REPUBLIC OF TURKEY

YILDIZ TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CALENDAR APPLICATION TO GROUPS TASKS AND EVENTS

17011081 — Ali Barış Zengin

18011071 — Elif Yağmur Duran

SENIOR PROJECT

Advisor

Instructor Furkan ÇAKMAK

Mayıs, 2022

ACKNOWLEDGEMENTS

We would like to thank our mentor and instructor Furkan Çakmak for his invaluable assistance and insights leading to the writing of this paper. Also our most sincere thanks goes to the academic staff of Computer Science and Engineering Department of Yıldız Technical University, for all their efforts and hard work that went into our education. We would also like to thank all the open source developers that helped us realize this idea and bring it to life. And finally a special thanks to the classmates who have aided us in the many obstacles we faced during the process of this project.

Ali Barış Zengin
Elif Yağmur Duran

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

API	Application Programming Interface
CI/CD	Continious Integration/Continious Deployment
CPU	Central Process Unit
ER	Entity-Relation
GB	Gigabyte
GTD	Getting Things Done
HTTP	Hyper Text Transmission Protocol
KVKK	Kişisel Verilerin Korunması Kanunu
MIT	Massachusetts Institute of Technology
NOSQL	No-Structured Query Language
UI	User Interface
UML	Unified Modelling Language
RAM	Random Access Memory
REST-API	Representational State Transfer
SOAP	Simple Object Access Protocol
SQL	Structured Query Languag
SSD	Solid State Drive

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Calendar Application To Groups Tasks and Events

Ali Barış Zengin
Elif Yağmur Duran

Department of Computer Engineering
Senior Project

Advisor: Instructor Furkan ÇAKMAK

In this study, a web based responsive application was developed, in order to bring a solution to a certain problem, which is having too many apps operating over a person's productivity system. The goal was to create an app that would allow the user to have a easy to use and hard to clutter workspace that syncs all kinds of tasks and events from their other preferred apps, such as Google Calendar or Outlook. Reason for this approach is that, while all these apps are similar in nature, they all have their ups and downs and none are superior to another. Most users tend to prefer them both. We offer a solution by an app that brings these two together.

Firstly, a research on productivity systems and certain habits of the target audience was conducted. After the modules of the system were decided upon, the back-end was built by express.js, which is a framework of node.js. MongoDB was preferred on the database side, for it's compatibility with express.js system. Other implementation to make the back-end side were also done at this stage. The front-end side was implemented using the React framework.

Keywords: calender, event organizing, task organizing

ÖZET

Görev Ve Etkinlik Gruplayan Takvim Uygulaması

Ali Barış Zengin
Elif Yağmur Duran

Bilgisayar Mühendisliği Bölümü
Bitirme Projesi

Danışman: Öğr.Gör. Furkan ÇAKMAK

Projemizde iş hayatını ve özel hayatını planlarken birden çok uygulama kullanmak zorunda olan bireylere, farklı takvimlerini bir arada görebilmeyi sağlayabilen bir web uygulaması yapmayı planlıyoruz. Uygulamamıza üye olan kullanıcılar, Outlook veya Google Calendar hesaplarını uygulamamıza entegre edebilecektir. Diğer takvimlerden getirilen etkinlik ve görevlerin uygulamamızda senkronize bir şekilde görülebilmesi özelliğinin yanı sıra, kullanıcılar uygulamamızın hesabına özel takvim özelliklerinden de faydalanabilecektir. Bu takvim özelliklerinden kastımız, kullanıcı hesabına özel görev ve etkinlikler oluşturabilmenin yanı sıra, takvimde toplanan görev ve etkinlikleri kullanıcının yaratabileceği kapsamlarda gruplandırabilme seçenekleridir. Bu şekilde kullanıcılar, hayatlarındaki bütün görev ve etkinlikleri bir arada görebilecek ve düzenleyebilecektir, ayrıca bunu yaparken verileri güvende olacaktır, üstelik uygulamamız ücretsizdir.

Anahtar Kelimeler: takvim, iş yönetimi, süreç yönetimi

1 Introduction

One of the greatest challenges the modern humans face is, without a doubt, organizing their daily lives in between the hustle of work and life balance. A day is only 24 hours, and it is rarely enough accomplish all that there is to do for an individual. Time management methods have long been a topic of discussion, but there is also the question of how to find energy to do all that there is to do. Yes, modern humans are burdened with the responsibility of not only managing their times, but also their energy to get behind on all the deadlines the busy life has been pushing upon them. The question this study has asked is, how can we aid these individuals in their quest with the power of software development.

Much research has been done on the topic of time (and energy) management, that this study could itself alone, never do justice to. Nearly every single person has a personalized system to get on top of every event or task of their daily life. Some keep it on a paper, a physical agenda, some gravitate towards options application have offered. The problem identified by and attempted to be solved in this study, is the problem of having too many options for a productivity system, all with their own merits and demerits, and no way of bringing these together for a user, so that they don't have to choose between very different apps, and see everything together instead.

The application that will be explained and built in this study, is one to synchronize all data that is pulled from popular productivity apps like Google Calendar and Outlook, and display it in it's own unique interface. In addition to all the data crawled from the other two apps, users will also be able use a reservation system our team personally has designed. The application is also preferable to others of it's kind because of the privacy it provides to users, and it's free and open source.

2 Preliminary Research

Before constructing the software of the system, a design for the productivity system it will use which will speak to users of all kinds must be agreed upon. The goal for the project is to, firstly offer a platform to help users sync their stuff from all kinds of productivity apps, and secondly offer a reservation system.

In light of this intent, a preliminary research on many other apps who have also offered similar solutions was done. One such website that was a source of inspiration would be, simplybook.me, which is a website designed to allow users to open hours for reservations and let others book spaces from a timetable. Also different system ideas by different productivity experts have been inspected and taken into account as inspiration. One remarkable one to give an example would be, "Getting Things Done" system by David Allen[1].

The idea he had was that most problems of the productivity systems arise from making the users think about doing the tasks too much, and causing them to have less energy to spend on actually doing. The idea of energy management[2] instead of time management has been a concept within the productivity community for a long time, particularly emerging around the time the legendary productivity book, Atomic Habits by James Clear was released, and nearly every single system and concept that can be talked about on the topic has emerged from the book, and is a huge recommendation for beginners.

But back to the GTD concept, the principle is based on users dumping the contents of everything going on in their life into an inbox, then comes the process of sorting said pile. This allows a user to give the thinking mind a rest, which helps in saving more energy into the actual productivity process.

To summarize, firstly, user dumps everything they can think of into an inbox. Then for every content, a question is asked. Is it actionable or not? If no, it is either an event that is to be scheduled, or an item that does not belong to the calendar system and is of no concern to us. Basically the idea is to sort out tasks from other things. If the answer is yes, it would be passed into the task management system, which is not the concern of this project. Our project mainly focuses on helping the users organize their work and life events in one place, and also have other people in their life see their schedule to find themselves a place in there.

These have been the ideas from said systems and articles that were researched prior to start of the projects software development side. One might also ask, what is the point of all this research, and the effort? Can't people just, well, remember things?

And the answer to that would be no. In today's world it is extremely easy to get lost in the sea of deadlines, most people can only get to the most essential ones at the very last minute, which are mostly school or work deadlines. This state of constant disorganization causes individuals to miss out on precious time they could spend on other things in life, such as time for oneself, time for family and relationships, time to relax and unwind. Not to mention systems like this are known to be lifesavers for neurodivergent individuals, who need help with getting on top of even the most basic essentials sometimes. This application hopes to help people from all kinds of backgrounds, and conditions, with all kinds of brains, to help spend their time more wisely.

