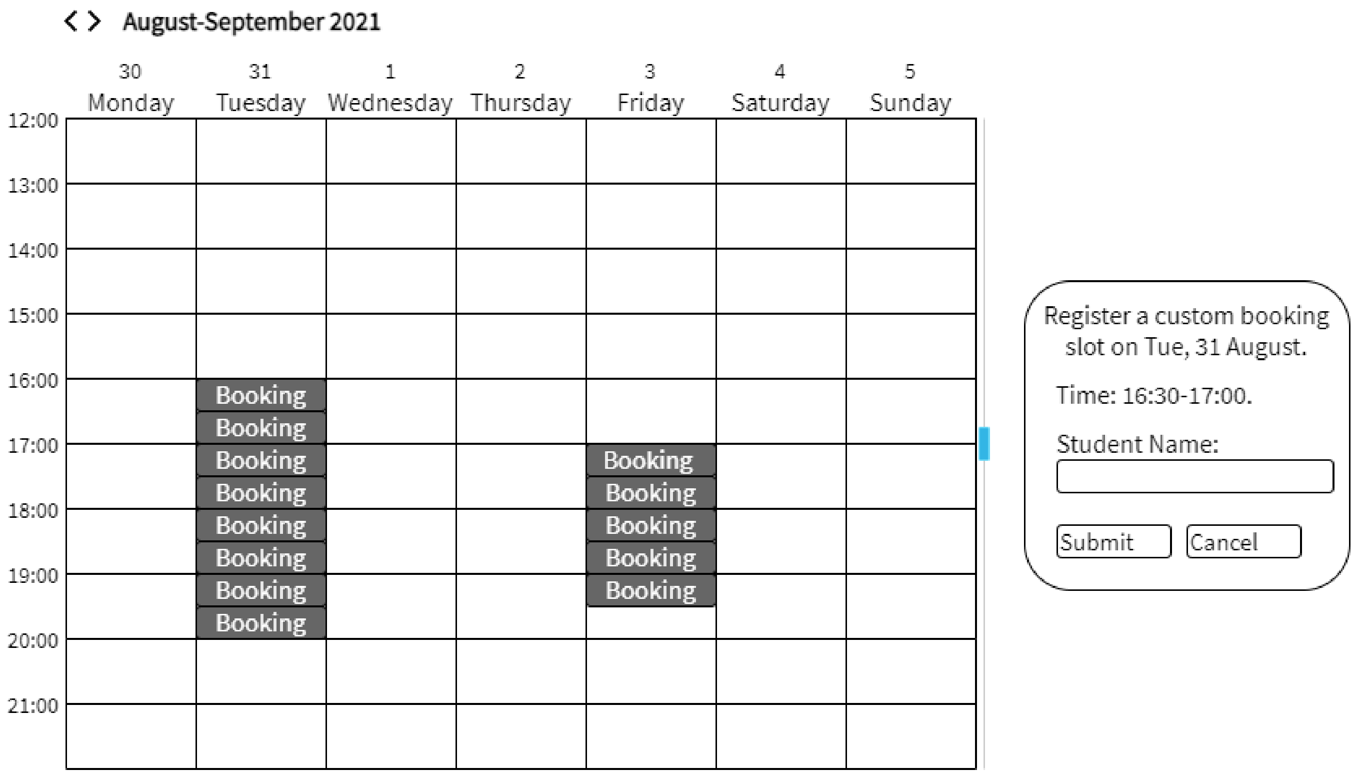


Figure 6 Booking calendar sketch.

4 System implementation

4.1 User authentication and authorization

The OAuth 2.0 protocol is used to authorize third-party clients accessing the resources of some external server. In this case, the client is the calendar manager web application. This authorization is done on behalf of the service users, where the third-party client obtains user resources from the external server (Hardt, 2012). The calendar manager web application will be authorized with the Google Authorization server, on behalf of Google users.

First, the calendar manager web application is registered with Google through Google API Console, which is Google's API management service. Upon registration, the API Console generates a Client ID string and a Client Secret string, which are interpreted as a unique application identification. These credentials are used to identify the client when making requests and to confirm that the client has been registered with Google.

Next, the client makes a token request to Google Authorization Server. This token request states the requested APIs and the scope of these APIs. The calendar manager web application is requesting scopes from the Google Calendar API. The request token will also prompt user authentication. The users login to their Google Account and grant permissions presented from the token request scope. The full user authentication is handled by the Authorization Server. The Authorization Server then sends back an authorization code, only containing the scopes that the user has permitted.

The authorization code is then sent to Google API Exchange Server. Upon receiving the authorization code, the Exchange Server sends back an access token and a refresh toke. The access token has a limited lifetime. The refresh token is used to generate new valid access tokens. The access token is then used to access the specific Google API (Azad, 2021).

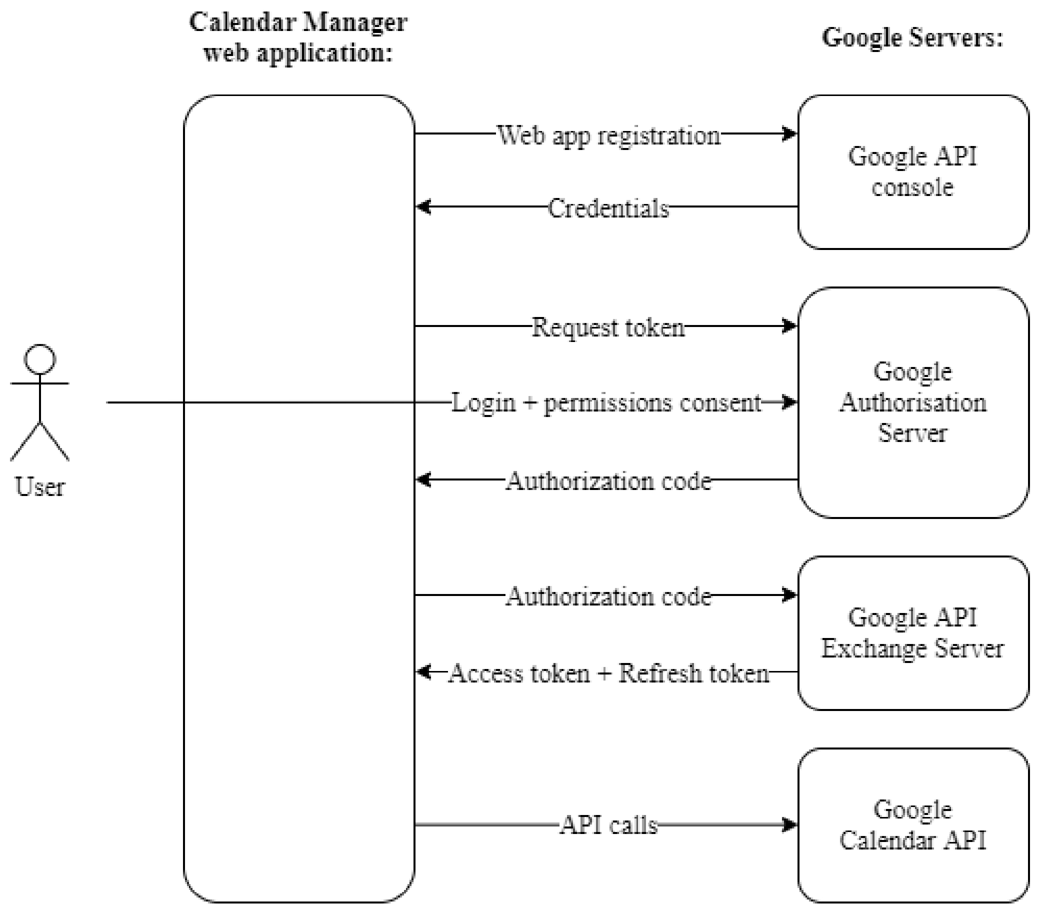


Figure 7 OAuth 2.0 authorization protocol.

The acquired Calendar Manager's credentials, composed of a Client ID and a Client Secret must be protected. A disclosure of these credentials would allow an attacker to make requests to Google on behalf of Calendar Manger by impersonating the client web server. To avoid this, all transmissions between the client web server and Google servers must use Transport Layer Security (TLS) cryptographic protocol (Lodderstedt et al., 2013).

The TLS protocol is built on top of Hypertext Transfer Protocol (HTTP), encrypting all communication between servers over the Internet. HTTP over TLS is equivalent to Hypertext Transfer Protocol Secure (HTTPS). The ASP.NET Web Framework allows the addition of its preconstructed middleware to the web application pipeline. The chosen middleware is then executed on every request made by the web application. HTTPS Redirection middleware and HTTP Strict Transport Security (HSTS) middleware have been explicitly added into the pipeline of Calendar Manager. HTTPS Redirection middleware redirects all HTTP requests to a port reserved for HTTPS if one exists, otherwise the request fails. This ensures that the client credentials sent to Google servers are transmitted over HTTPS. Additionally, all other outgoing requests from the Calendar Manager are redirected to HTTPS, regardless of the specified protocol and implementation of the HTTP client making the request. HSTS middleware adds a ”Strict-Transport-Security” field to the response header of every request. Upon receiving the header, the browser forces all transmissions to use HTTPS instead of HTTP. Within Calendar Manager, this policy forces all requests made by the browser to use HTTPS for transmissions, regardless of the axios HTTP client implementation. As a result of adding the two middleware, all requests made from the Calendar Manager back-end and fron-end are automatically redirect to a secure HTTPS port (Anderson, 2019).

The client credentials are required every time the client makes an API call on behalf of a user. Therefore, these credentials must be stored. The naive option is to store these in a configuration file within the Calendar Manager project. This method is not secure because this software is open source, the source code is publicly available and the credentials would be exposed to potential attackers. The next option is to store these in a secure database. This would result in a table with two columns and only a single entry, because the same application credentials are used for all users of the Calendar Manager application. This is a valid and a secure solution, however constructing an entire database for the sake of storing one entry, which can even be reduced to one string, is inefficient. The chosen solution is to use a secret configuration file located outside of the project solution, secured on the development machine. The required credentials are then pointed to this file. This keeps the application credentials out of source control. ASP.NET uses a secret manager for the secret configuration file, which has been utilized, that maps the Windows file path to its corresponding Linux file path and vice versa. This ensures that when transferring from development environment on one operating system to production environment on a different operating system, there is no need to adjust file pathing for the secret configuration file in the source code. The secret configuration file is simply, manually transferred over to the correct file path and the secret manager handles path mapping automatically (Anderson et al., 2020).

To request an authorization code from Google Authorization Server, a ‘state’ parameter is set. This is used to prevent Cross-Site Request Forgery attacks (CSRF). The state token is given a random value, which is hard to guess. Without this anti-forgery token, an attacker is able to POST a response to Calendar Manager, acting as the Authorization server, given that they can guess or know the callback URL (Auger, 2010). The callback URL is known because this project code is publicly available. In the current version of the application, the callback action checks the returned authorization code and then POSTs a request to exchange the authorization code for an access token. Google have designed this extra exchange verification step to protect application users from CSRF attacks, in case of poor application implementation. Therefore, if an attacker forges the response with their own authorization code, the exchange request will be invalid because of the invalid authorization code. The application throws a ‘Bad Request’ exception, and the user is protected. The state token is still essential because this application is open-source. If the application is extended in a way where the callback action calls some other critical function, the state token would prevent an unauthorized attacker from triggering and performing this critical function.

