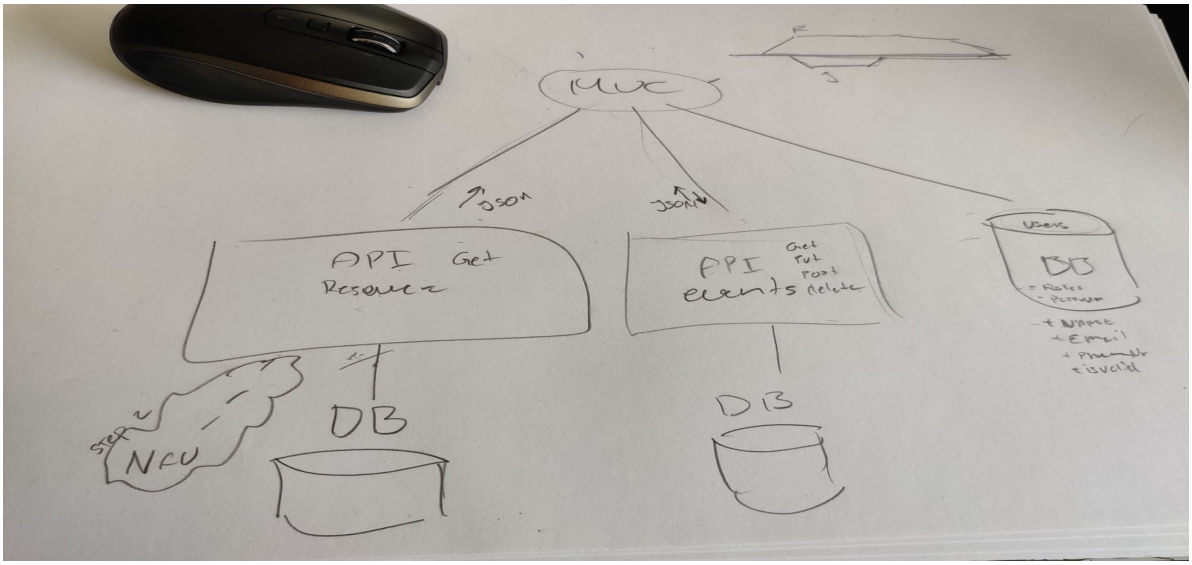
3. semesterprøven - del 1

Programmering 2 og Teknologi 2

Arkitektur - MVC og Web API

[Forfatter: Ragnar]



Vores program består af en .Net MVC løsning, som står for alt UI kaldet "Delpinbooking", under Delpinbooking har vi en database på azure som håndterer kunder, såvel som ansatte. Delpinbooking snakker sammen med 2 API'er. En til Ressourcer (ResourceAPI) og en som håndterer events (EventAPI). Disse to har hver især deres egen database som ligger på Azure.

Ved at opbygge vores program med den lave kobling som det har, ender vi med at have meget nemt ved at udskifte eventuelt API'er eller tilføje andre API'er, et eksempel vil være at vi bruger en API til at vise alle adresser i Danmark Kaldet DAWA/dataforsyningen[1]. Andre gode grunde til at have denne løse kobling er at, databaserne til programmet også let kan skiftes ud, eller ændres i uden at det påvirker hoved løsningen. Der vil være rig mulighed med den valgte arkitektur at skifte databaserne til api'erne så man i fremtiden kan bruge vores løsning til.

[1] https://dawadocs.dataforsyningen.dk/dok/api/adgangsadresse#autocomplete

Client - JavaScript, HTML og CSS

[forfatter: Jesper]

Som en del af vores MVC har vi udviklet vores frontend med HTML og JavaScript. Vi havde forskellige løsninger oppe omkring hvordan vi visuelt kunne præsentere kalenderen bedst muligt i forhold til at skabe brugervenlighed og overblik for DelPin. Efter at have sorteret flere løsninger fra stod vi til sidst mellem to JavaScript libraries fra henholdsvis DayPilot og FullCalender. Valget faldt på sidstnævnte, da det virkede en anelse mere moderne i udtrykket.

Måden vi har integreret det på er via et div-element[2] som renderes ud fra en lang række parametre. Dette inklusiv et par Json arrays der bliver kaldt fra vores Controller med Ressourcer og Events (bookinger)[3].