3
Feasibility

3.1 Technical Feasibility

In this section, the necessary software and hardware determination process for the project has been carried out based on the minimum specifications.

3.1.1 Software Feasibility

Both front-end and back-end software will be needed to realize the project. Also, a database software is needed to keep the models.

The basic structures for the front-end will be HTML, CSS and Javascript. In addition, the project will be coded with ReactJS, a Javascript library that works in accordance with the MVC principle. The reason for this being that they are common, easy to understand and basic technologies for web development. Working with the JSX extension, ReactJS stands out among Javascript libraries with its high DOM update performance and extensive use of libraries.

For the back-end, Express.js was preferred, which provides the continuation of coding with Javascript. Ease of use and asynchronous operation of middleware modules is the reason Express.js was preferred. also, providing front-end and back-end software integrity through Javascript-based React and Express.js will facilitate full stack project development.

MongoDB will be used as database software. MongoDB, which can be used over Express.js with the Mongoose package, offers a cloud service as MongoDB Atlas. MongoDB will be used as database software because it provides integrity and compatibility with the rest of the project.

The preferred development environment will be Visual Studio Code. It was chosen because of the wide variety of plugin options and powerful developer tools.

3.1.2 Hardware Feasibility

Personal computers used by the developers will meet the requirements of the research, design and implementation phases of the project. The hardware specifications of the computers that will meet the minimum needs in the development of the project are as follows.

- **Processor**: Intel Core i5-9300H
- **RAM**: 8GB
- **Memory**: 512GB SSD
- **GPU**: GeForce GTX 1650
- **OS**: Windows 10 Pro 64 bit

3.2 Economic Feasibility

Below is the economic feasibility created for the software and tools required for project development.

Feature	**Cost (Dollar)**	**Tax (Dollar)**	**General (Dollar)**
Employee Wages x2	700$ x 3 Months x 2 Employees	-	4200$
Developer PCs x2	800$ x 2	150$ x 2	1900$
Product Environment	180$	30$	210$
Sum	**5980$**	**330$**	**6310$**

Table 3.1 Economic Feasibility Table

3.3 Legal Feasibility

All programming languages, libraries and software tools used in this project are licensed, which are free for personal or commercial use and do not pose any problems. After the project is deployed, all data to be obtained from users will be subjected to the Law on Protection of Personal Data (KVKK) No. 6698.

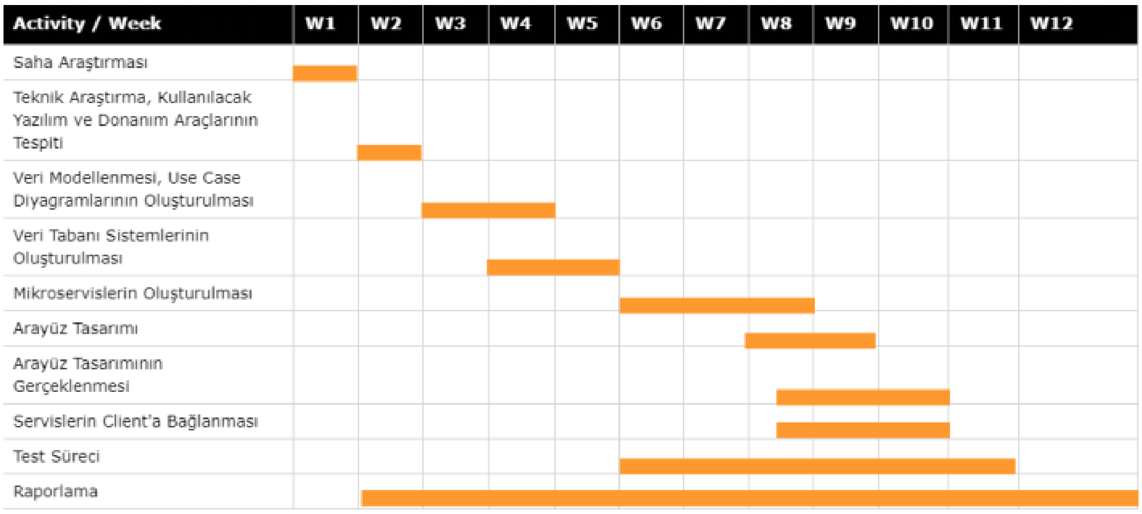
During the back-end development process, npm package manager was used. npm package manager, also known as Node Packet Manager, is a repository for using open source 3rd party software, tools and modules for JavaScript enhancements. If a developer has uploaded their software to the npm repository, they have accepted it's usage by others.[3]

Visual Studio Code was used as the main source code editor, with the aid of certain auxiliary tools that can be connected to VS Code as well. Every tool and software used in the project development process was used free of charge. The list of explanations regarding the use of the related software and tools together with their names is as follows.

3.4 Labor and Time Feasibility

Within the scope of the project, firstly a field research is carried out for competing products and infrastructures. Next is determining the necessary technical tools for the desired application and questioning the feasibility of the project. After this, comes the Data Models, and then database systems are created. It is important to start the interface design after the service layer is created, this is for the good of the subsequent interface implementation. The testing process starts in a way that supports agile development and continues in all software processes. Reporting is progressed in parallel with all processes.

The Gannt diagram for the process of the project, is as follows in Figure 3.1.



Activity / Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12
Saha Araştırması	■											
Teknik Araştırma, Kullanılacak Yazılım ve Donanım Araçlarının Tespiti		■										
Veri Modellenmesi, Use Case Diyagramlarının Oluşturulması			■	■								
Veri Tabanı Sistemlerinin Oluşturulması				■	■							
Mikroservislerin Oluşturulması						■	■	■				
Arayüz Tasarımı								■	■			
Arayüz Tasarımının Gerçeklenmesi								■	■	■		
Servislerin Client'a Bağlanması								■	■	■		
Test Süreci						■	■	■	■	■	■	
Raporlama		■	■	■	■	■	■	■	■	■	■	■

Figure 3.1 Gantt Diagram

4 System Analysis

Time management systems, as it can be derived from the name, are systems that are designed to aid the user in time management. When these applications are examined in terms of software, it is seen that they are systems where many data types and data connections are offered on a single platform. This causes the designs and methods of the relevant models to not be implemented within a single structure.

Usually during the building process of such systems, a relational model is created firsthand. Models need to be related to each other in action states. It is aimed that these actions are independent structures, that is, they can be developed without being affected by other models and their actions. In this way, studies carried out with module logic are transferred to software architecture by dividing large problems into small parts. The systems that mentions should also be able to adapt to the variability of daily life. Account management used in the system and should be able to follow modern developments. They must be versionable at the time of publication. In this case, the design analysis of such a system can be shown in parts as explained in the next section.

4.1 Data Modeling

The calendar applications are based on the events of the user. So the calendar as a whole would be the basic model. In the related projects, the user has a calendar and can make any labeling or classification they want in it. Even if the scope of each project changes, there are features such as user authentication, user profile operations, creating and editing tasks/events in certain time periods - according to the project's own naming - and notification settings if it is a mobile application.

In this particular case, a visual representation of the features can bee seen in Figure 4.1 below.

