Group 12 Technologies Presents:

新しい日の誕生

M A T T E R H O R N

Table of Contents

Purpose: Matterhorn is a reservation app that allows anyone to establish a reservation schedule and allows anyone to keep track of their appointments.

Functionality

I'm dividing this part into core functionality and extended functionality. Regardless of what the final project looks like, there needs to be a baseline of core functionality. Once that is implemented, we can add in extended features that we've discussed.

Core Functionality:
For the sake of clarity, I'll be referring to people making appointments as "users" and the owners of that appointment system as "schedulers".

For example, Company A is a scheduler that has tables that users can make reservations for.

- Anyone can make a reservation schedule.
- Anyone can be a scheduler
- We will treat users and schedulers as separate entities in our database and our website

Schedulers:

- Schedulers can signup using a username and password
- Schedulers can login using said username and password
- Schedulers can name their company/group
- Schedulers can edit all previously mentioned account information
- Schedulers can create 0 or multiple schedules
- Each schedule will have:
    - a date range (over what days is this available?)
    - a name(massages, haircuts etc),
    - a start time (when does business open),
    - and an end time (when does business close),
    - an average appointment length, (How long does an appointment last?)
    - a capacity for every appointment (A haircut supports one person, but a restaurant may be able to hold 40 people at one time interval)
- Schedules can be deleted
- Schedules can have timeslots blocked out manually by the scheduler. Say the scheduler is unavailable between 1pm and

3pm, they can show that on the schedule by blacking out certain times.
- Schedulers can specify whether users can make notes about their reason for appointment (A doctor's office might want a "reason for visiting").
- Schedulers will be able to view all concurrent schedules using the website and mobile app.
- Schedulers can logout
- Schedulers will be able to accomplish all previous mentioned actions using the app or the website.

Users:
- Users can signup using a username, password, and email
- Users can login using specified username and password
- Users can enter their names, that schedulers will see on their schedule. Their username will show up on schedules if the user does not enter a name
- Users can edit any of their account info as necessary.
- Users can search for any scheduler, whether that scheduler is a company, club, or group
- Users can select an interval on a schedule provided by a scheduler
- Users are able to view all their currently claimed intervals, as these are their appointments
- Users can cancel any previously made appointments
- Users can logout
- Users will be able to accomplish all previously mentioned actions using the app or the website

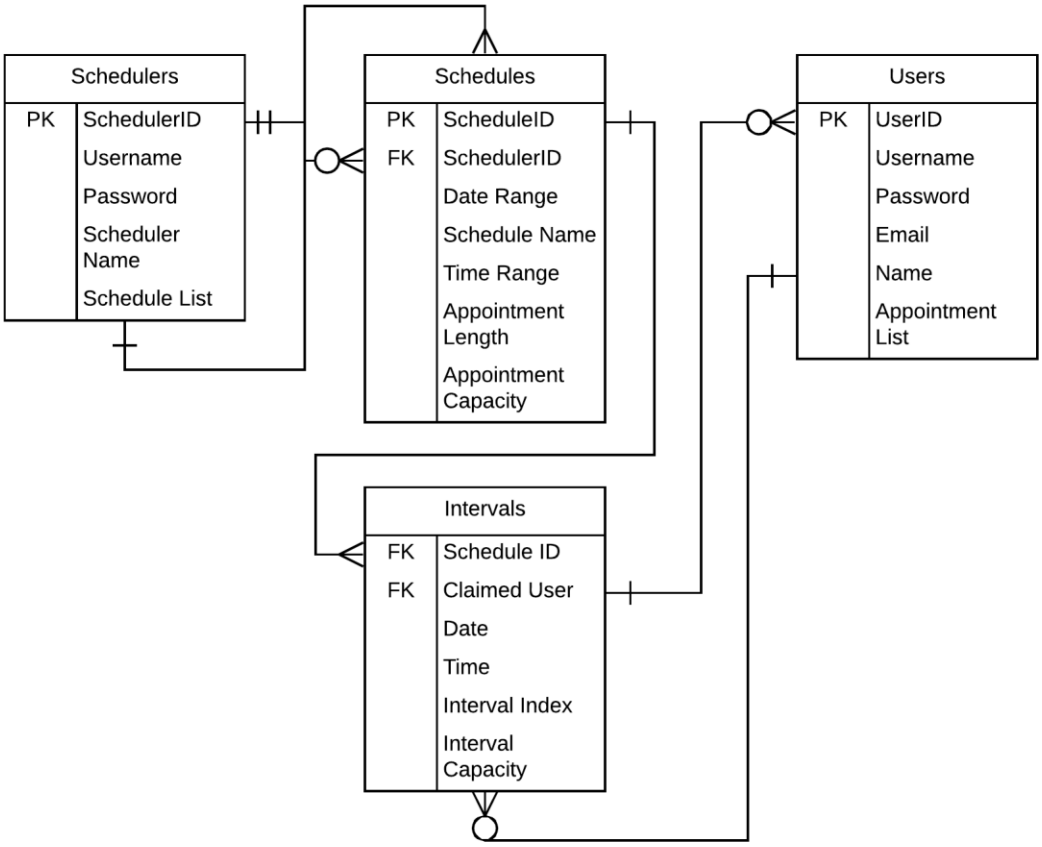
Extended Functionality:
This area can be added onto as the project progresses, and we can assign priority to these features once core functionality is established. If you think something should be added here or if I forget to add something here, remind me.

