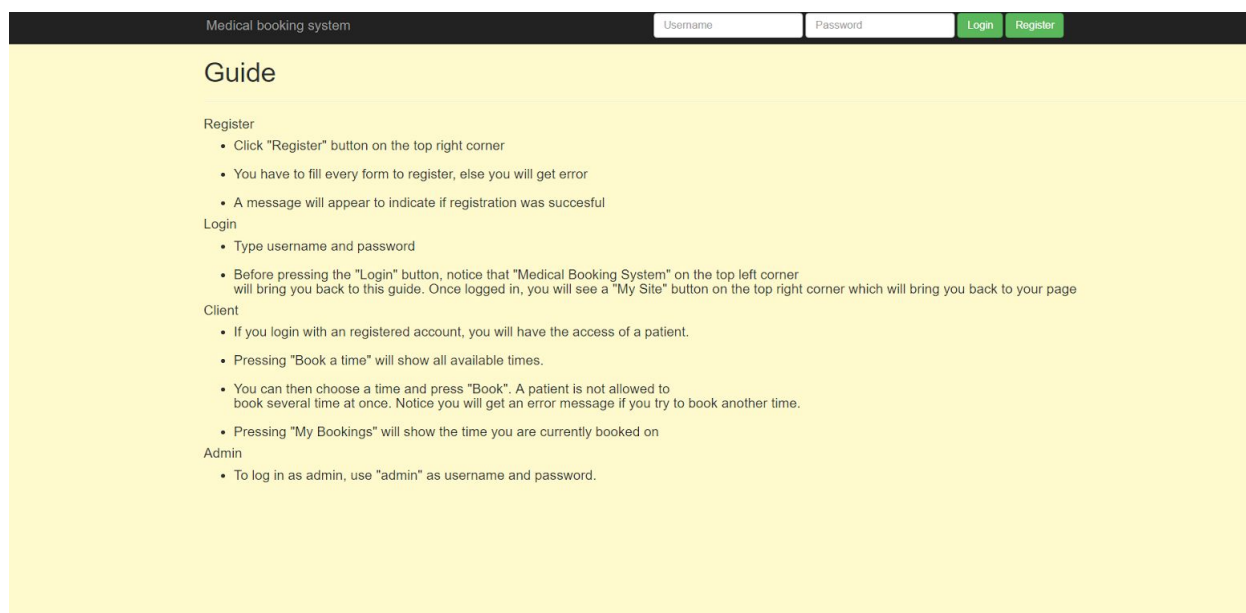


Inledning

Webb-applikationen som har utvecklats under denna projektkurs är avsedd till en tandläkare som snart skall starta en egen klinik. Applikationen är tänkt att fungera som ett bokningssystem där patienter kan registrera sig och boka tid för läkarbesök. Vi vill konstatera att hemsidan inte är fullständig utvecklad ännu utan det kommer byggas vidare på såväl det grafiska gränssnittet som de tekniska funktionerna såsom generera schema vid bokning, hantering av transaktioner, etc.

Flödet av applikationen

I denna del beskrivs applikationen på ett övergripande sätt och relevanta delar illustreras med bilder. Första sidan av applikationen innehåller en guide som förklarar de olika stegen i applikationen. Detta ansågs vara nödvändigt för att underlätta navigeringen mellan de olika delarna när en användare testar applikationen för första gången. Första sidan innehåller även två fält för användarnamn respektive lösenord, en knapp för inloggning och en för registrering. Nedan illustreras första sidan med bild.