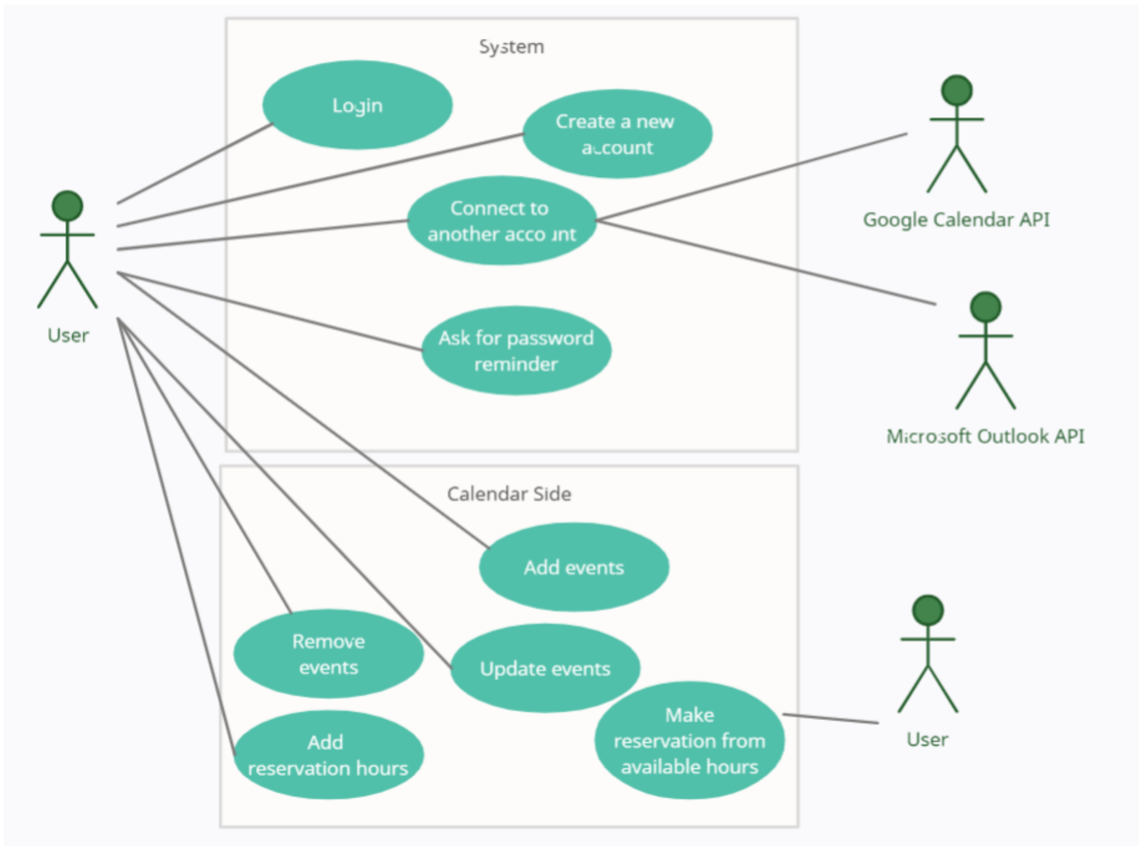


Figure 4.1 Use Case Diagram for the Calendar Application

The user has a calendar and includes the person's time plan, which is specified in parts divided into time zones. This planning can be decomposed in different ways according to demand.

For these reasons, a user model includes a calendar model as well as personal information. The calendar model, on the other hand, contains the sub-parts task/event model. Each related model includes a time period, classification or labeling, title, detail and similar information.

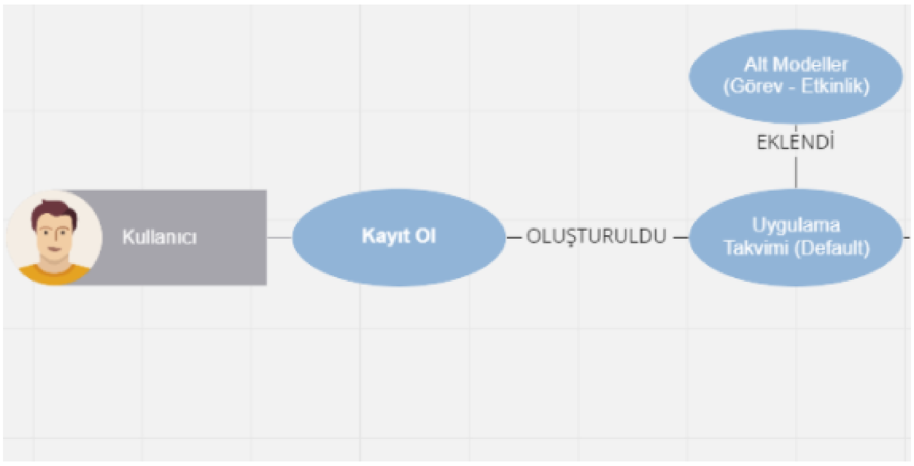


Figure 4.2 Steps Of Data Modeling

4.2 Data Storage

After the data modeling is done, the issue of data storage comes. After all data models are created, a Model-Relation Diagram (ER-Diagrams) is created between the set of values that the concepts can take and the other data models with which they are related. Based on this diagram, the database system can be shaped in different ways. Necessary investigations and studies are carried out to establish the most appropriate and correct system according to the project details to be created during the working process.

The database type to be chosen for storing the data is generally divided into 2 popular types. SQL type databases and NoSQL type databases should be selected according to the scope and content of the project.

4.2.1 SQL Database

SQL model databases started to become popular in the 1980s and still remain popular. This model, which can also be called the relational database model, is a table-based model with columns, rows and the information it contains. The table sets it contains are predefined, statically structured schemas. Allow data import, add, delete, update operations With its relational structure, it has taken great efforts and steps to make itself a standard from the distant past of the software to the present, and it is used in most projects today. SQL databases include Sybase, MySQL, PostgreSQL, Microsoft SQL Server, Oracle, IBM Informix and Microsoft Access etc. can be given as an example.

4.2.2 NoSQL Database

No-SQL databases, contrary to the concepts of SQL databases, do not require predefined schema data. That is, it has a dynamic structure. There are different data storage methods such as document-based, graph-based, key-value-based. The model performs well for situations where the set of values is very variable or of variable length. In the related event and calendar management projects, it contains data that can be added or deleted even just daily, due to the variability of historical proximity and distance of user actions. For this reason, NoSQL databases are a suitable choice for similar applications. Examples of NoSQL databases are MongoDB, DynamoDB, Cassandra, Couchbase.

4.2.3 SQL vs NoSQL

	SQL	NoSQL
Database Type	Relational	Distributed
Data Model	Table Based	Document Based Key-value Based Graph Based
Advantage	Complex Query Management Static, Structural Workstyle Non-Free or Open Source	Fast Horizontal Scaling Dynamic Workstyle Open Source

Table 4.1 SQL vs. NoSQL Comparison Table

So which one to use? If the project contains complex queries and the project is not expected to change much in the upcoming period, that is, if you do not expect much change in the traffic volume and data types of your application in the future, it is better to choose one of the relational databases.

NoSQL databases should be preferred for fast working methods, keeping the project as dynamic as possible, being open to change, and scaling the bed with a fast working method.

4.2.4 Other Databases

Other types of databases can be exemplified as distributed databases, object-oriented databases, cloud databases, multi-model databases, and the like. In the current situation, NoSQL databases, which have come a long way in the way of standardization from past to present, are preferred together with SQL databases, which are more evident with their use in the current period. For this reason, these 2 databases are considered as the basis and the details of the others will not be given.

4.3 Architecture Design

It is necessary to define procedures such as how the software system will communicate with each other, which endpoints and which information (SOAP, REST-API) will be accessed during the coding start, and it should be known from which data paths this information can be accessed while applying. In project development processes, we come across 2 different common design architectures. One of them is the Monolith (Single Roof) design architecture that we apply in our normal work, even if we do not notice it. Microservices design architecture, which manifests itself with its benefits in the recent software segment, and which is implemented by most business models, is another. The main differences are that the Monolith design architecture does all the work within a project (Service), while the Microservices design architecture creates a service structure for each job and uses the Gateway system for its communications. In this way, it provides many benefits with the execution of development, testing and similar processes of different front-ends independently of each other.