The recommended value for an anti-forgery token is a 30 character long, randomly generated string (Azad, 2020). A recommended scheme to generate the 30 character long anti-forgery token is not defined. An arbitrary scheme [a-z, A-Z, 0-9] has been chosen. This specific anti-forgery token implementation generates the recommended 30 characters long string from the 62 different options, equating to 62^{30} permutations. To ensure the implemented state token value is ‘hard to guess’, a comparison with Universally unique identifier (UUID) system is made. UUID is the current standard for labelling groups of data, within computer systems, with a unique ID. A UUID is composed of 32 randomly generated characters using a scheme of hexadecimal characters. This results in 16^{32} permutations. The space of all UUIDs is so large that generating two identical UUIDs is practically impossible (Wikepdia-Contributors, 2021c). The anti-forgery token implementation has a higher entropy than UUIDs. A possible state token value is hard to guess.

Upon successful user authentication and authorization, user access and refresh tokens are retrieved from the Google API Exchange Server. Each user has a corresponding access and a refresh token in a one-to-one-to-one relationship. To be able to handle multiple users accessing Calendar Manager at the same time, in parallel, a database to store these tokens is created. The access token is instantly used to request user’s calendar events. However it must still be stored in a database to allow page reloading. On page reload, a new calendar instance is instantiated, losing the previously assigned access token local variable. Therefore, the access token is retrieved from a database. The access token has a limited lifetime, one hour, after which it is no longer valid, resulting in an unauthorized access exception. In such case, the exception is caught and a refresh token is used to request a new valid access token, which is updated into the database and is used to retrieve user events. The refresh token is also used within the booking calendar. When a new booking is submitted, the refresh token is used to obtain a new access token, which is then used to insert a new event into the calendar owner’s Google calendar.

4.2 API

For Calendar Manager to be able to make requests to a Google API, an API key is needed. The API key is a unique identifier for a given web application. Its unique value is generated based on the specified application and API restrictions. The application is Calendar Manager and the API restriction is Google Calendar API. Similar to client credentials, the API key is manually generated with the Google API console. The API key is then used in combination with a specific user’s access token to make API calls to

Google Calendar API. The API key is secured and stored in secret configuration because its disclosure would allow an attacker to make API calls on behalf of Calendar Manager.

The Calendar manager application requires two Calendar API scopes for full functionality. Permission to view events on all calendars (calendar.events.readonly), and view and edit events on all calendars (calendar.events). These scopes are permitted or declined by the user during user authentication process.

The recommended approach is to use incremental authorization. In the context of Calendar Manager, during the initial login, the user is asked for ‘calendar.events.readonly’ scope, which allows the web application to dynamically populate the user’s calendar with Google events. The user then chooses their booking slots and booking options. Upon generating the booking link, the user is then asked for ‘calendar.events’ scope. This will allow any booked slots to be written into the user’s Google calendar. Incremental authorization requests scopes in context, allowing the user to better understand why the given scopes are required and as a result, the user is more likely to accept the requested permissions. It is most effective when asking the user for permissions across multiple APIs (Daugherty, 2021a). The downside of incremental authorization is the overhead on the user in the amount of tasks that they need to carry out. In the context of Calendar Manager, the main issue arises with the natural human memory phenomenon of closure. The human short term memory is finite and limiting, and as a result, doing tasks in series is easier and preferable than multitasking. Closure is a human desire to complete tasks in order to clear the short term memory stack. During the Calendar Manager booking process, the user executes a task of choosing available booking slots, followed by choosing the booking options, finalised by submitting and generating a link for their specific booking session. Introducing the permission request for ‘calendar.events’ just before submission would introduce another task of trying to understand the context of the scope and deciding whether to permit it or not. This new task is placed right before the closure of the first task, which is extremely inefficient for the human short term memory and processing (Dix et al., 2005). In addition, asking the user for full access to their Google calendar during the initial sign in does fit the context of ‘signing into a calendar manager application with Google calendar integration’.

During the initial user log in, there is an option to request ‘calendar.events.readonly’ and ‘calendar.events’, where the user decides on only allowing ‘calendar.events.readonly’, or both. The Calendar Manager functionality is then dependent on the allowed scopes. If only ‘calendar.events.readonly’ is chosen, then the calendar is dynamically populated with Google events, but the booking system is disabled. Without the booking system, Calendar Manager is useless as it provides no features that Google Calendar cannot do. But, Calendar Manager is specifically designed to be extendable with further features. However, there were no possible calendar features discovered during the research stage that solely require ‘calendar.events.readonly’. As a result, ‘calendar.events.readonly’ is deemed redundant and is removed from the requested scopes.

When a user of Calendar Manager clicks on ‘Log in’, they authorise with Google and are then redirected to their dynamically generated calendar web page. The standard method of implementing this back-end is to use a single Calendar controller. However, the functionality can be divided into several distinct components. Therefore, a decision has been made to split these components into several controllers to enforce separation of concerns design principle. An Authorization controller handles user authentication and authorization with Google servers. It is vital that this controller is completely isolated from any view because the authorization code is passed back to the controller through the URL. Serving a view using this controller's endpoint would result in the response parameters being exposed to the browser. Security vulnerabilities could then arise if Calendar Manager was extended with third-party plugins or scripts which would have access to the authorization code (Daugherty, 2021b). Upon successful authentication and authorization, the Authorization controller obtains an access token and a refresh token. Both tokens are stored in a database for later use and the access token is also passed to the API Controller. The API controller uses the retrieved access token with the API key to request user's calendar events. The received JSON response is then parsed into an events model and the model is passed to the Calendar Controller. The Calendar Controller returns a Calendar View with the events model passed in as the parameter. The Calendar View then serves the user a dynamically generated calendar HTML page using the passed events model.

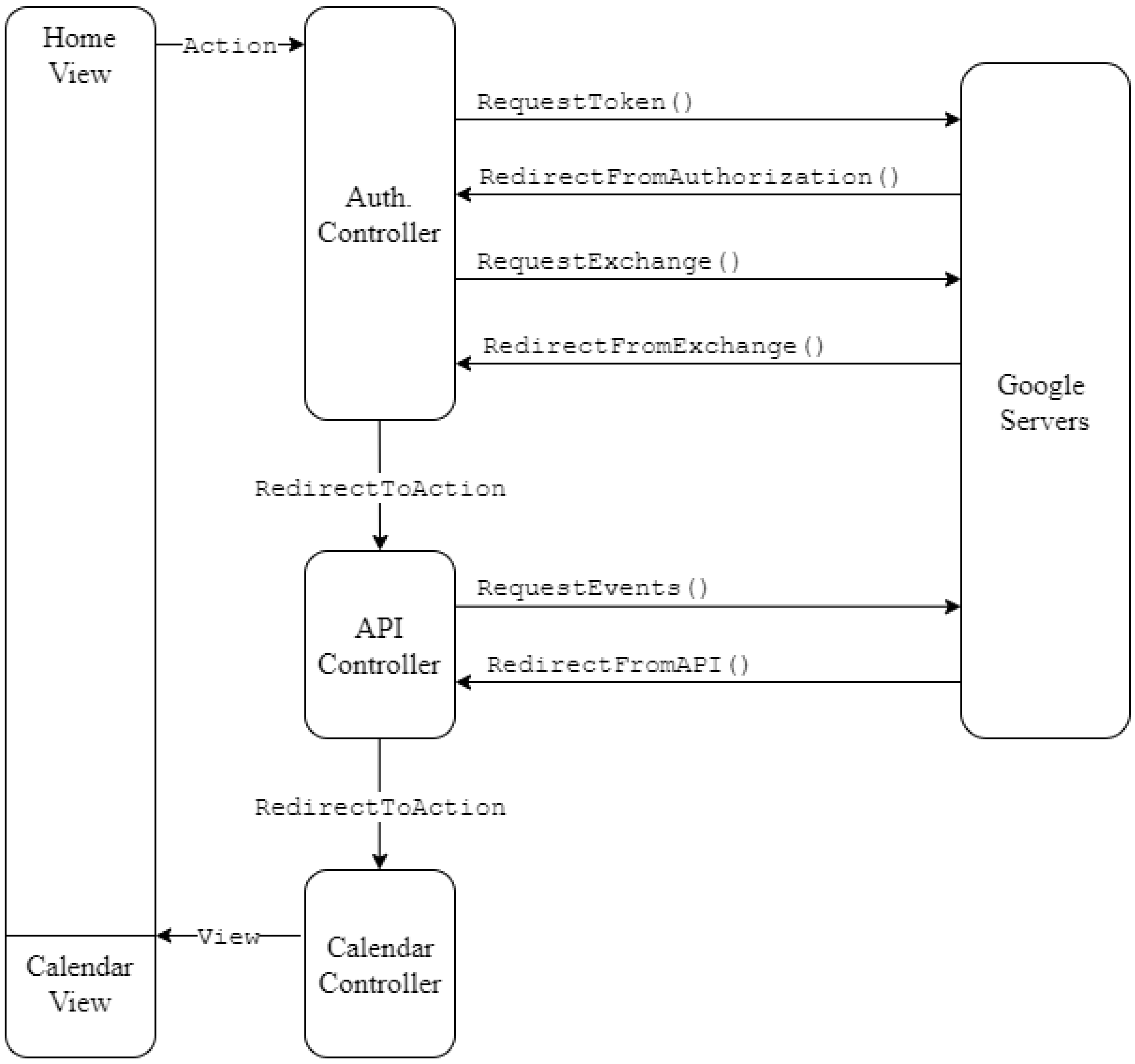


Figure 8 Desired program flow.

Unfortunately, it was not possible to redirect to another controller or another action from the API controller action receiving the JSON events response. The system automatically returned the Home View. Upon further investigation via the .NET debugger, it was found that any action receiving a POST response must return a view. Since no view is returned, the previously used view, the Home View, is automatically returned. This functionality is baked into ASP.NET. To overcome this hurdle, all API Controller functionality is placed in an action of the Calendar Controller. This single action makes a request to the API, retrieves the JSON response, parses it into an events model, which is then passed into the Calendar View, which renders the dynamic calendar HTML page.