Figur 1: Startsidan

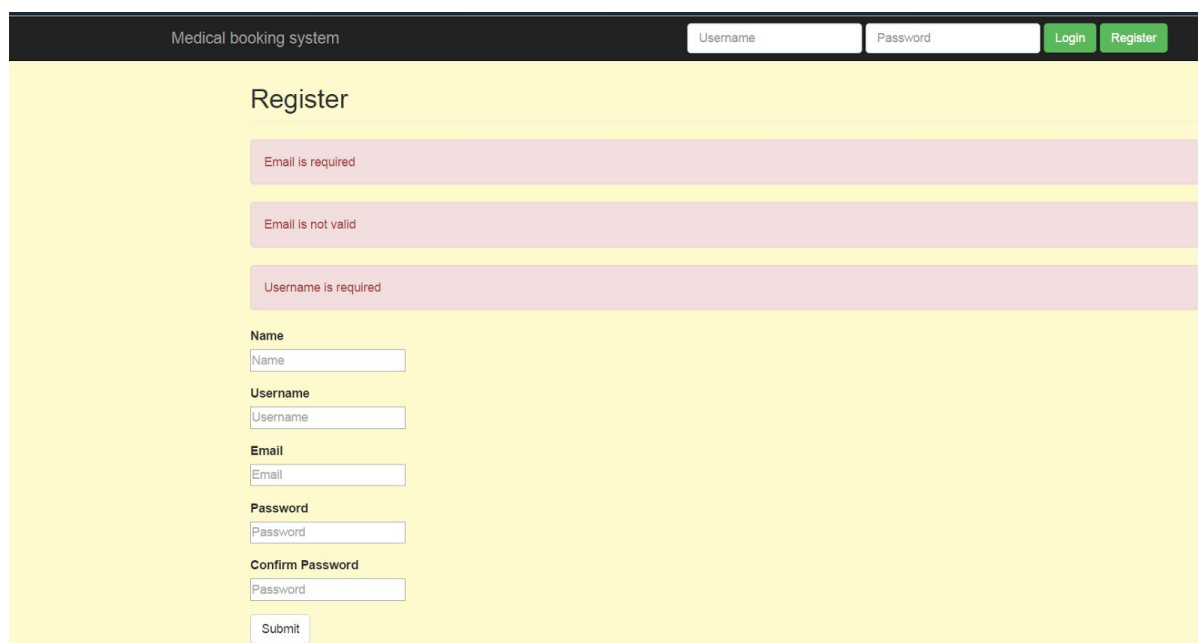
Vid registrering vidarebefodras användaren till följande sida.



The screenshot shows the 'Medical booking system' header with a dark background. On the right, there are input fields for 'Username' and 'Password', and two green buttons labeled 'Login' and 'Register'. The main content area has a light yellow background and is titled 'Register'. It contains a vertical stack of form fields: 'Name' (with a placeholder 'Name'), 'Username' (with a placeholder 'Username'), 'Email' (with a placeholder 'Email'), 'Password' (with a placeholder 'Password'), and 'Confirm Password' (with a placeholder 'Password'). At the bottom of the form is a 'Submit' button.

Figur 2: Registreringssida

Om något fält inte är korrekt ifyllt vid registrering är bilden nedan en möjlig scenario.



This screenshot shows the same 'Register' page as Figure 2, but with validation errors. Three red error messages are displayed in pink boxes above the form fields: 'Email is required', 'Email is not valid', and 'Username is required'. The form fields and the 'Submit' button remain visible below the error messages.

Figur 3: Varning visas på registreringssidan

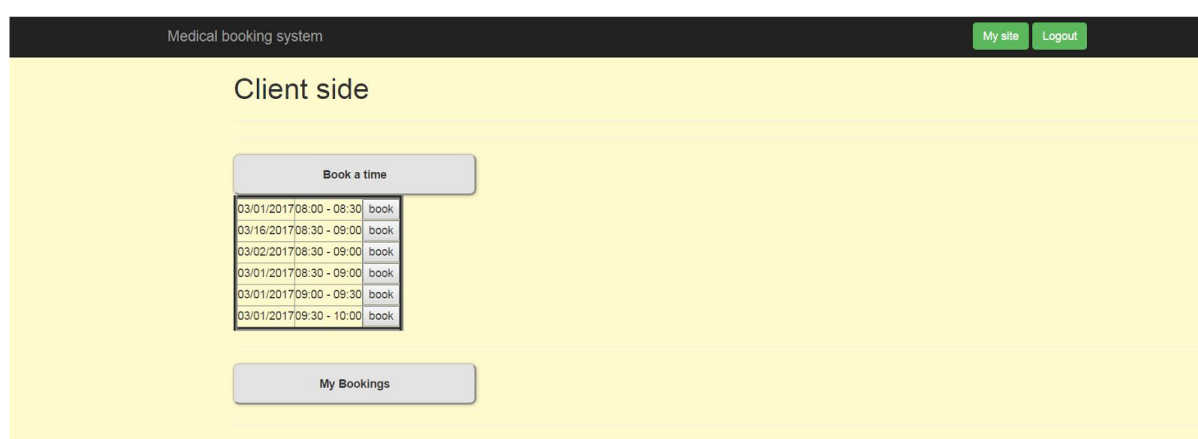
Om alla fälten i registreringssidan är korrekt ifyllda och användaren (användarnamn och e-mail) inte redan finns registrerad i vår databas visas ett meddelande om att registreringen lyckades. Applikationen återgår sedan till den första sidan. Det anses inte vara relevant att infoga bild på detta eftersom det ser nästintill likadan ut som förra bilden, skillnaden är bara att meddelandet som visas är annorlunda.