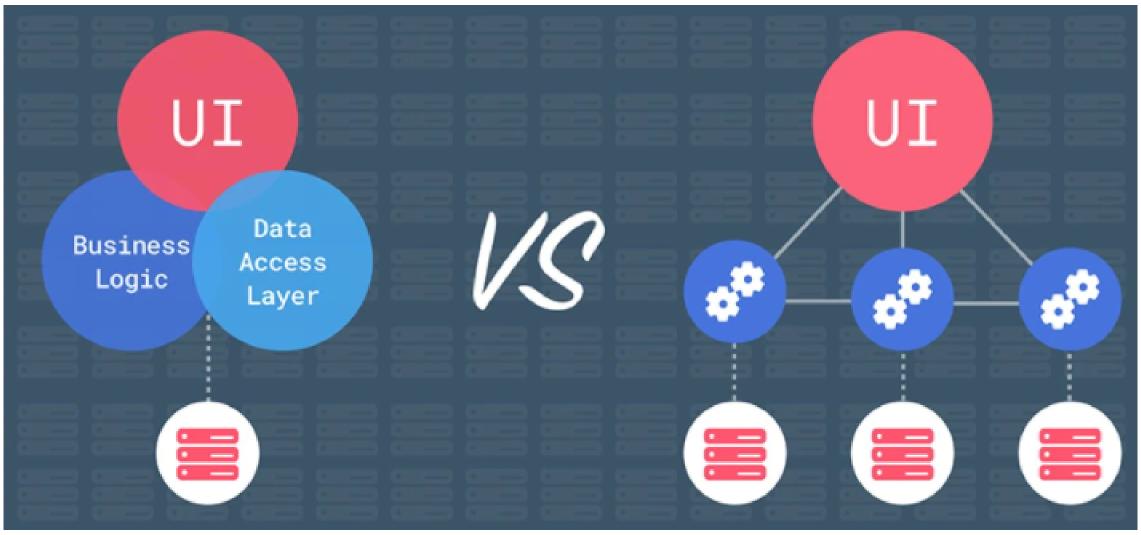


Figure 4.3 Monolith and Microservices Architecture Designs [4]

4.3.1 Monolith Architecture Design

The Monolith architectural system includes the entire application in a single main-frame (project file). This is where all endpoint queries and the request-response pair It is the name given to an approach that envisages coding from within the main-frame. The fact that there are many interactions and models in the project makes the use of Monolith design architecture inconvenient. Both the software dimension and the interaction dimension of the basic function of the project, Authentication - Communication with the database - may vary. In addition, these 3 basic parts may not be in every project, but most projects include more. For this reason, if there is not much expectation for the scope of the project and the future of the project, it would be appropriate to use Monolith design architecture. In this way, many installation and deployment processes brought by the Microservice architecture are avoided and allow rapid development.

4.3.2 Microservices Architecture Design

It is the name given to an approach that envisages coding from within the main-frame. The fact that there are many interactions and models in the project makes the use of Monolith design architecture inconvenient. Both the software dimension and the interaction dimension of the basic function of the project, Authentication - Communication with the database - may vary. In addition, these 3 basic parts may not be in every project, but most projects include more. For this reason, if there is not much expectation for the scope of the project and the future of the project, it would be appropriate to use Monolith design architecture. In this way, many installation and deployment processes brought by the Microservice architecture are avoided and allow rapid development.

In the Microservice Architecture, each service makes requests to perform the necessary operations with each other, and the related interaction and result reveal the work. So, when the client-side wants to do the relevant action, should the front-end send a separate request to each service? What is the factor that unites these different services as a service, makes it whole as a project, and enables it to receive requests from outside? Here comes the API-Gateway. It determines which service each request on a specific URL is related to, and directs the request to the relevant services for related actions.

5
System Design

5.1 Data Modelling

System of the project revolves around the user. Every user has a personal account and therefore a personal calendar system. Since they all must engage with an independent application than the other calendar apps they use, it is required for all users to create a new account in the beginning. After the user creates an account, their desired accounts from other apps can be synced, this will no be limited to one per every application either.

This makes the center model, the user. User models calendar will be accessed by and managed with the aid of the interface. User will also sync data with the other applications. At this point, the center model will be the user and the data and the functionality of our goal are will be the calendar/task models. That's why, we need to develop a relation that user have more than one calendars.

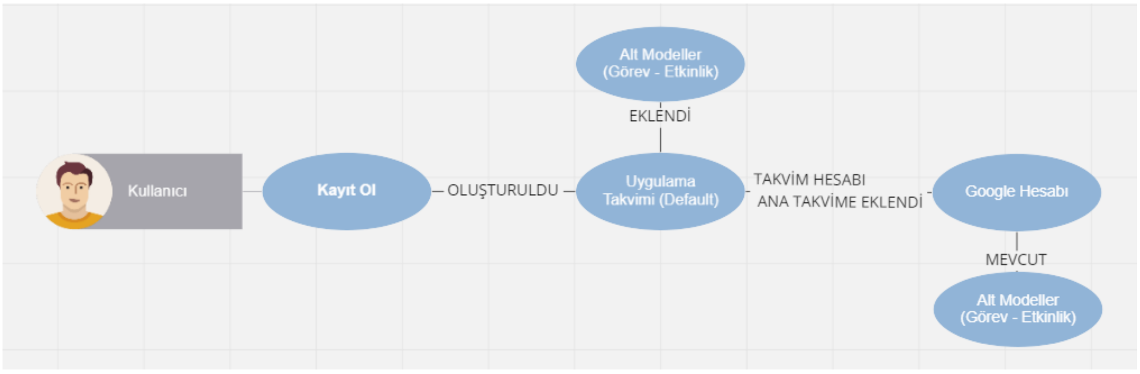

Kullanıcı
Kayıt Ol
OLUŞTURULDU
Alt Modeller (Görev - Etkinlik)
EKLENDİ
Uygulama Takvimi (Default)
TAKVİM HESABI ANA TAKVİME EKLENDİ
Google Hesabı
MEVCUT
Alt Modeller (Görev - Etkinlik)

Figure 5.1 Data Model Steps

Likewise, each calendar will contain different events and tasks, and the desired labels will be possible for each allocated time period. For this reason, calendar models may contain different event and task models, and each sub model will keep its own time zone and information.

5.2 Architecture Design (Microservice)

There will be a lot of new validation rule adjustments with new updates on the endpoint in the presence of schemes with many relationships and user action schemes. It was clear from the beginning of the project. For this reason, it is a large piece with a lot of change. Instead of coding these use cases, we break them down into relevant small chunks. It is more suitable for dynamic content and managed systems to communicate with endpoints. For this reason, the user-calendar-integration triangle is separated from each other. Externally managed within themselves will facilitate performance and workforce. In this context, within the scope of the project, the relevant parts are combined into small parts. When we divide it, the following microservices emerge:

- API Gateway: It is the microservice that will act as a Proxy (Proxy Server) for the relevant microservice transfer of incoming requests.
- Auth Microservice: Microservice that manages all user reactions (registration, login, password) change, profile editing, email confirmation, etc.) as authentication.
- Integration Service: It is the service that will allow importing calendars with all account integrations.
- App Microservice: Microservice that will enforce all business model rules.
- Database Service: It is a service that hosts the database and can grow independently in the horizontal.

The parts of all microservices that deal with the user request-response pair will be processed according to the type and content of the request from the user using HTTP REST API standards and the response will be returned. RabbitMQ, one of the third party Queue and Pub/Sub(Publisher/Subscriber) applications, will be used for synchronization problems that may arise in situations where microservices need to communicate with each other.