Schedulers
- Schedulers can have variable capacity for some intervals. This would allow restaurants to organize their tables. (Alper's Idea)

Users
- Users can be emailed after making their account. Can be emailed to reminded of their appointments. (Adam's idea)

The Design

The Database



So, let's talk about this. Don't let the ERD overwhelm you.

Schedulers
When a scheduler makes a schedule, this will make a schedule object that is tied to that scheduler via their schedulerID. Schedulers are capable of viewing all their made schedules via their schedule list.

When a scheduler makes a schedule, they will be prompted to enter all of those schedule parameters (date range, name, time range, appointment length, appointment capacity). Once the scheduler has entered the parameters for their schedule, the schedule object generates an array of interval objects to represent available appointments

The Interval Object:
The interval object is what will be used to represent appointments. It is generated from the time parameters the scheduler sent to the schedule object. An interval's width is established by the appointment length and the amount of intervals in a schedule can be represented by the following formula:

Time Range(in minutes) / Appointment length(in minutes)

If a schedule has an 8 hour schedule window, and the average appointment length is 20 minutes, the amount of intervals can be represented by (8 * 60) / 20) = 24 intervals.

These intervals can be represented by an array of interval objects within the schedule object.

Each interval object will have an associated index of its array, a time, a date, and a capacity to represent how many users can claim that interval. It will also have an array to represent users claiming that interval. By default, an interval is not taken, it is open, and any user may claim it. Once a user has claimed an interval, there are two cases for that interval. Either it remains open if more users may still claim it, or it becomes closed as it's user capacity has been reached.

Users:
Users create accounts with their username and password and email. They can also edit their display name. Their username may be Jane1234, but for reservations they may want to appear as Jane Smith. Users can view all their scheduled appointments, via their appointment list which is a list of all intervals that they have claimed.

The API

At some point, I'll put a use case diagram here to better outline the necessary functionality of our API. I'm going to layout the general design of our API here using the assumption we'll structure it similarly to our last project. This means we'll be using swagger.io to route to controllers which contain our API functions.

I'm going to divide these controllers into Scheduler actions and user actions. We can further divide these into more technical spaces in the future.

Scheduler Functions:

Scheduler signup (username, password, scheduler name):

This function will sign up the scheduler and register theirn information in our database

Scheduler login(username, password):

This function logs the scheduler in using provided info and matching it to a database entry. Assuming our authentication is structured how it was previously, we will give the scheduler a session object at this point so we know what scheduler is calling these other schedule functions.

Create Schedule(Date Range, Schedule Name, Time open, Time close, appointment length, appointment capacity):

This function will create a schedule based off the schedulers desired parameters. Remember this also includes auto-generating the interval object subarray from the principles discussed in the database section. This will probably be the hardest function to implement as it is most complicated. Also we must discuss how dates are implemented.

Date representation problem:
How can we represent a date? We cannot assume that our database will have infinite storage to represent an infinite calendar.

Currently the idea is that we'll represent our schedule as some time window (say, a week), and that future weeks examined by the scheduler will just be repeated based off their current

week. Users will only be able to see the current window that has been filled out by the scheduler.