Steget efter registrering är uppenbarligen inloggning. Det är förutbestämt att en registrerad användare skall anses vara en patient och nedan visas sidan för patienten.



Figur 4: startsidan för patienten

Om användaren trycker på “Medical booking system” som finns uppe i vänstra hörnet återgår applikationen till första sidan (utan att loggas ut). Användaren kan sedan återkomma till sidan som visas i bilden ovan genom att trycka på knappen “My site” som finns längst upp på högra hörnet. Vid knapptryckning på “Book a time” visas en tabell för användaren med lediga tider. Användaren kan sedan trycka på “Book” för att boka en tid. Följaktligen kommer den nybokade tiden markeras som “booked” på databasen och därmed kommer denna tid inte längre vara tillgänglig för någon annan användare. När en användare trycker på “Book a time” ser hemsidan ut på följande sätt.



figur 5: boka en tid

En patient kan även trycka på “Show Bookings” som visar bokningar för en specifik patient och här får användaren även möjligheten att avboka en tidigare bokad tid. För tillfället kan en patienten okontrollerad avboka en tid men i framtiden skall det implementeras funktioner som kontrollerar att en patient inte kan avboka en tid om det enbart är ett visst antal dagar kvar till läkarbesöket.



figur 6: visa mina bokningar

Web-applikationen innehåller även en sida för läkaren (admin). En läkarare skall inte registrera sig som patienter utan läkaren blir snarare tilldelat ett konto. Just nu sker denna tilldelning från databasen där en attribut för användare (isAdmin) sätts till true. Första sidan till admin ser ut på följande vis.



figur 7: startsidan för admin/läkaren

Första knappen på admin sidan är “Show and edit bookings” och precis som namnet tyder visas alla bokningar och detta innefattar såväl lediga som upptagna bokningar. Admin får även möjligheten att editera en bokning och trycka på “update”. Den nyinskrivna tiden/datumet lagras i databasen och visas upp för patienten när denne trycker på “Show bookings”.

Date	Start time	End time	Status	Actions
04/20/2017	13:00	13:30	Unbooked	<button>update</button> <button>delete</button>
04/20/2017	13:30	14:00	Unbooked	<button>update</button> <button>delete</button>
04/20/2017	14:00	14:30	Unbooked	<button>update</button> <button>delete</button>
04/20/2017	14:30	15:00	Unbooked	<button>update</button> <button>delete</button>
04/20/2017	15:00	15:30	Unbooked	<button>update</button> <button>delete</button>
04/20/2017	15:30	16:00	Unbooked	<button>update</button> <button>delete</button>

figur 8: visar innehåll av ‘Show and edit bookings’

Den andra knappen på admin sidan är “Create a new booking” och även här avslöjar namnet vad som skall hända vid tryckning på denna knapp.

Show and edit bookings

Hide bookings

Date
03/09/2017

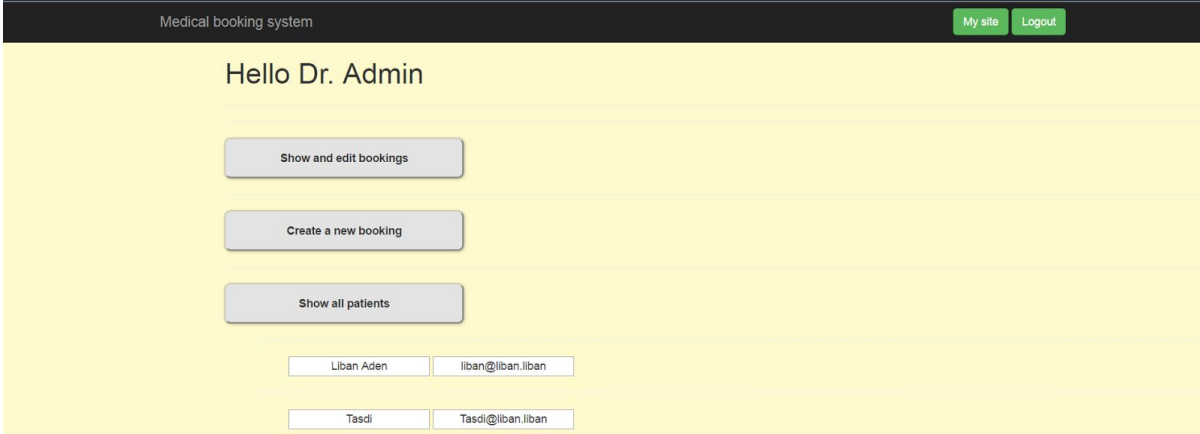
Start time
10:30

End time
endTime
12:00
12:30
13:00
13:30
14:00

ents

figur 9: skapa en ny bookningstid

Tredje knappen visar namn och email på alla patienter. Det är tänkt att vi senare skall implementera så att en läkaren kan trycka på en viss patients email-address och skicka ett meddelande till denna patient om så skulle önskas.