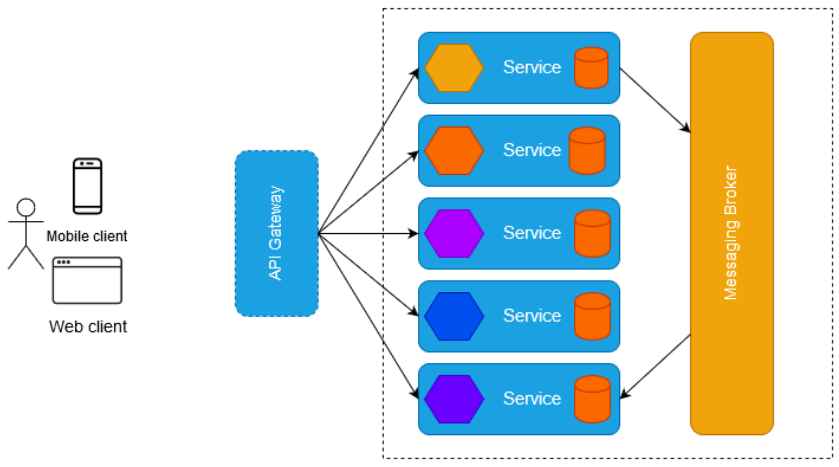


Figure 5.2 App Software Architecture Design [5]

5.2.1 API Gateway

In microservice architectures, in cases where the incoming request can go to more than one place or trigger more than one event, the direct query from the client to the microservice causes problems. For this reason, the incoming request directs the incoming request to the microservice that needs to go after the leading operations (Fraud Detection, Rate Limiting, etc.) and log records are made. API Gateway is the structure where these operations are carried out and the Load Balancing operations, which are the task of forwarding the request to other services in case of high demand for a single service, are undertaken. Since microservice architecture will be used in our application, API Gateway is a microservice like other microservices that will act as Proxy Host to direct the incoming request to the right place. In this service, NGINX Web Server and Load Balancer application will be used to act as a proxy that can forward to all other microservices.

5.2.2 Auth Microservice

It is a microservice where the necessary controls and situations are made for the user to register in the system and to continue in the system. A REST service will be designed, which accepts the information received from the user through the interface in accordance with HTTP API protocols and performs user registration, login, editing profile information, password changing, email confirmation, and communication with HTTP methods (GET, POST, PUT, DELETE). The service will be defined with node.js. JWT and hash packages will be used.

5.2.3 App Microservice

It is a microservice where actions such as adding and deleting calendars, tasks or tasks are met and notifications are made within the area we have determined within the scope of the application business model. After the integration of the relevant accounts to be integrated in the integration processes, it is planned to be abstracted by contacting the other microservice for the calendar adding processes. With Express.js, the use of all primary packages and the creation of the main structure will be performed in this microservice.

5.2.4 Database Microservice

Within the scope of the application business model, the use of NoSQL type databases in our system seems to be convenient in terms of performance, since there are user, calendar, integration, event and task objects in the UML object diagram and they frequently communicate statically with each other. For this reason, it seems convenient to use MongoDB database, which is the most suitable for our use cases among NoSQL type databases, the team is experienced and the related package named node.js and mongoose is available and the Atlas user interface offers convenience and possibilities.

5.3 Realization of Design and Containers with Docker

During the implementation of all this design, although it seems to divide the microservice architectures into more than one area and increase the code workforce, when the project grows and the business areas begin to become more specialized, it will provide very important optimizations in the independent design implementation, Unit Test processes and project size. In addition, thanks to the horizontal growth support, when the application starts to receive traffic, these microservices will be created from only that microservice despite the heavy traffic, and thanks to the balancing process with a high level Load Balancer, the need for an additional workforce will be eliminated.

It is designed to use the Docker Container service in order to implement the given designs and to do platform-independent coding. Application codes written with Docker container technology will be virtualized at the operating system level and will have their own special "/bin/" files that cannot be accessed by any user, and it is planned to solve common problems such as version mismatch, program-platform incompatibility.

6
Application

It is planned that the application will operate on a web application interface. As suggested by the Gannt diagram (see time and labor feasibility section), after the micro-service architecture is done, the user interface will be created. The connection between the front-end and back-end sides will be done with the aid of Swagger. For the front-end open-source and npm package released on GitHub called FullCalendar will be used as a template package. This template was selected for the well written documentation that proved ease of integration, and the user friendly and visually pleasing interface. It also is an npm package that is licensed under MIT and the creators have given permission to the software use free of charge, and have forfeited all right to their copyright claims, which is another reason this particular package was selected.

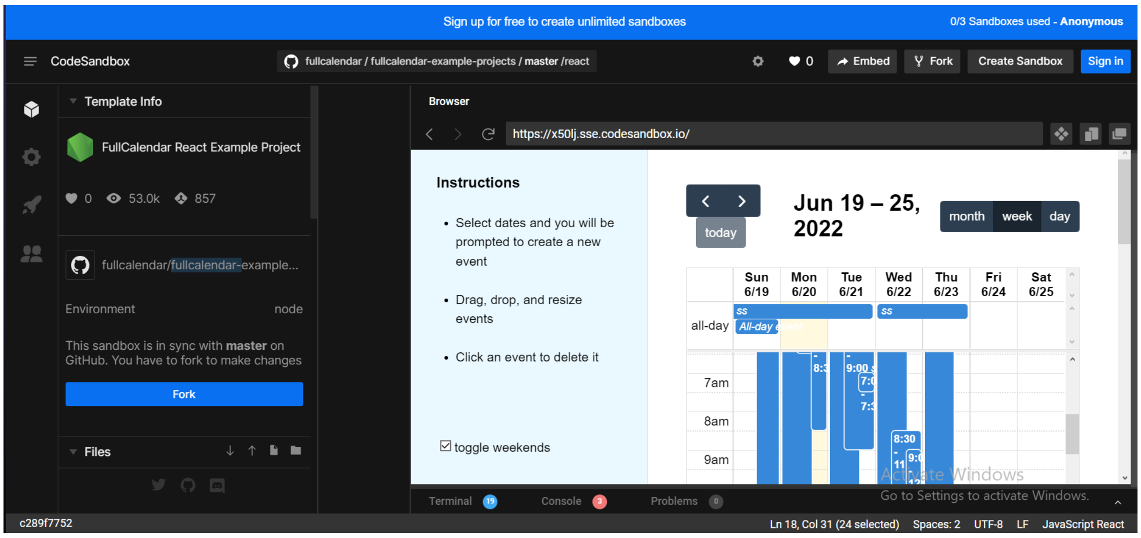
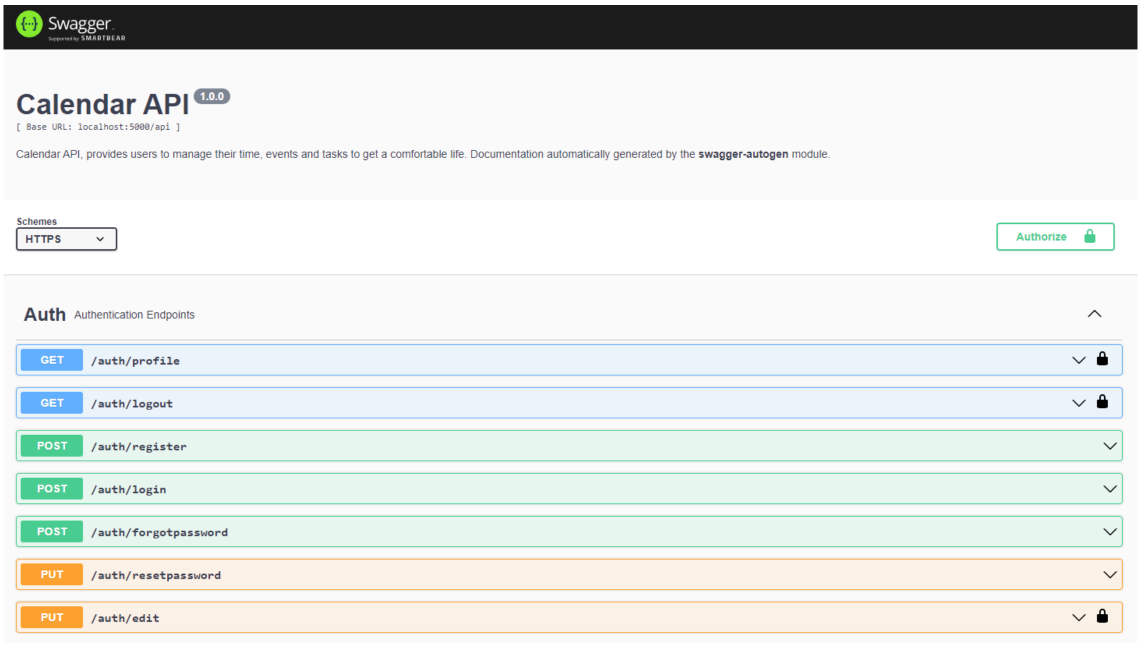