Vi har også brugt JavaScript (jQuery) til at tilføje funktionalitet til vores kalender. Herunder kalde flere forskellige Modal-elementer[4], hvilket er måden vi har valgt at bygge client delen på.

Vores Modal-elementer er derefter bygget op via ASP.NET hvilket giver os mulighed for at lave modelvalidering på client-siden inden der sendes eksempelvis POST requests til server.

Til validering af adresser har vi integreret en autocomplete Web API der stilles til rådighed fra DAWA/Dataforsyningen[5].

Til styling har vi brugt Bootstrap Library[6].

REST API

[Forfatter: Marc]

Vi har implementeret to REST API'er i vores projekt, ResourceAPI som står for at håndtere vores ressourcer og EventAPI som står for vores events. Hver API har deres egen database. Hvor både database og API'erne er deployed til Azure.

I vores ressource controller har vi kun en GET metode, da vores kalender

[2] ~Views/Calendar/index.cshtml Line 96
[3] ~wwwroot/js/Calendar.js Line 61 / 62
[4] ~Views/Calendar/index.cshtml Line 98 som eksempel på Modal til oprettelse af nyt event.
[5] https://dawadocs.dataforsyningen.dk/dok/api/adgangsadresse#autocomplete
~wwwroot/js/calendar.js Line 204
[6] ~Views/Shared/_Layout.cshtml Line 71

kun har brug for at hente ressourcer og kategorier som ligger på API'en.

I vores CalendarController(eventcontroller) har vi alle fire metoder, altså GET, POST, PUT og DELETE da disse alle er nogle vi har brug for i vores kalenders funktionalitet.

At vi bruger disse api'er giver vores arkitektur et ekstra lag og lavere kobling, samtidigt gør det system nemmere at vedligeholde og udskifte. F.eks. i forhold til at implementere navision som var et ønske, ville det kun være i ResourceAPI'en der skulle udskiftes.

Vi har udført tests på API'erne via postman hvor vi har testet nogle af de forskellige metoder for at se om de returnere korrekt status kode og tabeller, se bilag for billeder.

Sikkerhed

[Forfatter: Marc]

Authorization og Authentication

Vi bruger authorization og authentication til at styrer hvilke brugerroller og at det kun er brugere der er logget ind der kan tilgå de forskellige funktioner på siden. F.eks. kan medarbejdere kun tilgå kalenderen, hvor en customer ikke kan tilgå kalenderen, men derimod se deres bookinger. Et eksempel på hvor vi bruger Authorize er i vores UserController hvor den er sat til at kun admins kan tilgå den, da det er en admin funktion.

Derudover har vi også benyttet os af ValidateAntiForgeryToken på vores CalendarController på alle metoder der ikke er GET, for at undgå at der kan laves cross-site request forgery som gør at der ikke kan sendes ondsindede requests fra en ikke betroet bruger.

Sikring af API'er

Vi har sikret vores API endpoints via Azure, der har vi sat det op så den eneste der kan tilgå dem er ip'en fra vores side, hvis du f.eks via postman prøver at lave en POST vil du få en error 403 som betyder forbidden. Vi havde oprindeligt tænkt os at lukke dem via docker compose eller bruge bearer tokens til det. Begge dele viste sig dog at være en stor og tidskrævende opgave som vi valgte at prioritere fra grundet at vi ikke havde overskydende tid og at vi føler vores løsning giver os nok ro i sindet i forhold til at endpoints er sikret.

Concurrency

[forfatter: Jesper]

Vi har arbejdet med concurrency på vores EventAPI, da vores applikation selvfølgelig understøtter flere brugere samtidig. Vi arbejder ud fra optimistic concurrency, hvor vi egentlig tillader at der må opstå konflikter, men derfra forsøger at løse dem på bedst mulig måde.

På vores model lag/database har vi konkret implementeret det via en tracking property (RowVersion) på vores Event[7]. Vores PUT method på Controlleren *catcher* herefter en evt DbConcurrencyException og returnerer en 409 Conflict. Herfra mødte vi lidt modstand i den videre implementering i vores MVC applikation.