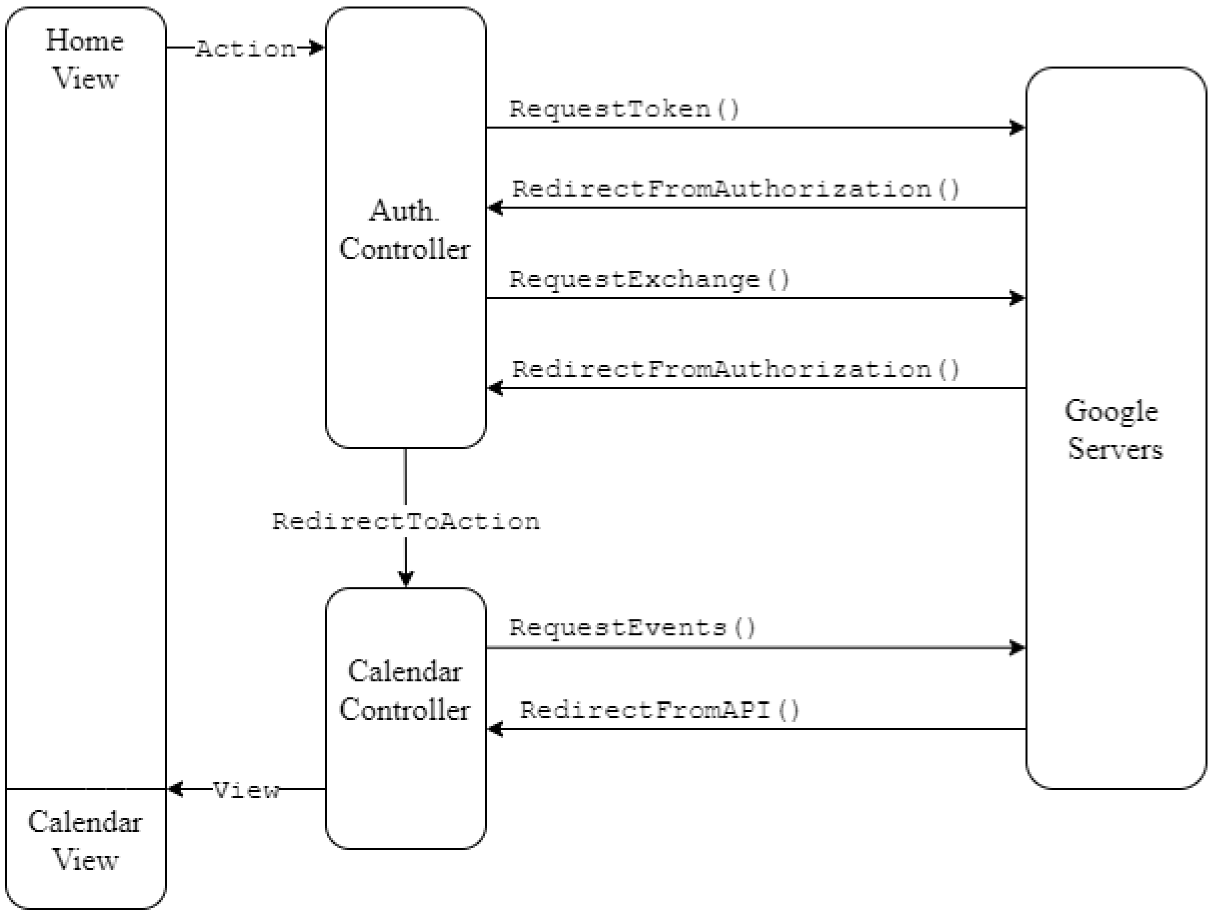


Figure 9 Actual program flow.

4.3 Database

An underlying relational SQL Server database is used to control the state of the application and provide secure storage for user secrets. The connection string to this database is stored in the secret configuration file, along with client secrets and the API key.

Upon user authentication and authorization, the user session is stored in a table as a combination of their email, access token and refresh token. These control the application state. There are no passwords, permissions or other extra user secrets utilized. Upon user login, the access token is used to populate the calendar. Upon user logout, the website simply redirects to the home page. The permissions are controlled by Google API, which throws an error if an invalid access token is supplied.

The database also stores booked events. A set of booked events is identified via a dynamically generated string, the dynamic URL. This mapping is used to transfer the user-defined booking events to the booking calendar. The most popular link-shortening service bit.ly, uses seven characters from 62 different options (A-Z, a-z, 0-9). This equates to three trillion permutations and is considered sufficient. Bit.ly manages all the URLs to ensure that there are no collisions (Gould et al., 2016). To avoid this overhead in complexity, Calendar Manager generates the dynamic string from a pool large enough, such that no collisions are possible. The chosen scheme is [a-z, A-Z, 0-9], because these are the most commonly used characters on keyboard inputs. As previously established, the UUID space is large enough to guarantee that no collisions are possible. To match this space, the dynamic URL string is given 21 characters.

The database is constructed using entity framework. Entity framework is an object mapping framework which maps SQL Server tables to classes within the Calendar Manager solution. This allows the use of C# syntax to define tables, columns, and column properties. The database created is a relational database, therefore foreign keys have been used. These enforce the referential integrity by preventing the insertion of illegal records that would destroy the relationships between tables. Additionally, an indexed property is appended to the foreign keys. Indexed keys are optimized for querying to enable faster lookups (Smith, 2021). This is a beneficial adjustment because all the lookups within Calendar Manager are done via primary keys, which are automatically indexed, and foreign keys.

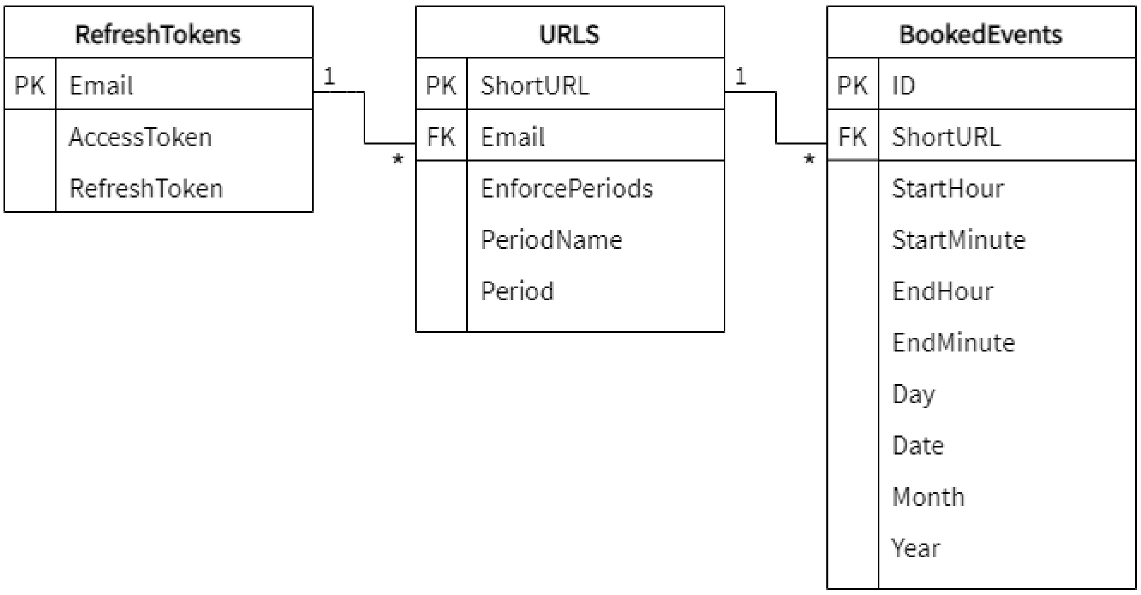


Figure 10 Underlying database.

4.4 Routing

Upon any request to the Calendar Manager web application, the request is parsed through the routing middleware. This middleware parses the request into an application endpoint. The parsing is based on a routing scheme, specified in a routing table. The default scheme parses the first part of the path into a controller, and the second part of the path into an action. For example, the path '/Calendar/About' is forwarded to the 'About' action in the 'Calendar' controller. If no action is specified, then the 'Index' action of the specified controller is used. For example, both '.../Calendar/Index/' and '.../Calendar/' are routed to the same endpoint, the 'Index' action of the 'Calendar' controller. The latter scheme is used throughout Calendar Manager because all web pages are main pages; there are no sub-pages.

The desired dynamically generated booking session URL is of the form '.../BookingSession/dynamicURL'. Under the current routing table pattern, this request would look for an action which does not exist in the BookingSession controller. To enable this specific pattern, a new pattern of the form '.../BookingSession/dynamicURL' is added to the routing table, where 'dynamicURL' is specified to be an id passed as a parameter. The default endpoint for this pattern is also specified, which navigates to the 'Index' action of the 'BookingSession' where the 'dynamicURL' is passed in as a parameter (Smith, 2021).

Route name	Pattern	Default endpoint
default	”{controller=Home}/{action=Index}”	Not specified
bookingPage	”BookingSession/{id?}”	new {controller=”Booking-Session”, action=”Index”}

Table 1 Routing table within the routing middleware.

5 System architecture

5.1 Dependency injection

ASPN.NET provides support for its in-built dependency injection. Calendar Manager utilizes this dependency injection for all its supporting business logic throughout the project solution.

A dependency, for example a class within the class library, is registered within a predefined dependency injection container as part of the application setup. This class is now available as a dependency throughout the project solution. This dependency can be used in some other class, where the object is passed through a constructor. Within the constructor, it is then set to a class member variable, as a private field. This private field is now available to the entire class.

This approach is preferable over hard-coding class instantiations throughout various methods, as it avoids code duplication. Code duplication adds extra technical debt and deteriorates maintainability. Dependency injection also aids with unit testing where the user does not need to be concerned with instantiating classes throughout test methods, as long as they declare the required dependencies through the test class constructor (Smith, 2021).

5.2 Revised project structure

At the start of this project, it was decided that a class library would be used for all reusable business logic. A class library was chosen over a 'Services' folder which would have been placed within the Calendar Manager project. The 'Services' folder would hold all business logic locally. The events retrieved from Google Calendar API are returned as an array of JSON objects. A functionality for the application is needed, which deserializes the wanted information into a list of 'Event' models. This list is then passed to the browser which populates the calendar. This functionality cannot be placed in the Calendar controller because the controllers should only handle browser requests and routing. Logically, this functionality should be stored in the class library with all the other supporting business logic. Creating a class in the class library which handles this functionality requires access to 'Event' model. However, the 'Event' model is in the 'Calendar Manger' project. A dependency on the 'Calendar Manager' is needed to reference the 'Event' model. Now, the 'Calendar Manager' project is dependent on the class library and the class library is dependent on the 'Calendar Manager' project, creating a circular dependency. As a result, the project cannot be compiled without compiling the library first, and the library cannot be compiled without compiling the project first (Wikepdia-Contributors, 2021a).

The first solution to this problem is to create another class library, within the project solution, which would contain this newly needed class, breaking the circular dependency. This fixes the issue in the short term. When extending this application, new classes can be made which may depend on all the previously created projects, requiring more and more class libraries to break the circular dependency. Numerous class libraries for the sake of breaking circular dependency clutters the project solution. Additionally, it would become difficult to manage all the projects within the project solution. The second, more appropriate option is to create the aforementioned 'Services' folder within the main project. The 'Services' folder contains business logic dependent on other classes within the project, such as models or the database interface, and the class library contains all independent business logic. The drawback now, is that there are two sources of business logic, deteriorating from the compact project structure. Alternatively, the class library business logic can be moved to the 'Services' folder, adding more structure to the project. However, the decision has been made to keep both sources of business logic because the benefits of having a class library outweigh this minor drawback, given the project aim.

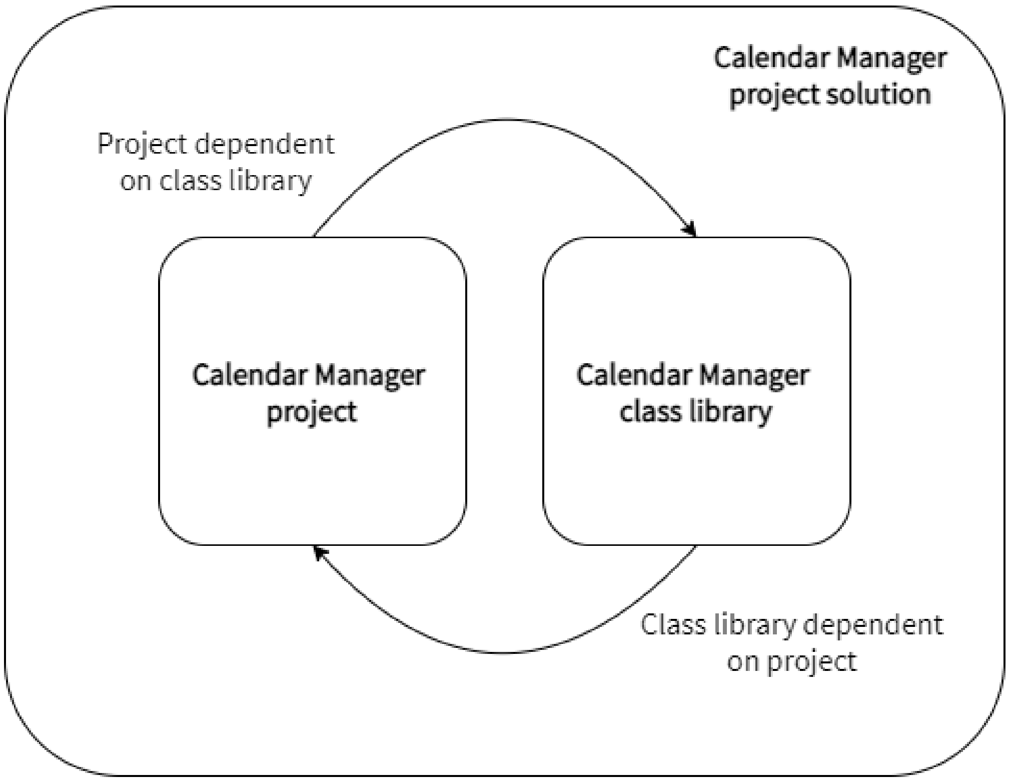

Calendar Manager project solution
Project dependent on class library
Calendar Manager project
Calendar Manager class library
Class library dependent on project

Figure 11 Circular dependency.