Sign up for free to create unlimited sandboxes
0/3 Sandboxes used - Anonymous
CodeSandbox
fullcalendar / fullcalendar-example-projects / master /react
Embed
Fork
Create Sandbox
Sign in
Template Info
FullCalendar React Example Project
0 53.0k 857
fullcalendar/fullcalendar-example...
Environment node
This sandbox is in sync with master on GitHub. You have to fork to make changes
Fork
Files
Browser
https://x50lj.sse.codesandbox.io/
Instructions
Select dates and you will be prompted to create a new event
Drag, drop, and resize events
Click an event to delete it
toggle weekends
today
Jun 19 – 25, 2022
month
week
day
Sun 6/19
Mon 6/20
Tue 6/21
Wed 6/22
Thu 6/23
Fri 6/24
Sat 6/25
all-day
7am
8am
9am
Terminal
Console
Problems
c289f7752
Ln 18, Col 31 (24 selected) Spaces: 2 UTF-8 LF JavaScript React

Figure 6.1 A demo of the FullCalendar package provided on CodeSandbox

6.1 Swagger

Swagger is the selected tool for establishing the communication between front-end and back-end. Given below is the explanation for the authentication end points designed for this project.


Swagger
Supported by SMARTBEAR

Calendar API 1.0.0
[Base URL: localhost:5000/api]

Calendar API, provides users to manage their time, events and tasks to get a comfortable life. Documentation automatically generated by the **swagger-autogen** module.

Schemes
HTTPS

Authorize

Auth Authentication Endpoints

GET /auth/profile
GET /auth/logout
POST /auth/register
POST /auth/login
POST /auth/forgotpassword
PUT /auth/resetpassword
PUT /auth/edit

Figure 6.2 Swagger Interface and Authentication End Points

/profile : The endpoint that returns the user profile info.

/logout : Endpoint for a logged in user to send requests for logout and controls of said operation.

/register : Endpoint that handles requests for a user that wants to register, and responses when registration state is acceptable.

/login : Endpoint that receives info from the user for login operations and manages authentications for the operation.

/forgotpassword : Endpoint to handle request for users that forget their passwords, and send out the reset mails.

/resetpassword : Endpoint that handles new password info after the previously mentioned mail is sent, and also manages sending password resetting or error messaging responses.

/edit : Endpoint that handles operations for a logged in user that wants to update their own profile info.

6.2 Frontend

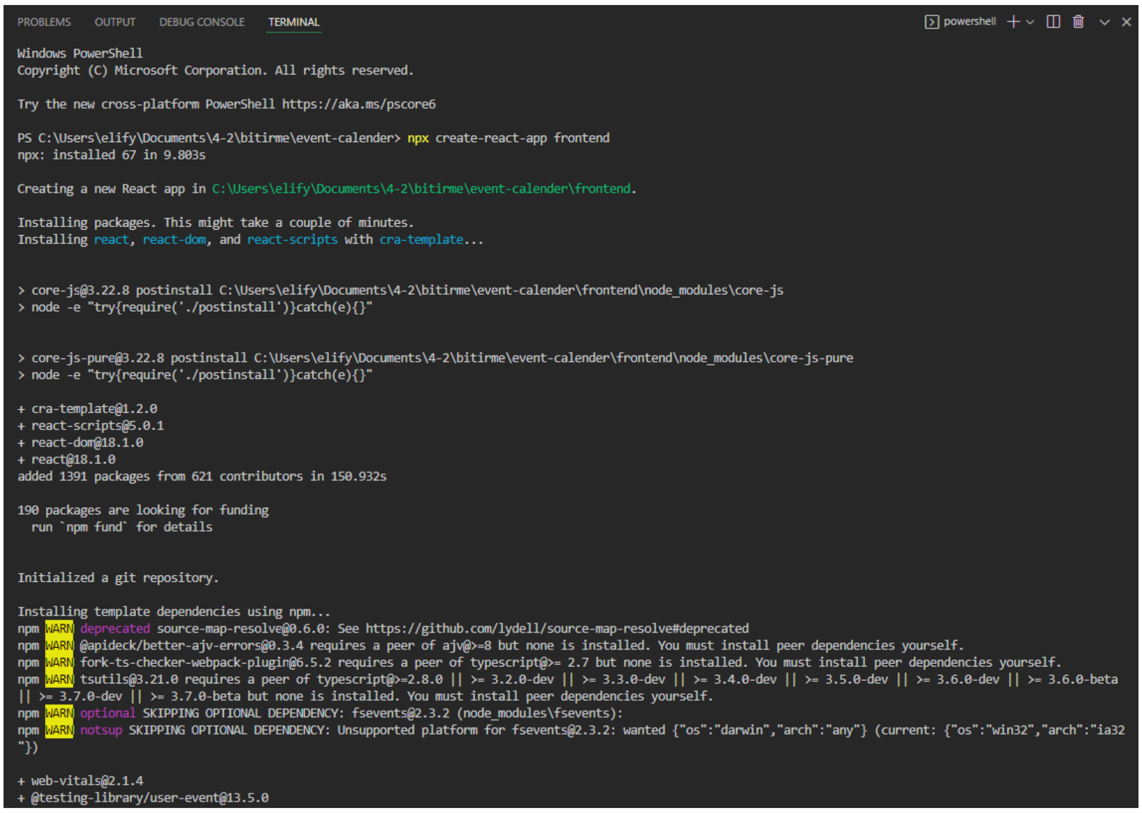
6.2.1 Build

Development environment selected to operate on is VS Code, which was selected for the many helpful extension it provides for web developers (see Figure 6.4 below).

React framework is being used in this part of the project, which is not what tui.Calendar was initially designed for. It was designed to operate on plain ES6 JavaScript. This problem was solved by using a version that is compatible with React by using a bunch of wrappers provided in the documentation.

This is done by organizing the project files into two parts: front-end and back-end. Essentially, this type of architecture is actually two different application communicating with each other through an API (which is the Swagger explained above), and should be treated as separately.