As far as how are we representing dates, are they integers? Objects? Currently we're assuming that the date creation is handled on the front end, so we don't have to think about it for the API. We just know that the API will store dates and understand that they are ordered in some manner (Jarrod's idea that I like).

Remove Interval(scheduleID, date, time begin, time end):

This function will be used by schedulers to black out times they are unavailable. Theoretically this should be the same as user's making appointments that claim intervals. Instead of the user claiming an interval, the scheduler is. Note that the schedule ID refers to the schedule the scheduler is editing, and the rest of the parameters are used to manipulate the interval object.

View Schedules(ScheduleID[array])

Schedulers should be able to view a list of their created schedules.

List Schedule(ScheduleID, date begin, date end)

When a scheduler goes to view one of their their schedules, it should display their schedule intervals over a certain date range. The date range should have a default window of time, but it should also be adjustable by the scheduler's input.

Edit Info(username, password, company name)

Hopefully, self-explanatory. Companies can edit their info in the database.

Logout()

Signs the scheduler out. Assuming this is structured like our last project, when the user signs in, we can give them a session object, and remove it when they leave. There's probably simpler ways to perform authentication than Nik did though.

User Functions:

Signup(Username, password, email, name(optional))

This will be used to add the user to our database. Remember the user does not need a screen name, but they do need a username. If a user does not provide a screen name during signup, they can add one or change it using the edit info function.

Login(Username, Password)

Logs the user in. Gives user session object so we may know who is making calls to other user functions once the user has logged in.

Edit Info(Username, password, email, name):
Users should be able to edit their account info as necessary.

List Schedulers(Search parameters):

Users should be able to search for schedulers, which are then displayed in a list. Users will then be able to select a scheduler to access that scheduler's schedules to make appointments.

View Schedules(ScheduleID[array])

User's should be able to view all schedules provided by a specified scheduler.

List Schedule(ScheduleID, date begin, date end)

I realize that these last 2 functions exist for both users and schedulers so maybe we will route the two of them to some auxiliary controller for organization.

Claim Appointment(ScheduleID, UserID, Interval Index)

This should assign that user to this particular interval of the schedule. This should enter the user's id into the claimed user's field of the interval object. This should first check if the interval is already full capacity, so we do not overcrowd any particular interval. The interval object should also be added to the user's appointment list, so they may later view their appointment.

List Appointments(userID)

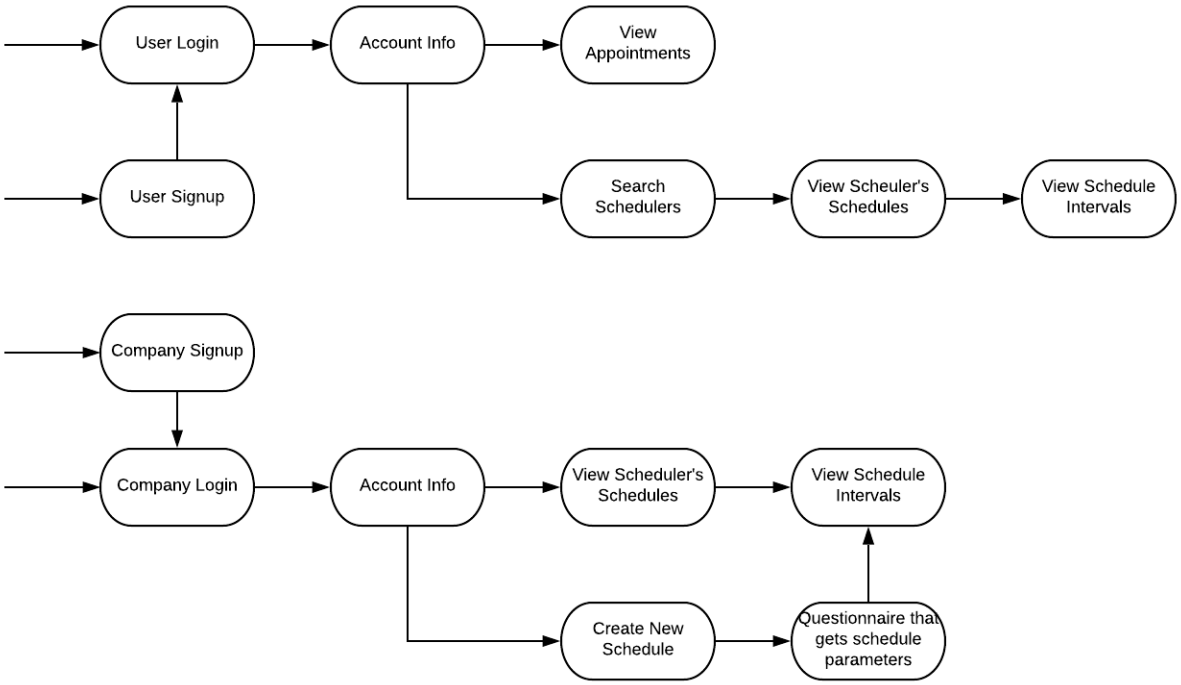
This should list all intervals claimed by a user and their respective schedule and scheduler names, as well as date and time associated with that interval.