5.3 Browser-side HTML design

The purpose of HTML code is to define the structure of a web page. All Calendar Manager pages utilize the standard wrapper HTML elements that form a valid HTML page. The content within the <body> element should contain HTML elements that describe the semantic meaning of each component within the web page. The HTML language was not designed for this sole purpose. It was designed to include some basic styling via non-semantic styling elements. Many current websites depend on these non-semantic elements and therefore they cannot be dropped from the HTML language. Throughout the evolution of HTML however, an effort has been made to adjust the non-semantic elements' meaning. For example, HTML 4 defines <b> as ”bold”, whereas HTML 5 defines <b> as ”bring attention to element”. There are still some semantic elements which should be avoided, such as <span>, therefore it is important to distinguish between semantic and non-semantic elements.

All HTML pages of Calendar Manager strictly contain semantic elements only. The only exception is the <div> element because styling pages without it, is unrealistic and impossible within the confines of current state of web technologies and web architecture. The <div> element is only used for wrapping and styling when no other semantic element is appropriate.

The majority of Calendar Manager HTML code covers the default calendar and the booking calendar. Both use extremely similar HTML markup. The <main> element represents the central functionality of the calendar page. For this reason, the page navigation has been placed outside of this element. The <section> element is a generic semantic element which depicts a singular, unrelated section within the web page. It is mostly used when no other semantic element is appropriate and it helps alleviate the <div> element usage. The <aside> element is used to represent supplementary content related to the main content. Calendar Manager utilizes the <aside> element for a side panel of the main calendar. The <aside> element is typically used alongside the <main> element; however, Calendar Manager needs the <aside> content to be within the <main> scope because of the Vue.js relationship with HTML elements, covered in the next subsection. The final semantic element used is <article> which represents an independent component, which can be reusable and attributable. The calendar wrapper containing the calendar grid, as well as the booking popup, are defined as an <article> within the calendar page, because they are being reused in the booking session calendar page (MDN-Contributors, 2021).

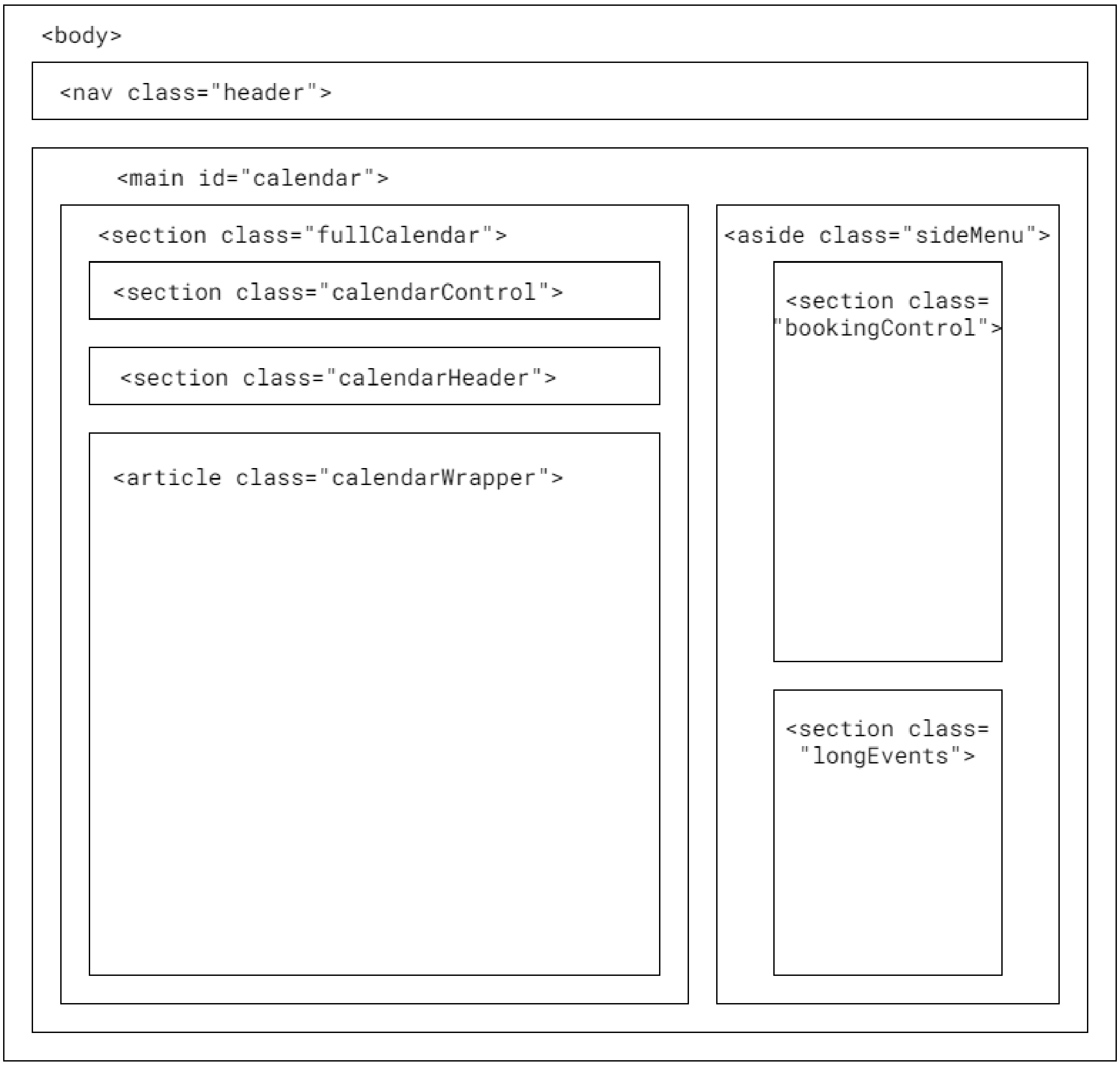


Figure 12 HTML semantic structure.

5.4 Browser-side Vue.js design

The Vue.js framework implements and adheres to a Model-View-ViewModel (MVVM) pattern. MVVM enforces the separation of business logic and user interfaces. A View represents the interface, in this case it is the HTML Document Object Model (DOM). The Model holds business data. The ViewModel provides business logic and converts Model data objects into View data objects. In the case of Calendar Manager, the ViewModel is an instance of Vue and the Model is data contained within this instance (Wikepdia-Contributors, 2021b).

An instance of a Vue object passes a JavaScript object through its constructor. The properties specified within this JavaScript object define the Vue object. The first property is ‘el’, which specifies a HTML DOM element that this Vue instance targets.

This DOM element and all of its child elements are now treated as the View. Within Calendar Manger, the Vue instance targets the <main> element as its 'id' value matches the 'el' value. The <aside> element was placed within the <main> element as its child, to allow its manipulation through this specific Vue instance. The 'data' property stores all variables belonging to this specific Vue instance. This single property is perceived as the Model. The next property 'computed', is a ViewModel. The 'computed' property contains functions which manipulate data from the 'data' property and pass the results to the View. Within Calendar Manager, 'computed' property handles minor computations, such as getting the number of days in February, depending on the year, and minor validation, such as disabling the previous button when the year 2009 is reached. The final property is 'methods'. This is also a part of the ViewModel and is reserved for larger functions (You et al., 2021).

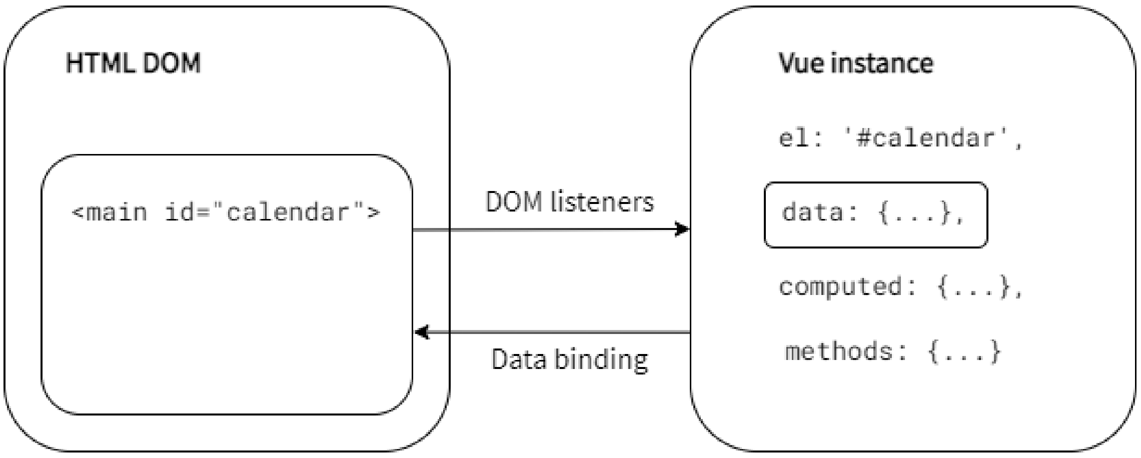


Figure 13 Vue.js MVVM design.

The Vue object instantiation goes through a specific life-cycle, comparable to how a compiler goes through several stages. Vue provides access to several methods which are executed throughout this life-cycle. Upon the creation of a new Vue object, the data of the Vue instance is read. Once the data is processed, Vue provides access to a 'created' function. This is reserved for preprocessing the instance data. The instance data contains user events which were passed to the browser from Calendar Manager server-side. Calendar Manager uses the 'created' method to ascertain the current date from which other helper data is computed. It also filters out long events from the events list. The next step in the life-cycle is compiling the HTML DOM templates. A template is a string that contains HTML code, which is then injected into the HTML DOM, the View of Vue. Vue now mounts the DOM, where it establishes a link between the ViewModel and the View. At this stage, the DOM elements are available to be manipulated through the ViewModel. A 'mounted' function is now available. Calendar Manager utilizes this 'mounted' function to populate user events and booking events, and enforce validation (You et al., 2021).