Firstly a react application is created automatically through a command: ***npx* create-react-app frontend** (Figure 6.3)



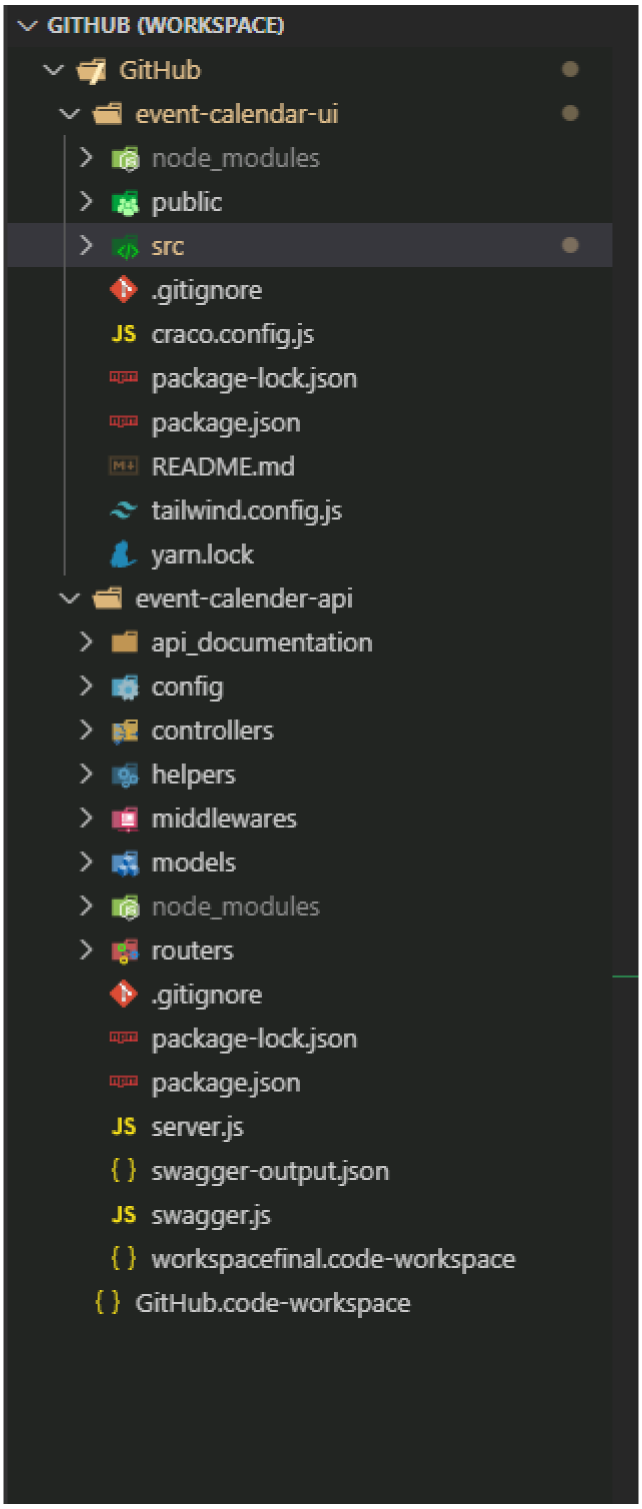
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\elify\Documents\4-2\bitirme\event-calender> npx create-react-app frontend
npx: installed 67 in 9.803s

Creating a new React app in C:\Users\elify\Documents\4-2\bitirme\event-calender\frontend.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...


> core-js@3.22.8 postinstall C:\Users\elify\Documents\4-2\bitirme\event-calender\frontend\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"


> core-js-pure@3.22.8 postinstall C:\Users\elify\Documents\4-2\bitirme\event-calender\frontend\node_modules\core-js-pure
> node -e "try{require('./postinstall')}catch(e){}"

+ cra-template@1.2.0
+ react-scripts@5.0.1
+ react-dom@18.1.0
+ react@18.1.0
added 1391 packages from 621 contributors in 150.932s

190 packages are looking for funding
  run `npm fund` for details


Initialized a git repository.

Installing template dependencies using npm...
npm WARN deprecated source-map-resolve@0.6.0: See https://github.com/lydell/source-map-resolve#deprecated
npm WARN @apideck/better-ajv-errors@0.3.4 requires a peer of ajv@>=8 but none is installed. You must install peer dependencies yourself.
npm WARN fork-ts-checker-webpack-plugin@6.5.2 requires a peer of typescript@>= 2.7 but none is installed. You must install peer dependencies yourself.
npm WARN tsutils@3.21.0 requires a peer of typescript@>=2.8.0 || >= 3.2.0-dev || >= 3.3.0-dev || >= 3.4.0-dev || >= 3.5.0-dev || >= 3.6.0-dev || >= 3.6.0-beta
|| >= 3.7.0-dev || >= 3.7.0-beta but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"ia32
"})

+ web-vitals@2.1.4
+ @testing-library/user-event@13.5.0
```

Figure 6.3 Creating a React App through Terminal Commands

This is the front-end side of the project so it is named as such. Next step is to install the npm packages related to tui.Calendar with the command taken directly form the documentation: ***npm* install –save @fullcalendar/react @fullcalendar/daygrid**

Next step would be to edit the file structure of the frontend and add React routers. This ensures that pages can be switched between Login, Sign Up and Calendar.

After the preparations are done, figure 6.4 demonstrates what can be seen on the development environment.