Da vi har bygget vores View op via Modal-elementer, vil vi helst undgå at blive redirected til anden side i forbindelse med en conflict. Men omvendt vil vi gerne give brugeren besked om at der er opstået en conflict. Det er ikke lykkedes med en løsning der giver brugeren en besked, men i forbindelse med conflict kan sidste bruger ikke gennemføre sin opdatering (bliver i #EditModal)[8].

Det er ikke den mest optimale løsning, men vi undgår eventuelle lost updates og alle bookinger bliver refetched så snart Modal-elementer bliver lukket. Herefter kan en opdatering forsøges igen og gennemføres.

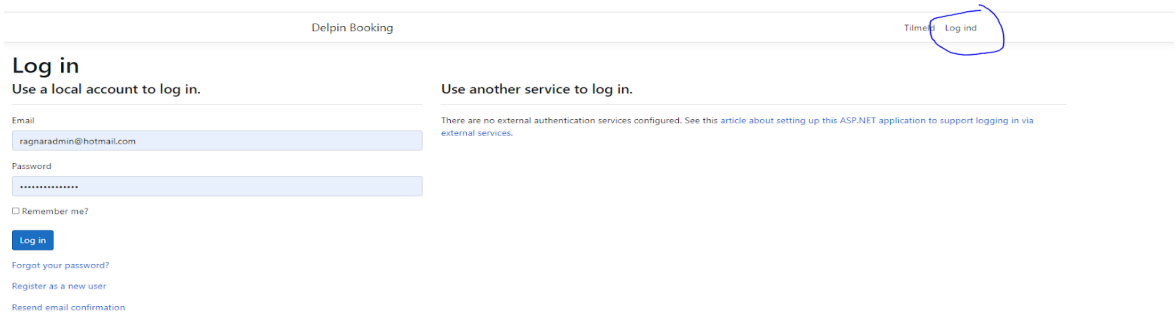
Use case 1 gennemgang

[Forfatter: Ragnar]

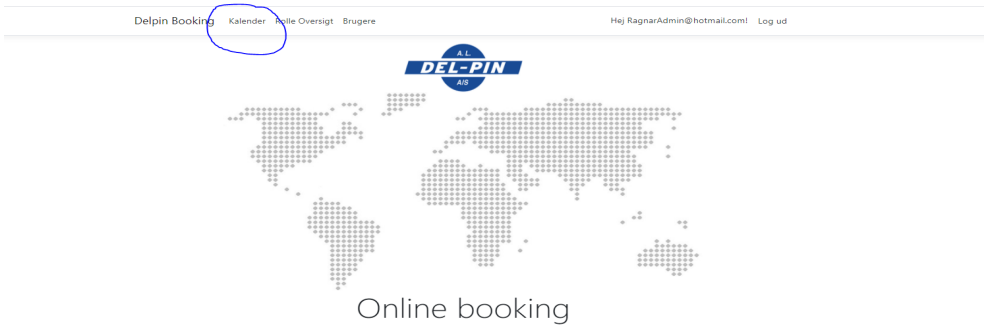
Use case 1: Se aktuelle bookinger, udført af medarbejder.[9]

Medarbejder kommer ind på siden: https://delpinbookingv2.azurewebsites.net/ skal så indtaste det korrekte login/password.