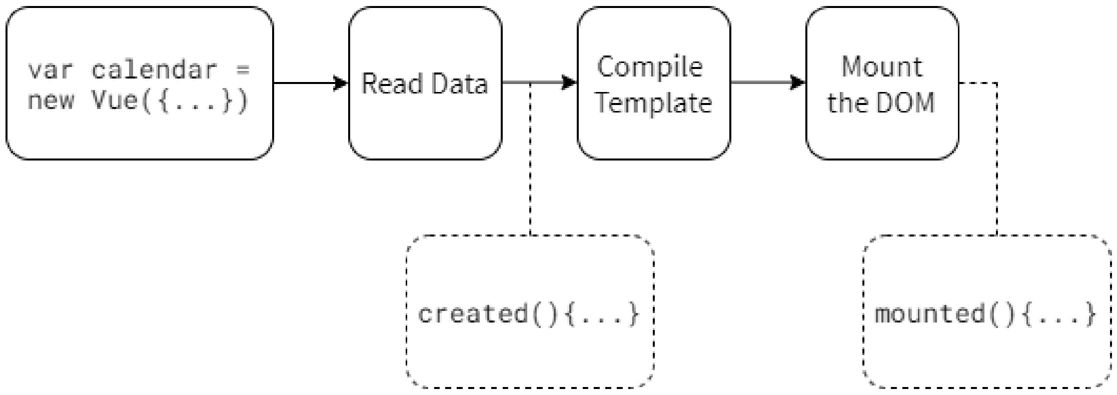


Figure 14 Vue.js life-cycle (simplified).

5.5 Documentation

Documentation is critical in a project such as Calendar Manager, who's key properties are maintainability and extendibility. Appropriate documentation helps with understanding and reusing the Calendar Manager skeleton code.

C# provides XML documentation which binds to C# constructs such as classes or methods. The XML documentation provides several XML tags. The <summary> tag has been utilized to provide a description to every class within Calendar Manager. Unlike a multi-line comment, the summary is baked into the signature of the class as a description. When an external user wants to know the description of this class, they do not need to locate the original source code for a multi-line comment description, they just need to inspect the signature of the class (Wagner, 2021b). To document methods, a preprocessor directive '#region' has been used. This directive wraps around a user-defined region and provides a user-defined label to it (Wagner, 2021a).

HTML comments are HTML markup with tag lines, which can add additional meaning to semantic elements describing the HTML DOM. However, they can also cause extra clutter within the HTML. Cluttering HTML can introduce extra difficulty when perceiving the HTML DOM structure. The HTML elements have been specifically designed, such that the semantic HTML element in combination with an appropriate named class or id gives enough information about each HTML element in question. Therefore, HTML comments are omitted. Any extra detail required to label HTML elements is supplied in an external documentation, such as '5.3 Browser-side HTML design' section of this document.

Vue.js does not supply any additional constructs for documentation or comments. The only constructs available are the vanilla JavaScript single line comments and multi-line comments. The extension of Calendar Manager will most likely involve the addition of new features. These features will be appended onto the current front-end implementation of Calendar Manager. It is therefore imperative that all Vue instances are well documented. In addition to comments labeling methods and describing code blocks, a documentation should be supplied describing the high-level structure of

Vue instances and how the different components work with each other, such as '5.4 Browser-side Vue.js design' section of this document.

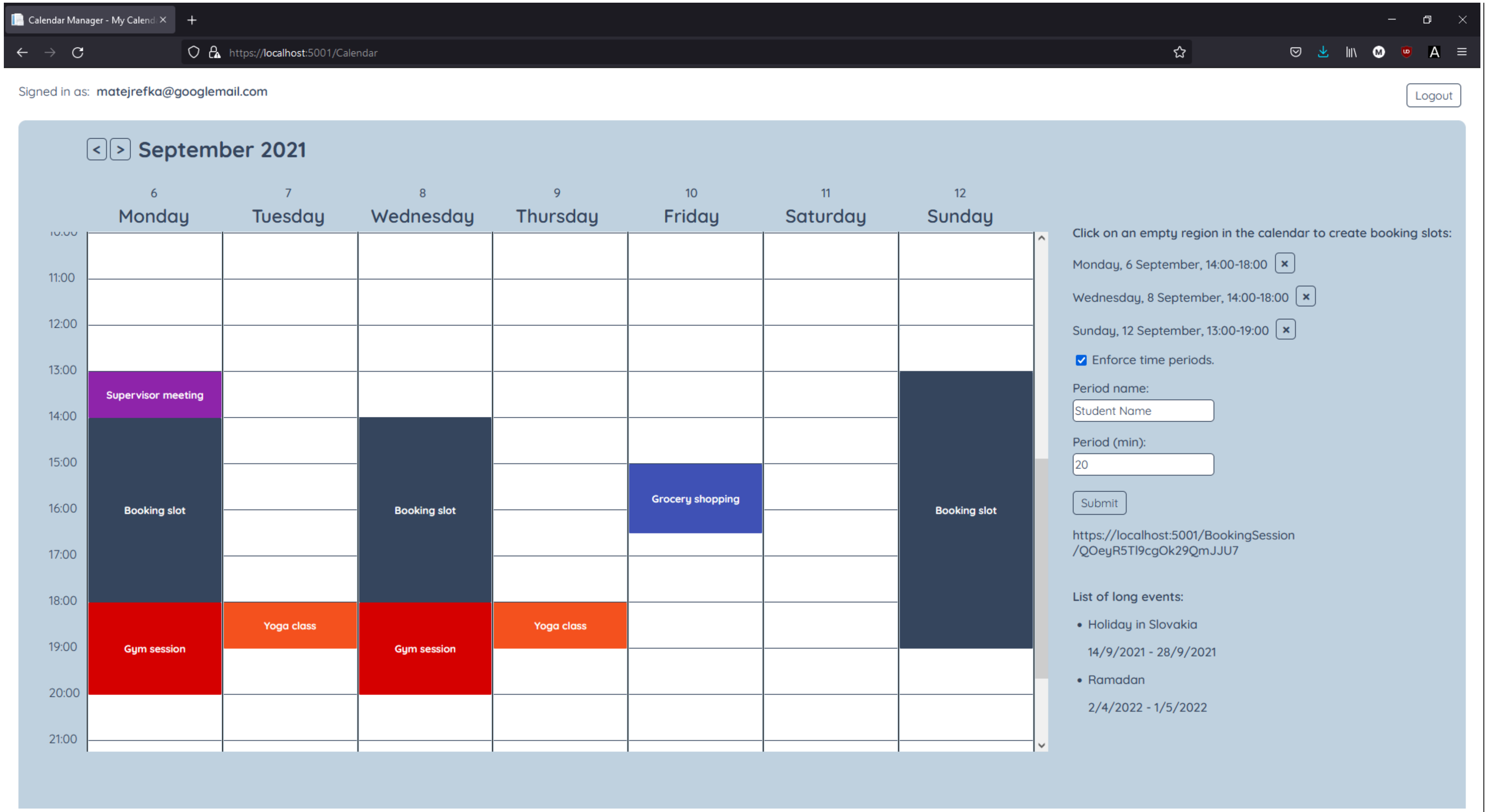
6 Results

This project can be viewed in two parts. The implementation of the skeleton Calendar Manager web application and appropriate use of software engineering design, making the skeleton application extendable and scalable.

6.1 User interfaces

The deliverable component of this project is the Calendar Manager web application. This component can be viewed via its two main interfaces. The evaluation of these results is supplied in the 'Analysis' section via usability testing.

Contents Tables Figures References



Signed in as: matejrefka@googlemail.com

Click on an empty region in the calendar to create booking slots:

Monday, 6 September, 14:00-18:00

Wednesday, 8 September, 14:00-18:00

Sunday, 12 September, 13:00-19:00

[✓] Enforce time periods.

Period name: Student Name

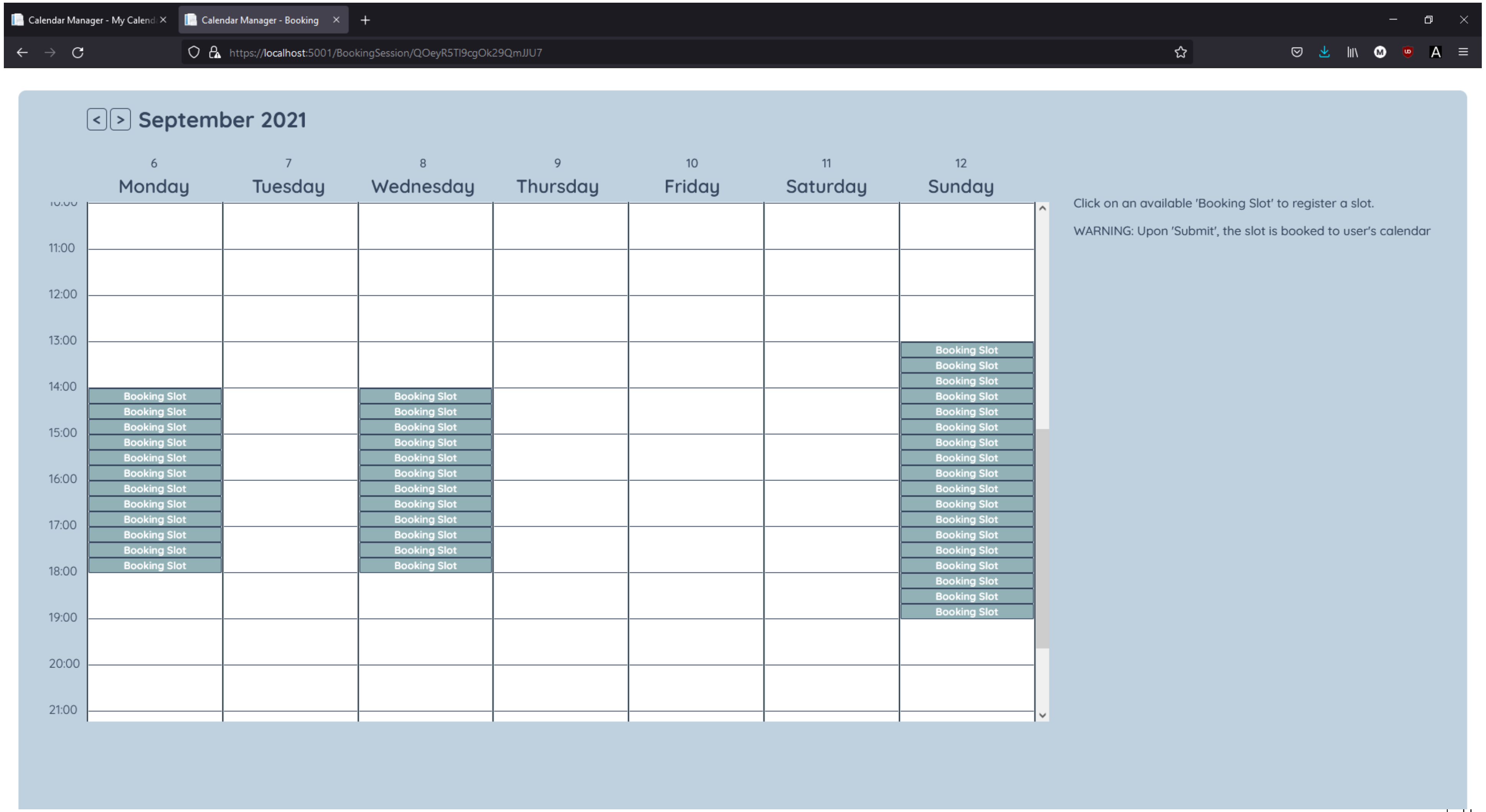
Period (min): 20

https://localhost:5001/BookingSession/QOeyR5TI9cgOk29QmJJU7

List of long events:

- Holiday in Slovakia
  14/9/2021 - 28/9/2021
- Ramadan
  2/4/2022 - 1/5/2022

Figure 15 Calendar UI.



Click on an available 'Booking Slot' to register a slot.

WARNING: Upon 'Submit', the slot is booked to user's calendar

Figure 16 Booking UI.