The screenshot shows a web application interface for a medical booking system. At the top, there is a dark header bar with the text "Medical booking system" on the left and two green buttons labeled "My site" and "Logout" on the right. Below the header, the main content area has a light yellow background. It starts with the greeting "Hello Dr. Admin". There are three stacked buttons: "Show and edit bookings", "Create a new booking", and "Show all patients". Below these buttons, there are two rows of text input fields. The first row contains "Liban Aden" and "liban@liban.liban". The second row contains "Tasdi" and "Tasdi@liban.liban".

figur 10: visa alla patienter

Applikationen i “Top-down approach”

Klient

Denna applikation använder Handlebars filer för att hantera dess templates. Handlebar är extension av mustache template och är väldigt användbart eftersom den använder olika hjälpmedel som inte finns med i en vanlig html fil. Exempel på detta är:

- expression: För att visa upp data på html format
- Helpers: Handlebar använder olika typer av helpers för att kunna hantera data på ett bättre sätt som t.e.x `#each` för att loopa igenom någon lista.

Server

Denna applikation är uppbyggd med node som har express som ramverk. Express ansågs vara lämpligt i detta projekt eftersom det har en routningstabell som tillåter navigering mellan olika sidor i applikationen och tillåter hantering av alla middleware som har använts i denna applikation.

Middleware

Denna del kommer att handla om de olika middleware som har använts i denna Web-applikation.

Bcrypt

För säkerhetsåtgärder använder vi Bcrypt som tillåter kryptering av lösenord när en klient registrerar sig och lagras i vår databas.

Passport

För vår applikation är det viktigt att en användare inte kan komma åt en annan användares sida genom att t.e.x. manipulera url. För detta ändamål används Passport som erbjuder tjänster för autentisering. I vår applikation använder vi även ‘serializeUser’ och ‘deserializeUser’. Den förstnämnda metoden används för att bestämma vad för data som skall lagras för en specifik användare under en viss session. Den sistnämnda är motsatsen, hämtar rådatan och konstruerar den enligt user modellen.

BodyParser

Detta middleware gör det lätt att hantera post request eftersom det hela tiden tar emot förfrågan och lagrar det i req.body.

ExpressValidator

Vi anser att detta middleware är relevant för detta projekt eftersom den erbjuder funktioner som validerar parametrar, förfrågan, headers, cookies och body. I detta projekt så användes funktionerna:

- isEmail: För att kolla att om inskriven email är giltigt.
- notEmpty: För att kontrollera att en nödvändig fält (t.ex. användarnamn, lösenord och email) inte är tomt.
- equal: För att se till att vissa fält (t.e.x. lösenord vid registrering) matchar varandra.

ExpressSession & CookieParser

HTTP är tillståndslös, så för att kunna associera en förfrågan med en annan så behövs det ett sätt att kunna lagra användardata mellan HTTP förfrågningarna. Session och cookie parser löser detta genom att session tilldelar klienten en id och cookie.

ConnectFlash

Connect flash är ett middleware som används för att lagra meddelanden från servern och visa upp det för klienten i ett form av flash meddelande. Flash meddelandet har olika typer av utseende beroende på vad för typ av meddelande som skrivs ut. Om något inte är korrekt visas t.e.x. felmeddelande med rött.

Databas

Denna del kommer att handla om vår databas och hur den är kopplad till våra modeller (User och Bookings).