```
GITHUB (WORKSPACE)
  GitHub
    event-calendar-ui
      node_modules
      public
      src
      .gitignore
      craco.config.js
      package-lock.json
      package.json
      README.md
      tailwind.config.js
      yarn.lock
    event-calender-api
      api_documentation
      config
      controllers
      helpers
      middlewares
      models
      node_modules
      routers
      .gitignore
      package-lock.json
      package.json
      server.js
      swagger-output.json
      swagger.js
      workspacefinal.code-workspace
    GitHub.code-workspace
```

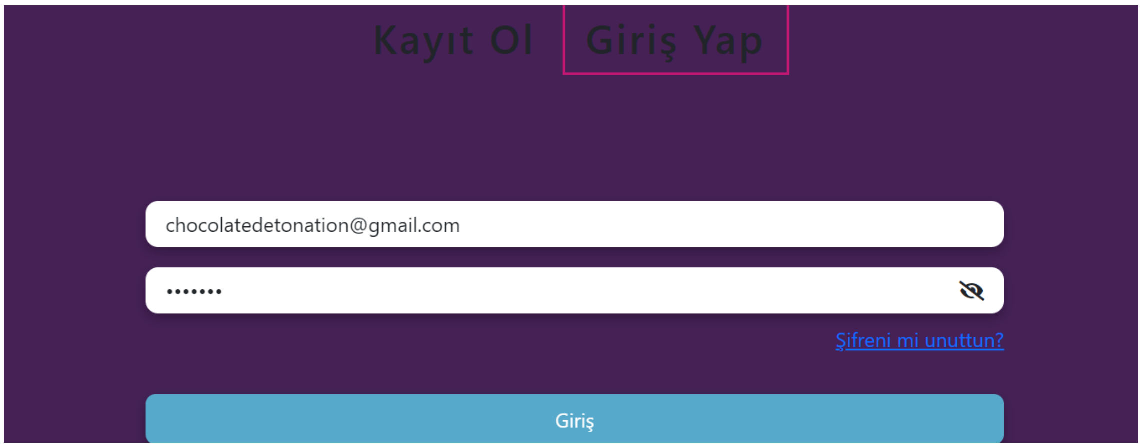
Figure 6.4 File Structure

From this point forward certain changes are made to the files React automatically created, mainly *App.js* that is the app component and *index.js* which is responsible for the rendering logic and entry point of the application.

6.2.2 UI

In this section a brief walk through of the interface will be conducted.

The application starts with the login page (Figure 6.5).

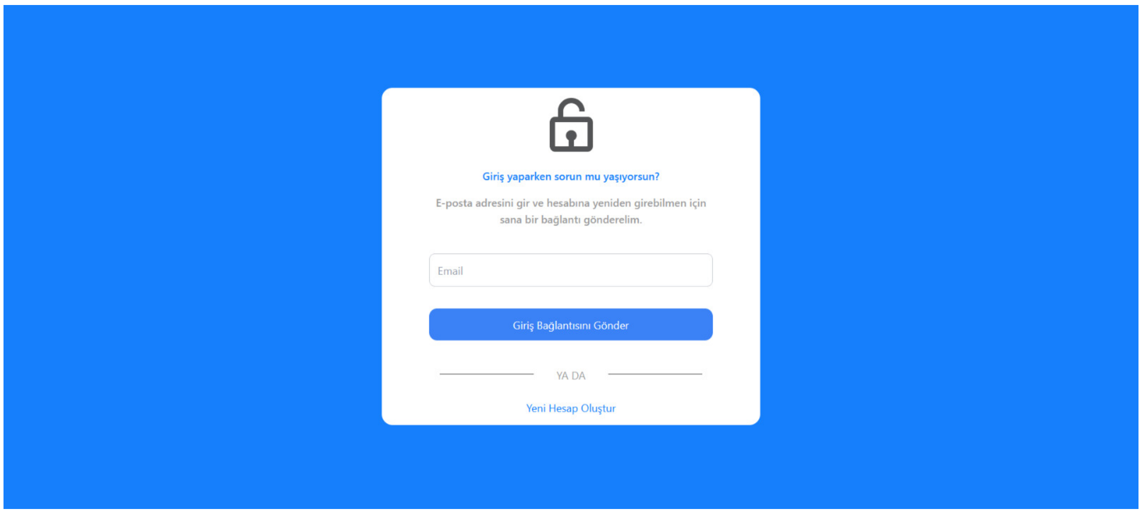


chocolatedetonation@gmail.com

•••••••

Figure 6.5 Login Page

Users can click the register button on the page above to jump to the register page for creating new accounts. There is also another page for reminding forgotten passwords (Figure 6.6).



Giriş yaparken sorun mu yaşıyorsun?

E-posta adresini gir ve hesabına yeniden girebilmen için sana bir bağlantı gönderelim.

Figure 6.6 Forgot Password Page

The calendar is the react application part (Figure 6.7). This page is re-rendered continuously with every update, and is responsive with drag and droppable events.

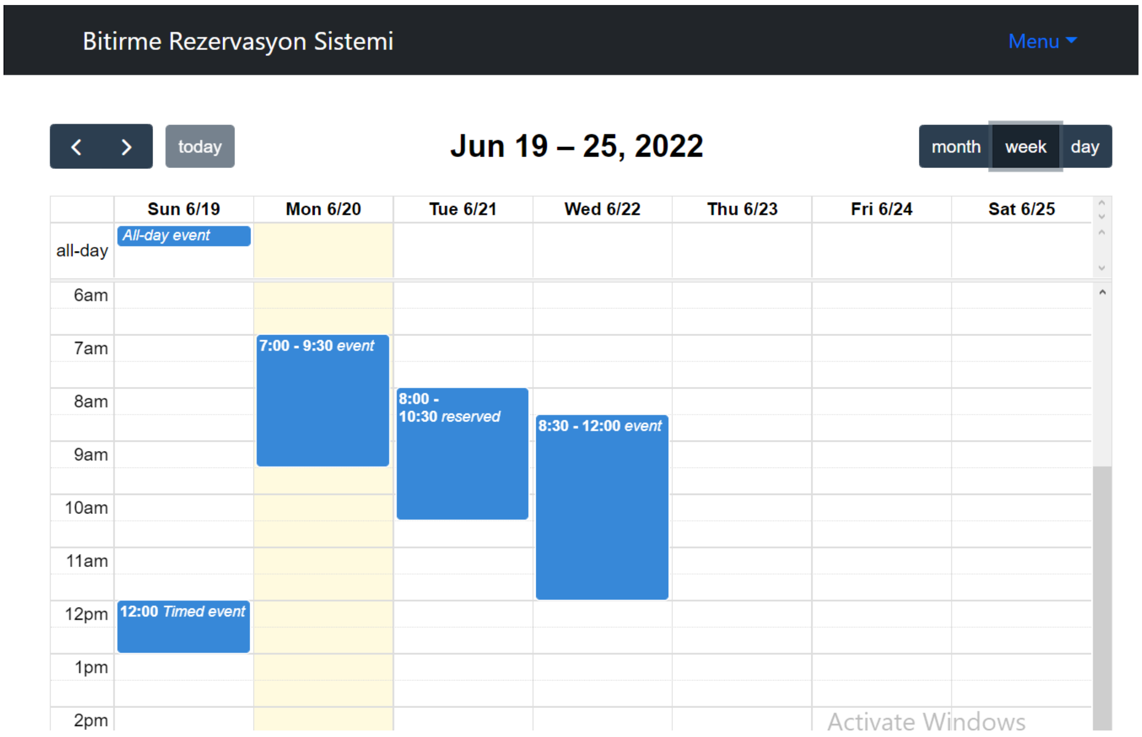


Figure 6.7 Calendar Page, Weekly Look

7 Conclusion

In this study a project that connects different calendar application together in an independent interface in order to view them together was developed and deployed live. In order to realize this project, we conducted research about different productivity systems. The project was intended operate on the web environment from the start. We learned about the microservice architecture, and familiarized ourselves with the React framework during the preliminary process.

When starting the implementing process, a relational model was created for deciding the features. Then the issue of data storage arose, which resulted in the database system being designed. For the data storing SQL type databases were deemed appropriate and it was decided that NoSQL would be used.

For the issue of software architecture a study on monolith and microservice type designs was conducted and in the end, microservice architecture was selected for the project. At this point in the project, the Docker Container Service came to our aid, in order to virtualize the application codes and organize the roles of access.

And lastly it was time to create the front-end application. An open-source and free to use npm package called FullCalendar was selected for use here. This was installed and integrated into the project with some modifications on our side. Lastly Swagger API was used for communication between the back-end and front-end sides.

References

[1] D. Allen, *Getting things done: The art of stress-free productivity*, 2001.

[2] T. Schwartz. “Manage your energy, not your time.” (2007), [Online]. Available: https://hbr.org/2007/10/manage-your-energy-not-your-time.

[3] NPM. “Npm open source policies.” (), [Online]. Available: https://docs.npmjs.com/policies/open-source-terms.

[4] A. Barashkov. “Microservices vs. monolith architecture.” (2018), [Online]. Available: https://dev.to/alex_barashkov/microservices-vs-monolith-architecture-4l1m/.

[5] F. Taleb. “Microservices architecture: To be or not to be.” (2019), [Online]. Available: https://feras.blog/microservices-architecture-to-be-or-not-to-be/.

Curriculum Vitae

FIRST MEMBER

Name-Surname: Ali Barış Zengin
Birthdate and Place of Birth: 15.05.1999, İstanbul
E-mail: l1117081@std.yildiz.edu.tr
Phone: 0531 029 15 03
Practical Training: Logo Yazılım Stajyeri

SECOND MEMBER

Name-Surname: Elif Yağmur Duran
Birthdate and Place of Birth: 26.07.1999 İstanbul, Türkiye
E-mail: elif.duran@std.yildiz.edu.tr
Phone: 0537 820 70 66
Practical Training: Experian Limited Şirketi, Analiz Departmanı

Project System Informations

System and Software: Container(Docker), Node.js, MongoDB, Redis, NATS, CI/CD, Javascript
Required RAM: 4GB
Required Disk: 50GB