6.2 Software architecture

It can be difficult to assess whether appropriate software architecture has been used within a software engineering project (Smith, 2021). To capture these results, the Calendar Manager architecture is compared against a set of standard, predefined software design principles. Principles which are not fully adhered to are then further evaluated in the 'Analysis' section.

Design principle	Adherence	Use within Calendar Manager
Separation of concerns	Yes	MVC Middleware, project structure, folder structure
Encapsulation	Partial	Class library
Dependency inversion	Yes	Inbuilt dependency injection
Explicit dependencies	Yes	All objects instantiated through constructor dependency injection
Single responsibility	Partial	Encapsulation of business logic
Don't Repeat Yourself (DRY)	Partial	Encapsulation of business logic

Table 2 Predefined software design principles.

6.3 Accessibility

The accessibility of Calendar Manager is determined by referencing WCAG 2.1 (Kirkpatrick et al., 2018). Each component of WCAG 2.1 is evaluated to determine whether or not Calendar Manager adheres to this component guideline. Any intermediate steps and calculations required, to derive at these conclusions, are displayed below. The full test plan is supplied in Appendix B.

Visual component	Contrast
Text inside the calendar, background	6.2
Text outside the calendar, background	9.71
Booking slot text, background	9.71
Disabled input field text, background	6.3
Non-text (Any two-color combination of the color scheme)	over 3

Table 3 Contrast ratio of components within Calendar Manager.

7 Analysis

7.1 Encapsulation principle evaluation

Encapsulation involves isolating parts of the project from each other. The use of a class library ensures that business logic is completely separated from the main project. A change of implementation within any class of the class library will not break any functionality within the main project, given that no external contracts are violated. External contracts are class and method signatures. These include class and method names, constructors, method parameters and their types, and method return types. Enforcing these external contracts can be achieved by introducing interfaces for all business logic.

To ensure full encapsulation within Calendar Manager, each class of the class library and the Services folder should be accompanied by a supplementary interface. An extension or a re-implementation of business logic classes may result in altering the class signature. Under the current implementation of Calendar Manger, this would violate the external contracts within the main project, enabling the propagation of errors throughout the entire project. However, an interface of such class, ensures that an error is raised, allowing the user to alter the class implementation to adhere to its corresponding interface, preserving the functionality of components dependant on this class.

In addition to the use of interfaces, business logic classes should limit outside access to their state. Their state should only be altered through setter methods (Smith, 2021). The Calendar Manager application achieves this via the use of private fields with setter methods and C# Properties constructs.

7.2 Single responsibility principle evaluation

The single responsibility principle is similar to the separation of concerns principle. Separation of concerns is mainly associated with high-level system structure, whereas the single responsibility principle covers both the high-level and low-level system components. The high-level includes project structure, folder structure, and separation of user interfaces, business logic, data access and system testing. The low-level components include classes and methods. Each class should only have a single responsibility. This responsibility should be implemented via a limited number of methods. When extending some extra functionality, it is preferable to add it to a new class if possible, rather than an already implemented class. Adding this extra functionality to a new class is safer because re-implementing or adding extra methods to a functioning class may cause dependency issues with components that depend on this already established class (Smith, 2021).

The Calendar Manger application includes a case where the single responsibility principle is violated. This occurs within the ‘Event Deserializer’ class which consists of four methods. The first method parses a JObject retrieved from Google API into a list of Event models. The second method retrieves user email string from the JObject retrieved from Google API. The last two methods parse some models to some other models. These last two methods were added to this class as they are both technically deserializing an event. The Event Deserializer class has been presumed to be, and used as a general parses for all types of objects. The Event Deserializer class has multiple responsibilities.

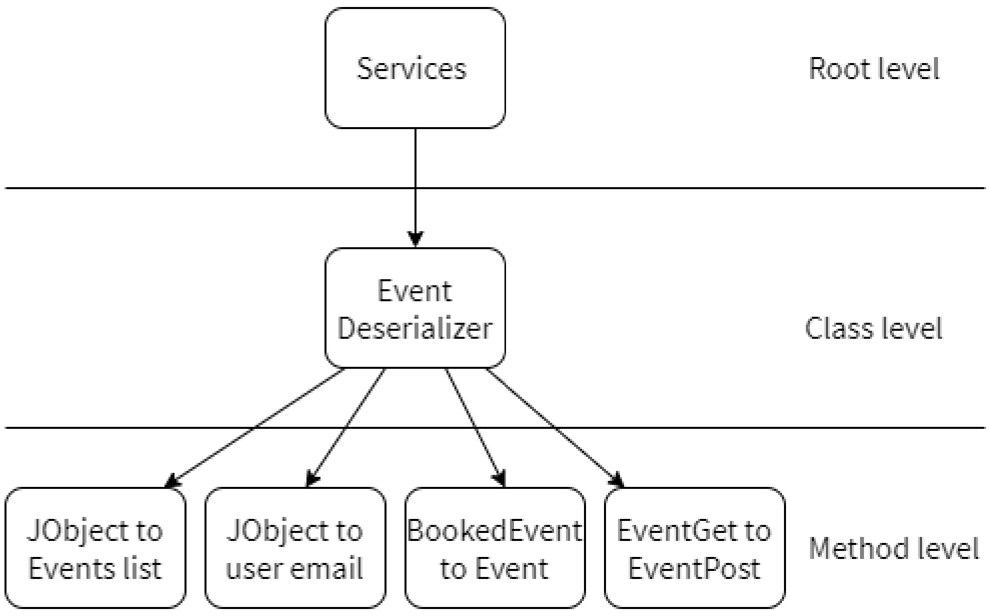


Figure 17 Single responsibility principle violation.

To enforce the single responsibility principle, the methods should be split into separate classes where each class has a single responsibility. The 'API Deserializer' handles the returned JObject and the 'Models Parser' handles parsing between models. If the amount of methods increases, the 'Models Parser' can be broken down further, where the 'Models Parser' is a folder rather than a class, with each specific model parser being a separate class rather than a method. At the moment, a class to hold these is sufficient, as using a folder would introduce redundancy and folder structure clutter. A key detail within this example is enforcing appropriate naming conventions. The name of a class that has a single responsibility should not be generalized, it should be direct. 'Event Deserializer' is a general name, especially when there are lots of parsing functions and events passed around within the Calendar Manager application. The naming convention was the main reason for violating the single responsibility principle in this case.

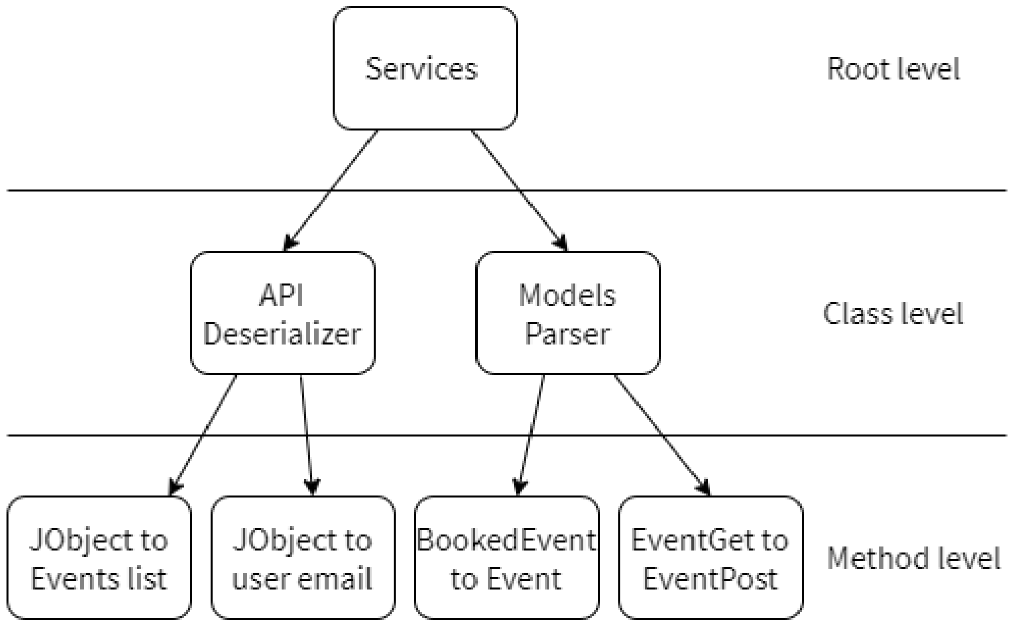


Figure 18 Single responsibility principle enforced.

7.3 Don't Repeat Yourself (DRY) principle evaluation

The DRY principle is self explanatory. Logic throughout the application should not be replicated (Smith, 2021). This has been achieved by creating specific constructs for business logic within the 'Services' folder and the class library, which are reusable. Additionally, the MVC design aids the DRY principle as constructed models are reused throughout the application. Upon low-level code evaluation, Calendar Manager contains several instances where this principle is being violated. This involves overusing local variables and local variable declarations, where a field declaration would be more appropriate. There have been many occurrences where a local variable was needed, but it was declared out of scope. This resulted in either rewriting logic to access this local variable or creating more local variables. This issue can arise again when extending the system. A solution to this problem would be to declare all major variables throughout all methods of a class as class member variables. This way, the class member variables, capturing the state of an object, can be accessed and reused from anywhere within the given class.

To adhere to the encapsulation principle, these class member variables must be private. Within C#, class member variables can be declared as fields or properties. A property is synthetic sugar for a private field with get and set methods. The get and set methods tend to clutter code, whereas a default property is written in one line. Therefore, the use of properties is preferable. Additionally, all dependencies passed through the constructor are assigned to a private field. Using properties aids with code readability and helps the user distinguish between the two types of class member variables, where the class state is captured by properties and the dependencies are capture by fields.

The DRY principle has been completely violated within the front-end implementation. The default calendar functionality has been achieved by constructing a single Vue instance. This gave the written JavaScript a well-defined structure. This Vue instance is a single component, a very extensive component. This large Vue instance has been copied over and used to create the booking calendar. This new Vue instance is slightly altered, however most of the code is repeated, with some code being repeated and redundant. A solution to this problem is to use Vue components. A component is just another Vue instance. The single calendar Vue instance can be split up into several Vue components. A component includes a template which holds HTML code. Templates can then be injected back into the HTML page, the View. Each component should have a single responsibility and should consist of one template. A proposed refactoring strategy involves splitting up the calendar Vue instance into a tree of reusable components. The calendar Vue instance is the root node. The child nodes are derived from the aforementioned HTML semantic structure, where each node corresponds to a semantic HTML element. The leaf nodes are then evaluated to decide whether it is appropriate to split them even further. For example, the calendar wrapper is split further into a component responsible for the calendar grid, and a component responsible for labeling this grid with hours. These components can then be injection into both the default calendar and the booking calendar, without duplicating their corresponding JavaScript code (You et al., 2021).