MongoDB

MongoDB är ett dokument baserat databas och den ledande databasen som inte använder "relational database management system" (RDBMS). I detta projekt användes Mlab som är en cloud baserat databastjänst och hanterar MongoDB. Vi använde även Mongoose för att koppla våra modeller med vår databas (se figur 11 på nästa sida).


```
var mongoose = require('mongoose');
var mongodbUri = 'mongodb://dat076_project:rucus1@ds157439.mlab.com:57439/database_clinic';

mongoose.connect(mongodbUri);

var conn = mongoose.connection;
conn.on('error', console.error.bind(console, 'connection error:'));
conn.once('open', function () {console.log("Booking connection successful");});

// Booking schema
var BookingSchema = mongoose.Schema({
  date: {
    type: String,
    index:true,
    required: true
  },
  startTime: {
    type: String,
    required: true
  },
  endTime: {
    type: String,
    required: true
  },
  patient: {
    type: String,
    required: true
  },
  isBooked: {
    type: Boolean,
    default: false,
    required: true
  }
});

var Booking = module.exports = mongoose.model('Booking', BookingSchema);
```

figur 11: visar hur bokningsmodellen kopplas till databasen med mongoose

Valet till MongoDB var inte enbart för att vi ville lära oss en NoSQL databas men även p.g.a. att den är mer tillämpad för programmerare, rent strukturmässigt. Valet av databasen har varit någorlunda lämpligt i detta projekt. Fördelen var att de nödvändiga funktioner som har använts för att hämta, lagra, uppdatera samt ta bort information från databas redan fanns inbyggd. Vi märkte dock att en nackdel med MongoDB är att det tar långt tid att hämta information vid förfrågan. MongoDB var även mindre effektivt jämfört med t.e.x. mysql eftersom man inte kan använda 'joins' (i alla fall inte i den versionen som vi har använt). Följdaktligen blev vi tvungna att skicka flera förfrågningar när vi skulle hämta eller uppdatera element från olika kollektioner. Nedan visas ett exempel på detta.

```
router.post('/unbookIt', function(req, res, next){
  Booking.update({'_id':req.body.id},{ $set:{patient:'Unbooked', isBooked:'false'}}, function (err, result) {
    if (err) {
      return handleError(err);
    }
  });

  User.update({'_id':req.user.id},{ $set:{hasBooked:'false'}}, function (err, result) {
    if (err) {
      return handleError(err);
    }
  });
  res.redirect('/patient');
});
```

figur 12: två separate förfrågan till databasen

Databasmodeller

Users

Modellen för användare som innehåller nödvändig information såsom namn, användarnamn, email, lösenord, en attribut för att avgöra om användare är admin eller vanlig patient och slutligen en attribut för att kontrollera om en patient är bokad till en tid eller inte.

Bookings

Modellen för bokningar som innehåller nödvändig information såsom datum, starttid, sluttid, namn på patient som är associerad med bokningen (markerad med 'unbooked' om ingen patient är bokad) samt en attribute som avgör om en viss bokning är tillgänglig eller inte.

Tester

För våra databasmodeller 'users' och 'bookings' har vi implementerat tester som kontrollerar att såväl användare som bokning kan skapas, uppdateras och hämtas från databasen på rätt sätt. Förutom tester för dessa modeller har vi även implementerat tester för vissa av våra GET och Post metoder. I båda fallen använde vi mocha för att skapa testerna och samtliga testfiler finns under mappen 'tests'. Testerna kan köras med kommandot 'mocha' eller 'npm run test' och nedan är en bild på resultatet av testerna som har implementerats.

```
Server started on port 3000
Booking connection successful
User connection successful
Test cases
(node:10624) DeprecationWarning: Mongoose: mpromise (mongoose's default promise library)
1
  ✓ Should be invalid if attributes of bookings are empty
  ✓ Tests if bookings can be stored
  ✓ Test to save a booking to the database
  ✓ Finds a booking from the database (130ms)
  ✓ Updates created time in database

Routing tests
  ✓ Testing get function of homepage (59ms)
  ✓ Testing get function of registration
  ✓ Login page should be found
  ✓ Patient page should be found
  ✓ Admin page should be found
  ✓ Should not work

Test cases
  ✓ Should be invalid if attributes of users are empty
  ✓ Tests if user can be stored
  ✓ Test to save a user to the database
User already in database
  ✓ Finds a user from the database (639ms)
  ✓ Updates email of the created user in test

16 passing (4s)

PS C:\Users\Tasdi\Documents\GitHub\Web-Applikationer-DAT076\dat076> |
```

figur 13: bild på utseende efter tester har körts

Vad skulle en annan programmerare ha nytta av att veta?

Models

Modeller (Users och Bookings) kopplas med databasen och här inkluderas även hjälpmetoder för inhämtning, borttagning och uppdatering.

Views

Inkluderar handlebars och javascript templates/mall som används av alla routing-funktioner.

Routes

Modellernas kontroll klass existerar här och ansvarar för CRUD (Create Read Update Delete) operationer.

Tester

Ramverket mocha används för att testa routing samt databasmodeller.