Logout()

Signs user out and deletes their session object.

The Front end

Currently, We don't know what every page is going to look like. That will be established at the front end development stage (See Development Plan section). Adam did make a graph of what our website's general functionality/layout should be. I cleaned that up so you can see below:



As Users and schedulers re treated as separate entities, they will have different logins and at each of these steps, they'll be able to take different actions, as seen in our API.

As stated previously one of the problems we face is determining how dates are represented as data and ordered chronologically. We are assuming this will be done on the front ends part, either through some plugin, or javascript magic. We'll hammer that out during the front end development stage.

The App

We do not have much planned for the App at this stage. It represents a final frontier in our knowledge base for this project. As such, I have designated it a complete stage of the project development plan.

What we know:

We'll be using an Android development platform as it's free and we all know Java to an extent.

Alper is looking into how to use the Ionic framework. If he wants to take a head dive into spearheading the app development design, I'm all for it as we currently have nothing.

We must have our app as a downloadable app, not just a mobile friendly version of our website. Adam was saying it may be possible to have a downloadable app that is basically a clone of our website. That may be useful. If we use bootstrap for website development, from my understanding that is natively mobile friendly and may make our app easy to make.

Currently we have specified that we want users and schedulers to have full functionality on both the website and mobile versions of our project.

The Development Plan

The way I see it, a lot of our issues from the previous project came from the way we divvied up work from the beginning without fully understanding how each part was interconnected. This resulted in informational deadlock. The front end could not build as they did not know how the backend functionality was to be done, the backend could not be developed because requirements and functions had not been established. We sat on our hands while Nik eventually whipped us into shape. However, by the end of the project I feel as though we didn't understand why the project worked or how parts of it fed into each other and we relied entirely on Nik to resolve that.

To resolve this I think it will help us to break the development into stages in which a single part of the project is developed by us all. At each stage of development we first assess the design of that stage, establish technologies, build a timeline for the tasks to be completed, and then finally assign those tasks.

Having everyone working together across stages will also allow our meetings to have more unified purpose.

Please see the development outline on the next page.

Stage 1: The data

1. Assess Database and API design
2. Establish Technologies to be used. MongoDB with Compass? Swagger?
3. Create tasks
4. Create Timeline (I think each stage should have a Gantt chart)
5. Assign tasks/create roles
6. Begin development
7. Test funcitonality

Stage 2: The Front End

1. Assess Website design
2. Establish website structure. What will pages look like?
3. Establish front end Javscript function desig, make sure it can line up with the back end
4. Establish Technologies to be used. Bootstrap? Other Frameworks?
5. Create tasks
6. Create Timeline (I think each stage should have a Gantt chart)
7. Assign tasks/create roles
8. Begin development
9. Test functionality

Stage 3: The App

1. Assess APP design
2. Establish Technologies to be used.
3. Create tasks
4. Create Timeline (I think each stage should have a Gantt chart)
5. Assign tasks/create roles
6. Begin development
7. Test functionality

Stage 4: Final Preparations

1. Assess possible extended functionality that could be added to each previous stage of development
2. Establish what can be added with remaining time
3. Assess how our presentation will happen
4. Create Slides
5. Test Extended Functionality
6. Prepare demo