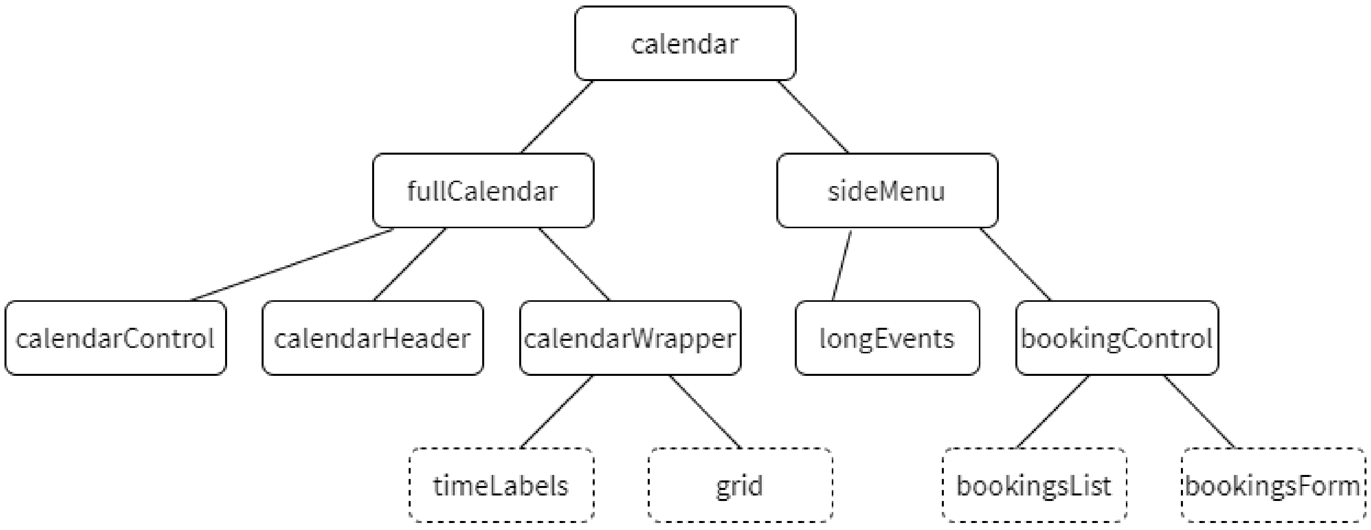


Figure 19 Proposed calendar component system.

7.4 Testing

Calendar Manager employs three testing strategies. Unit testing is used to test single units of code that do not have any dependencies. Automated unit tests have been used to test all classes of the class library, because these classes do not depend on any other classes. Integration testing is used to test units of code that have external dependencies. Automated integration tests have been utilized to test every class within the ‘Services’ folder. All classes within this folder have dependencies on other classes, mainly the ‘Models’ classes, and require their instantiation. Functional testing is used to test whether the requirements and specifications of the system are met, from the user’s perspective. Manual, functional tests have been employed to test the front-end functionalities of Calendar Manager.

An application should mostly utilize unit tests, followed by integration tests, and lastly, functional tests. This test distribution can be seen via the standardised testing pyramid (Smith, 2021). Calendar Manager mostly employs functional testing because the front-end calendar holds most of the functionality and logic within this application. To shift the actual testing distribution towards the desired distribution, the calendar preprocessing can be moved to the server-side where it can be tested using unit or integration tests. This would include all functionality within Vue.js ‘created’ method, such as filtering long events and computing current data. The results of these functions can then be passed to the front-end, avoiding the front-end preprocessing stage. The front-end testing could be further optimized by splitting the Vue instance into several components, portrayed in the previous section. These components can then be tested by a third party framework or a library, such as Vue Testing Library (You et al., 2021). The leaf nodes of the proposed component system would be classed as unit tests, and the remaining child nodes would be classed as integration tests. The result of this refactoring would adhere to the desired testing pyramid.

The full test plan is supplied in Appendix B.

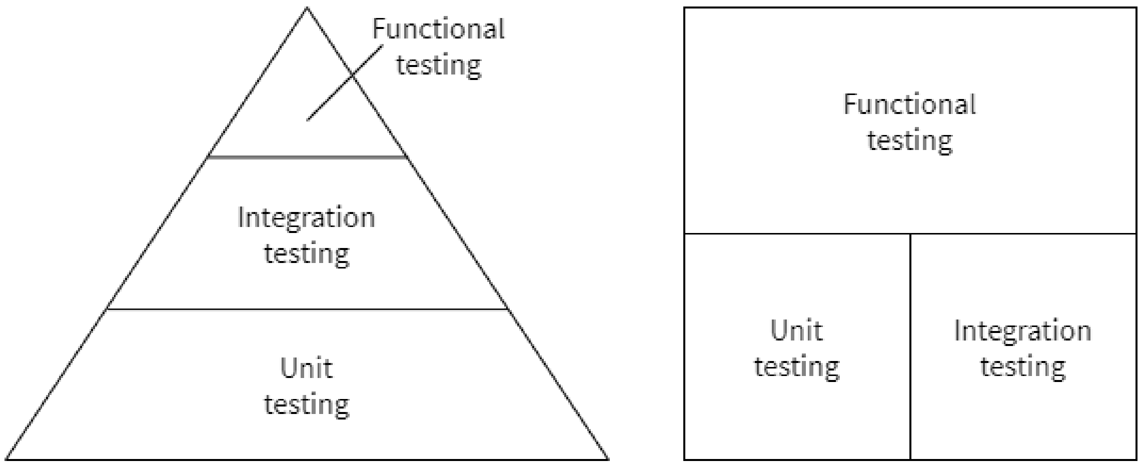


Figure 20 Desired testing pyramid (left) vs actual testing distribution (right).

7.5 Accessibility testing

The WCAG 2.1 revolves around four main accessibility principles. First, the perceivable principles ensures that all users are able to perceive all of the website content, using one or more of their senses. Second, the operable principle ensures that all website functionality must be operable using one or more standard input devices. Third, the understandable principle ensures that all content is readable and predictable for all users. Lastly, the robust principle ensures that all content and functionality works across different browsers and is designed using current web standards. Each of these four principles contains several guidelines. Each guideline contains several criteria which determine whether this guideline has been adhered to. Each criterion has a level of conformance from A to AAA, where A covers the most common accessibility issues and AAA covers rare accessibility issues. Passing AAA criteria is generally much more difficult than passing A criteria (Kirkpatrick et al., 2018).

The testing strategy chosen is to manually evaluate each guideline by iterating over all its criteria. If Calendar Manager adheres to all criteria within a guideline, then the test passes, otherwise it fails. Testing these guidelines can be classed as functional testing. Surprisingly, Calendar Manager adheres to most guidelines. This is because Calendar Manager is a smaller application and does not contain all possible constructs available within a web application. For example, guidelines 'Guideline 1.2 Time-based Media' and 'Guideline 2.3 Seizures and Physical Reactions' are adhered to, simply because Calendar Manager does not contain any time-based media, animations or videos. The second main reason why Calendar Manager passes most guidelines is because web technologies are already predefined and constructed with accessibility in mind. For example, guideline 'Guideline 2.1 Keyboard Accessible' is adhered to because browsers support keyboard shortcuts with inbuilt 'onFocus' visual aid to see the HTML element in focus.

Calendar Manager did not adhere to two accessibility guidelines. 'Guideline 1.1 Text Alternatives' states that each non-text element must contain a text alternative. This is achieved by setting an appropriate name value that describes the purpose of the non-text element. This criterion is Level A, meaning that it is very important to adhere to. Adhering to this criterion allows any element within a web page to be rendered visually, auditorily and tactilely. Luckily, adding name values to every element should be straight forward. The second failed guideline is 'Guideline 1.4 Distinguishable'. The criterion '1.4.3 Contrast (Minimum)' of Level AA, states that contrast ratios between text and its background must be of at least 4.5:1. Contrast ratio is the difference in luminance between two colors. All text within the application has a contrast ratio over 4.5:1. Calendar Manager also adheres to '1.4.11 Non-text Contrast' of level AA, which states that all visual components must have at least 3:1 contrast ratio against all adjacent colors. Any color combination within the Calendar Manager color scheme has a contrast ratio that is over 3:1. The criterion '1.4.6 Contrast (Enhanced)' of level AAA, states that contrast ratios between text and its background must be of at least 7:1. This criterion has not been passed as the lowest contrast ratio is 6.2. To pass this criterion and adhere to the overall guideline, a new color scheme for Calendar Manager should be chosen, because all text and its background follows this predefined color scheme.

The full test plan is supplied in Appendix B.

7.6 Documentation revised

To enforce the single responsibility principle, large components are split up into several smaller components with a single responsibility. At low-level, this results in the creation of more classes. At high-level, this results in the creation of more folders. As the application grows, the number of projects, folders and subfolders will also grow. The description and the purpose of classes and methods is covered by summaries and regions. However there are no documentation tools to cover projects and folders. Therefore, external documentation is needed that would describe the structure of the project, but also the contents of each folder. This addition would result in a sufficient documentation coverage.

During the unit testing stage, consisting of testing each method within the back-end business logic, it was discovered that a region label is not sufficient. It was difficult to remember what each method does and therefore, it was difficult to decide what to test for. To overcome this issue, in addition to regions, summaries should also be provided for each method within Calendar Manager. JavaScript does not provide the functionality of XML documentation, therefore a description of front-end methods should be supplied to the aforementioned external documentation describing the front-end structure.

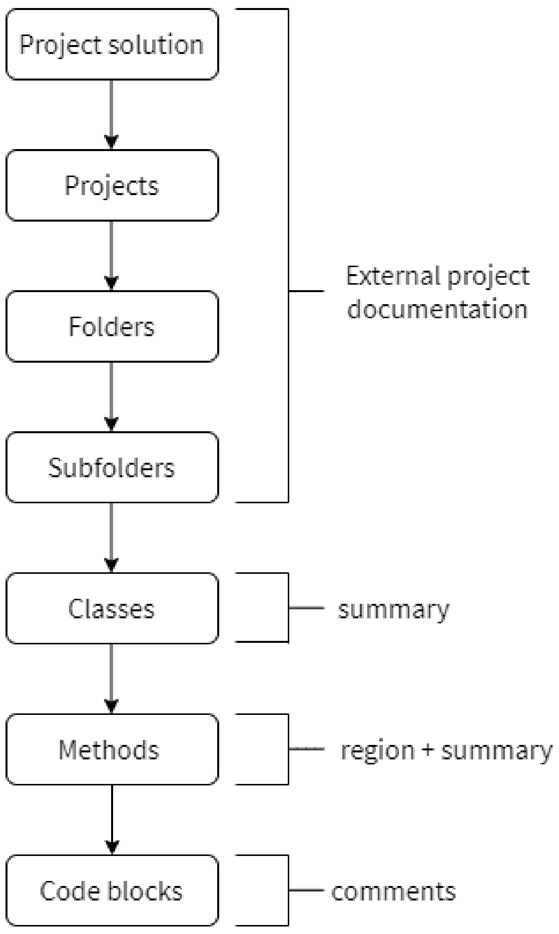


Figure 21 Proposed documentation coverage.

8 Conclusions

8.1 Summary of the project

Every substantial software development project has its highs and lows, with Calendar Manager being no exception. Retrieving events from the Google Calendar API has proved to be more difficult than expected. It required a full understanding and implementation of OAuth 2.0 protocol. Implementation of this protocol introduced the process of making requests and capturing responses via both, a predefined HTTP client, and URL parameters. It also introduced the handling of client-side secrets, user secrets, and assessing common security vulnerabilities during user authentication and authorization. Processing the API events into models was straight forward due to the MVC design. At this stage, it was discovered that storing client secrets within the browser can introduce security vulnerabilities. It was therefore decided, that an underlying database will be used to store the user secrets and control the state of Calendar Manager. Integrating a database into the application was straight forward because the entity framework allowed the use of C# syntax to construct the database and handle data. Additionally, SQL Server Management Studio (SSMS) provided a very intuitive interface which was used to query the database. The routing problem which consisted of incorporating dynamic URLs within the system was tricky. However, it was a very interesting problem to solve, as it provided an insight into the application pipeline and middleware. The front-end implementation was extremely tricky. The combination of learning a new framework, the weakly typed nature of JavaScript, and the amount of functionalities needed for the calendar, introduced many hurdles throughout the front-end development. Nevertheless, these hurdles were overcome and project aims defined at the start of this project have been met.