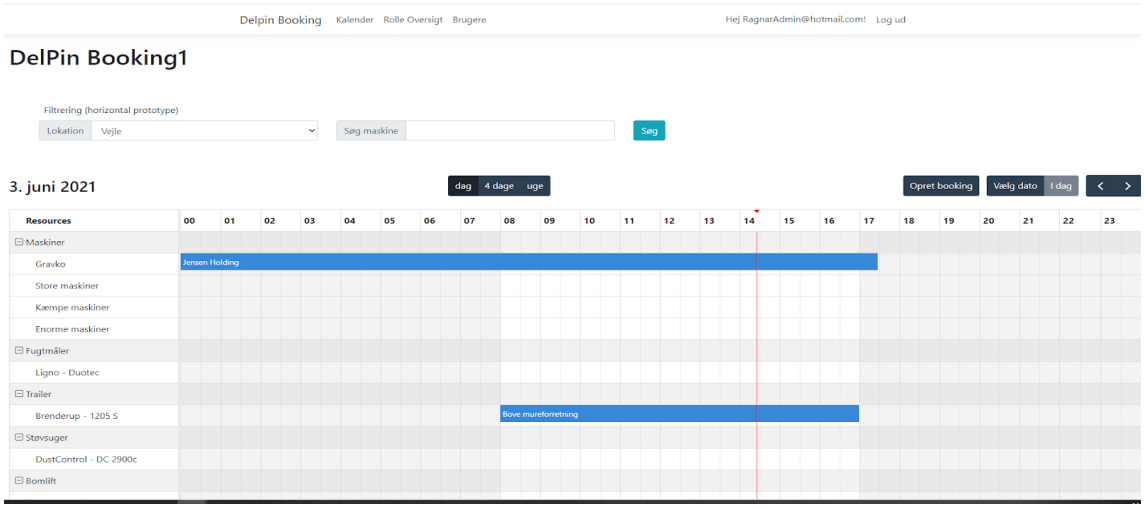
[7] EventAPi ~Model/Event.cs Line 20
[8] MVC ~Controllers/CalendarController Line 86. Returnere NoContent( ) hvis IsSuccesStatusCode er false.
[9] Se bilag 4 Use case 1.



Herefter vælges der kalender



som starter et render af kalenderen på clienten, og fetcher Ressourcer og events så vedkommende kan se hvilke events, ressourcer / subressourcer der er hos Delpin.



Disse 2 kald udføres af 2 seperate API's[10] derefter kan der redigeres i hver enkelt booking som ønsket, lokation adresse og lign.

[10] Se bilag 1,2 og 3 af henholdsvis Get resources, Get events og Post events.

Deploy

Azure

Vi har hosted programmet i en live udgave på Azures App Service.

Link til live version
https://delpinbookingv2.azurewebsites.net/

Logins
testercustomer@mail.com
testeremployee@mail.com
testeradmin@mail.com

Password
Azure1234!

Docker Hub

Derudover har vi lagt en af vores API ud på Docker Hub og kan tilgås fra nedenstående link
https://hub.docker.com/r/marc3411/eventapi

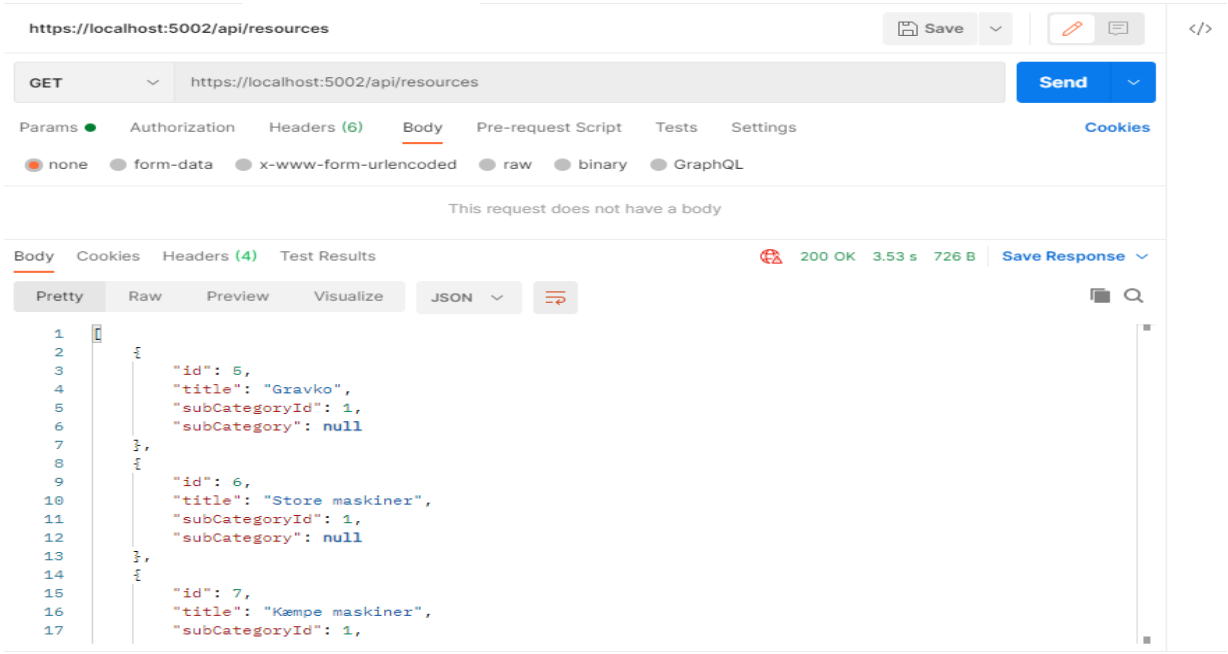
eller laves som pull [docker pull marc3411/eventapi]