8.2 Future work

The strict time constraint has been the major limitation to this project. With regards to software design and architecture, several adjustments and re-implementations are suggested within subsections of the 'Analysis' section. An improvement in software design will enable easier implementation of additional features. Implemented additional features, on top of the current Calendar Manager skeleton, may result in a more difficult design re-implementation. Therefore, these are the most critical aspects of future work and should be prioritised.

Calendar Manager has been specifically designed to be extendable and maintainable to allow for future addition of extra features. These features can be appended onto the skeleton Calendar Manager. During the initial background research of calendar scheduling web applications, a list of proposed features was constructed. This list is also appended with feature suggestions provided by the project supervisor and inspector.

- View the default calendar as an image.

- Generate a shareable link for the default calendar.
- Generate a shareable link for the default calendar in incognito mode (hidden events).
- Merging additional calendars into the original calendar.
- Calendar owner authorization to allow optional approval of events.
- Email notifications to calendar owner.
- Support for additional views of the default and booking calendars (daily, monthly, yearly).
- Support for Outlook and Apple calendars.
- Real-time calendar updating.

8.3 Personal development

My main personal aim of this project was to develop my software engineering skills. With previous knowledge in C#, my goal was to expand upon this by learning about software design. I believe that this project has given me great exposure to software engineering design and principles, which I believe have been appropriately utilized within this project.

With extremely limited knowledge within front-end development, my goal was to learn a JavaScript framework. Even though there is a lot to Vue.js framework, I believe that I have managed to grasp the basics of this framework. Any future Vue.js development will allow me to build upon the core basics learned throughout this project.

The one thing that I will definitely take with me, until the end of programming days, is adding alternative text to every non-text HTML element so that my websites pass the '1.1 Text Alternatives' Level A accessibility guideline. Such a simple implementation that can make such a great difference to many users.

8.4 Reflections

Throughout the project, I have thoroughly enjoyed developing the back-end of Calendar Manager. I find that building upon already known skills is easier and much more interesting. My previous experience with the fundamentals of C# enabled me to study the design and architecture of web applications by exploring the intricacies of the ASP.NET framework in detail.

The front-end implementation of Calendar Manager has given me the most trouble throughout the project. I have greatly underestimated the amount of time that it would take to implement a fully functioning calendar with an accompanying calendar booking system. The calendar required the implementation of many unforeseen functionalities and edge cases. These issues were exacerbated by the weakly typed JavaScript language and the lack of front-end debugging tools.

I must concede however, that it is naive to think that software engineering projects run smoothly. If I had to start this project again, I would strip the project aims, features and functionalities to a bare minimum, and I would add contingencies for my contingencies. Regardless, these struggles were a vital part of my personal growth throughout the project. Developing Calendar Manager has certainly exposed me to new skills and knowledge, which have allowed me to achieve my personal aim of this project.

A Test plan

A.1 Server-side testing

Test description	Result
Rounds down minutes (int) to nearest five	Pass
Short URL correct length	Pass
Randomly generated 100 short URLs do not clash	Pass
State token correct length	Pass
Randomly generated 100 state tokens do not clash	Pass

Table 4 Unit tests.

Test description	Result
List of BookedEvent conversion to List of Event type check	Pass
Chop 60min event with 20min period	Pass
Chop 150min event with 30min period	Pass
JObject conversion to List of Event type check	Pass
Retrieve email (string) from JObject type check	Pass
Retrieve email (string) from JObject contains ‘@’	Pass
ChosenEventResponse conversion to ChosenEventPost type check	Pass

Table 5 Integration tests.

A.2 Client-side use-case testing

Test description	Result
User email is correct	Pass
Initial dates and days of a week in a month correctly populated	Pass
Initial dates and days of a week over two months correctly populated	Pass
Initial month correctly populated	Pass
Initial months correctly populated where two months overlap	Pass
Initial year correctly populated	Pass
Initial years correctly populated where two years overlap	Pass
Previous button updates days, dates, months, years correctly	Pass
Next button updates days, dates, months, years correctly	Pass
Leap years have 29 days	Pass
The previous button is disabled upon reaching the year 2010	Pass
List of long future events is correct	Pass
List of long future events is in an ascending order of start date	Pass
Event spanning over one day is correctly displayed	Pass
Event spanning over two days is correctly displayed	Pass
Event spanning over two days and over two weeks is correctly displayed	Pass
Event spanning over two days and over two months is correctly displayed	Pass
Event spanning over two days and over two years is correctly displayed	Pass
All inserted events match Google calendar’s events color	Pass
Populated events are disabled	Pass
Booking prompt displays correct one hour time block based on user click position	Pass

Table 6 Functional tests #1.

Test description	Result
Booking prompt contains validation for incorrect time format	Pass
Booking prompt contains validation for time which overlaps an event	Pass
Booking prompt contains validation for time which overlaps a previously booked event	Pass
Booking slots are correctly displayed in the calendar	Pass
Booking slots are correctly displayed in the booking slots list	Pass
Booking slots are correctly displayed in an ascending order	Pass
Close button deletes the corresponding booking from the booking list	Pass
Close button deletes the corresponding booking from the calendar	Pass
Period name has a 50 character limit	Pass
Period allows integer input only	Pass
Submit button is enabled once a booking slot is added	Pass
Submit button displays the booking session shareable URL	Pass
Booking button opens a booking popup	Pass
Booking popup time and date correspond to the booking button	Pass
Student name input has a 50 character limit	Pass
Submit button writes an event into owner's Google Calendar	Pass
Submit button writes an event into owner's Google Calendar with period name	Pass
Submit button deletes the booked slot from the booking calendar	Pass

Table 7 Functional tests #2.

A.3 Usability testing

Accessibility guideline	Result
1.1 Text alternatives	Fail
1.2 Time-based media	Pass
1.3 Adaptable	Pass
1.4 Distinguishable	Fail
2.1 Keyboard accessible	Pass
2.2 Enough time	Pass
2.3 Seizure and physical reaction	Pass
2.4 Navigable	Pass
2.5 Input modalities	Pass
3.1 Readable	Pass
3.2 Predictable	Pass
3.3 Input assistance	Pass
4.1 Compatible	Pass

Table 8 Functional tests #3.

B Source code

The project solution can be found on the University of Birmingham git server. All code written is my original work, unless otherwise specified.
https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2020/mxr080

References

Anderson, Rick. (2019). **Enforce https in asp.net core**. Retrieved July 8, 2021, from https://docs.microsoft.com/en-us/aspnet/core/security/enforcing-ssl?view=aspnetcore-5.0&tabs=visual-studio

Anderson, Rick et al. (2021). **Aspnetcore.docs**. Retrieved July 6, 2021, from https://github.com/dotnet/AspNetCore.Docs

Anderson, Rick, Larkin, Kirk, Roth, Daniel & Addie, Scott. (2020). **Safe storage of app secrets in development in asp.net core**. Retrieved July 25, 2021, from https://docs.microsoft.com/en-us/aspnet/core/security/enforcing-ssl?view=aspnetcore-5.0&tabs=visual-studio

Anderson, Rick & Smith, Steve. (2020). **Asp.net core middleware**. Retrieved July 8, 2021, from https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-5.0

Auger, Robert. (2010). **The cross-site request forgery (csrf/xsrf) faq**. Retrieved July 13, 2021, from https://www.cgisecurity.com/csrf-faq.html

Azad, Badi. (2020). **Openid connect**. Retrieved July 13, 2021, from https://developers.google.com/identity/protocols/oauth2/openid-connect#createxsrftoken

Azad, Badi. (2021). **Using oauth 2.0 to access google apis**. Retrieved July 9, 2021, from https://developers.google.com/identity/protocols/oauth2

Daugherty, Brian. (2021a). **Using oauth 2.0 for web server applications**. Retrieved July 28, 2021, from https://developers.google.com/identity/protocols/oauth2/web-server#incrementalAuth

Daugherty, Brian. (2021b). **Using oauth 2.0 for web server applications**. Retrieved July 28, 2021, from https://developers.google.com/identity/protocols/oauth2/web-server#protectauthcode

de Kunder, Maurice. (2016). **The size of the world wide web (the internet)**. Retrieved July 5, 2021, from https://www.worldwidewebsize.com/

Dix, Alan, Finaly, Janet, Abowd, Gregory & Beale, Russel. (2005). **Human-computer interaction**. Retrieved July 26, 2021, from https://scholar.google.co.uk/scholar?q=Human-computer+interaction++alan+dix&hl=en&as_sdt=0&as_vis=1&oi=scholart

Gould, Sandy, Cox, Anna, Brumby, Duncan & Wiseman, Sarah. (2016). **Short links and tiny keyboards: A systematic exploration of designtrade-offs in link shortening services**. Retrieved July 13, 2021, from https://www.sciencedirect.com/science/article/pii/S1071581916300854

Hardt, Dick. (2012). **The oauth 2.0 authorization framework**. Retrieved July 9, 2021, from https://datatracker.ietf.org/doc/html/rfc6749#section-1.2

Huckle, Natasha. (2019). **Evaluation using PROMPT**. Retrieved June 22, 2021, from http://www.open.ac.uk/libraryservices/documents/advanced-evaluation-using-prompt.pdf

Kappert, Lars et al. (2021). **Programming principles**. Retrieved July 6, 2021, from https://github.com/webpro/programming-principles

Kirkpatrick, Andrew, Connor, Joshue, Campbell, Alastair & Cooper, Michael. (2018). **Web content accessibility guidelines (wcag) 2.1**. Retrieved July 13, 2021, from https://www.w3.org/TR/2018/REC-WCAG21-20180605/

Lodderstedt, Torsten et al. (2013). **Threat: Disclosure of client credentials during transmission**. Retrieved July 23, 2021, from https://datatracker.ietf.org/doc/html/rfc6819#section-4.3.3

Mandalios, Jane. (2013). **Radar: An approach for helping students evaluate internet sources**. Retrieved July 5, 2021, from https://journals.sagepub.com/doi/abs/10.1177/0165551513478889

MDN-Contributors. (2021). **Html: Hypertext markup language**. Retrieved July 13, 2021, from https://developer.mozilla.org/en-US/docs/Web/HTML

Myhre, Sarah. (2012). **Using the craap test to evaluate websites**. Retrieved July 5, 2021, from https://scholarspace.manoa.hawaii.edu/handle/10125/22479

Pekarsky, Max. (2020). **Does your web app need a front-end framework?** Retrieved July 10, 2021, from https://stackoverflow.blog/2020/02/03/is-it-time-for-a-front-end-framework/

Smith, Steve. (2021). **Architecting modern web applications with asp.net core and microsoft azure**. Retrieved July 6, 2021, from https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/

W3Schools-Contributors. (2021). **Html references**. Retrieved July 13, 2021, from https://www.w3schools.com/html/

Wagner, Bill. (2021a). **C# preprocessor directives**. Retrieved August 22, 2021, from https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/preprocessor-directives

Wagner, Bill. (2021b). **Recommend xml tags for c# documentation comments**. Retrieved August 22, 2021, from https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/xmldoc/recommended-tags

Wikepdia-Contributors. (2021a). **Circular dependency**. Retrieved August 3, 2021, from https://en.wikipedia.org/wiki/Circular_dependency

Wikepdia-Contributors. (2021b). **Model-view-viewmodel**. Retrieved August 14, 2021, from https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel

Wikepdia-Contributors. (2021c). **Universally unique identifier**. Retrieved July 26, 2021, from https://en.wikipedia.org/wiki/Universally_unique_identifier

You, Evan et al. (2021). **Vue.js guide**. Retrieved July 13, 2021, from https://github.com/vuejs/vuejs.org/tree/master/src/v2/guide