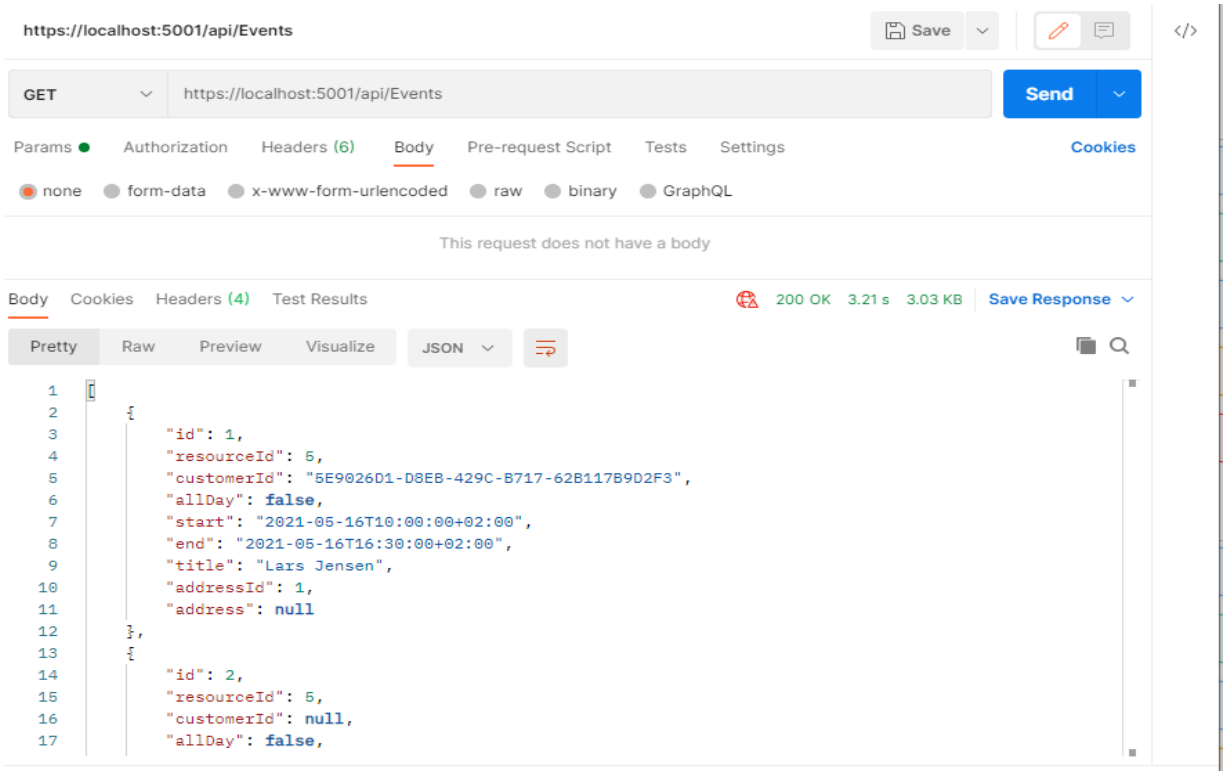
GitHub Repositories

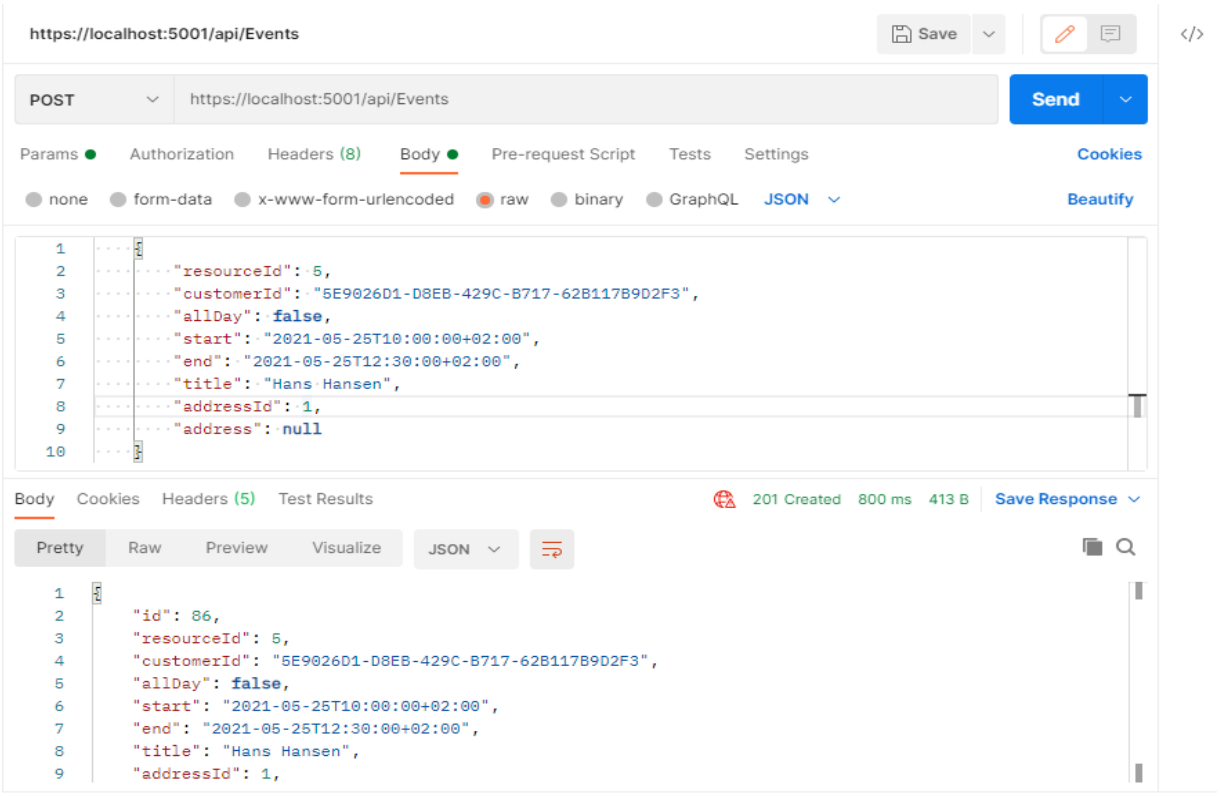
https://github.com/JesperlaCour/BookingSystem

Bilag

Test af api'er via postman:






https://localhost:5001/api/Events
Save
POST
https://localhost:5001/api/Events
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"resourceId": 5,
"customerId": "5E9026D1-D8EB-429C-B717-62B117B9D2F3",
"allDay": false,
"start": "2021-05-25T10:00:00+02:00",
"end": "2021-05-25T12:30:00+02:00",
"title": "Hans Hansen",
"addressId": 1,
"address": null
}
Body
Cookies
Headers (5)
Test Results
201 Created
800 ms
413 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 86,
"resourceId": 5,
"customerId": "5E9026D1-D8EB-429C-B717-62B117B9D2F3",
"allDay": false,
"start": "2021-05-25T10:00:00+02:00",
"end": "2021-05-25T12:30:00+02:00",
"title": "Hans Hansen",
"addressId": 1,

Use Cases

UC1 Se aktuelle bookinger

Actor: Medarbejder
Pre-conditions:
Der er oprettede bookinger som kan vises og medarbejder er logget ind med medarbejderrettigheder.
Main success scenario:

1. Medarbejder logger ind
2. Vælger herefter kalender som viser aktuelle dag og en oversigt med alle ressourcer
3. Medarbejder filtrer på kategorier, lokation og lignende.

Post condition:
Der vises aktuelle bookinger og medarbejderen har mulighed for at redigere i dem.