# meWap

- even easier scheduling


**Group info**

Group name: **meWap**

GIT-repository: **https://github.com/raksooo/meWap**

Group members:

| | | |
|---|---|---|
| Josefin Ondrus | (921006 - 4128) | josefin.ondrus@gmail.com |
| Elin Ljunggren | (920423-1626) | elin.l.ljunggren@gmail.com |
| Oskar Nyberg | (930319-1713) | oskar.nyberg@gmail.com |
| Emma Gustafsson | (880920-7841) | emma.i.gustafsson@gmail.com |

## What is meWap?

meWap is a tool for scheduling meetings, get-togethers and other events. It's main purpose is to simplify the moment when everyone needs to check their calendars to agree on one date which is suitable for everyone involved. We have all to some extent been introduced to the already existing tool doodle, but all of us agree upon a few functions where doodle could have been better.

## User roles

meWap should be easy to use, whether you are the person who wants to invite people to an event or if you are invited. Therefore there are no administrative roles in the application, only user roles. All users can create their own events, and in these events they have certain rights as creators such as editing, removing and inviting participants to the event.

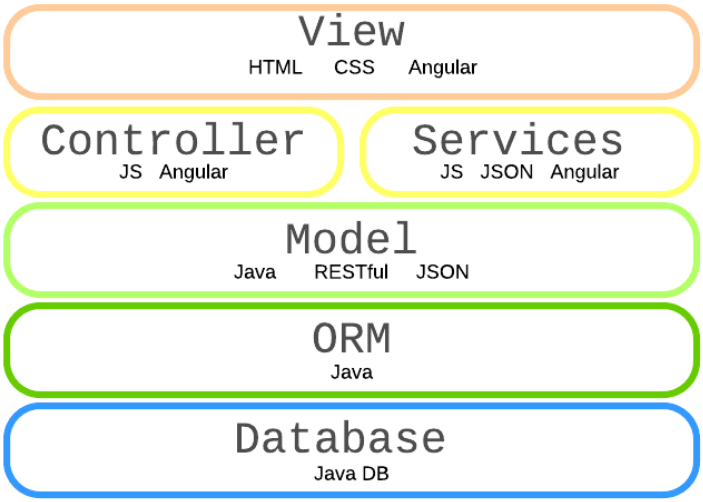
## Implemented use cases

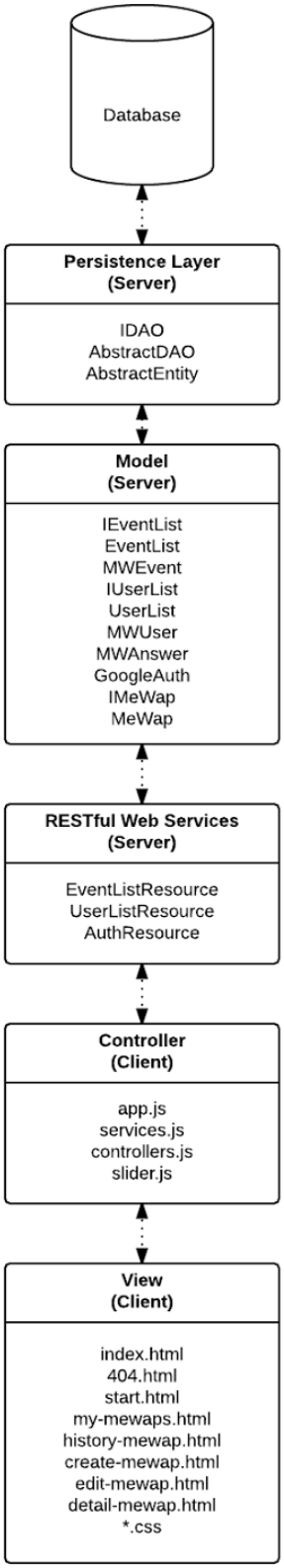
- Log in to meWap with your Google account, which saves your user to the database.
- Log out
- Create a new event
    - Name the event
    - Choose length of event (time or all day)
    - Choose dates and time
    - Set a deadline for answering the event
    - Choose if you want to send a deadline reminder to all participators
    - Choose if and when you want notifications when the participators answer
- Show current events in a list, one list for the events you created and one list for the events you participate in.
- Show detailed view of a specific event
    - Who are participating in the event
    - Information about the participators answers
    - Number of answers on different days
    - Which participators have already answered
    - When the answer deadline occurs
- Edit existing event
- Delete event
- Invite participators
    - Receive invites by email
- Answering functionality
    - Choose days/times you can participate on/in
    - Autofill answer for an event based on logged in users Google Calendar.
- Creator receives notification by email when participators have answered, the number of notifications depends on the settings chosen.
- Show a list of past events
- Detailed view of the past events
    - no editing available on the event
    - not possible to change your answer as a participator
- Clear history - all events where the deadline is passed and the logged in user is the creator.

## Technical specifications

### UML/layered view

Layered view of the system:



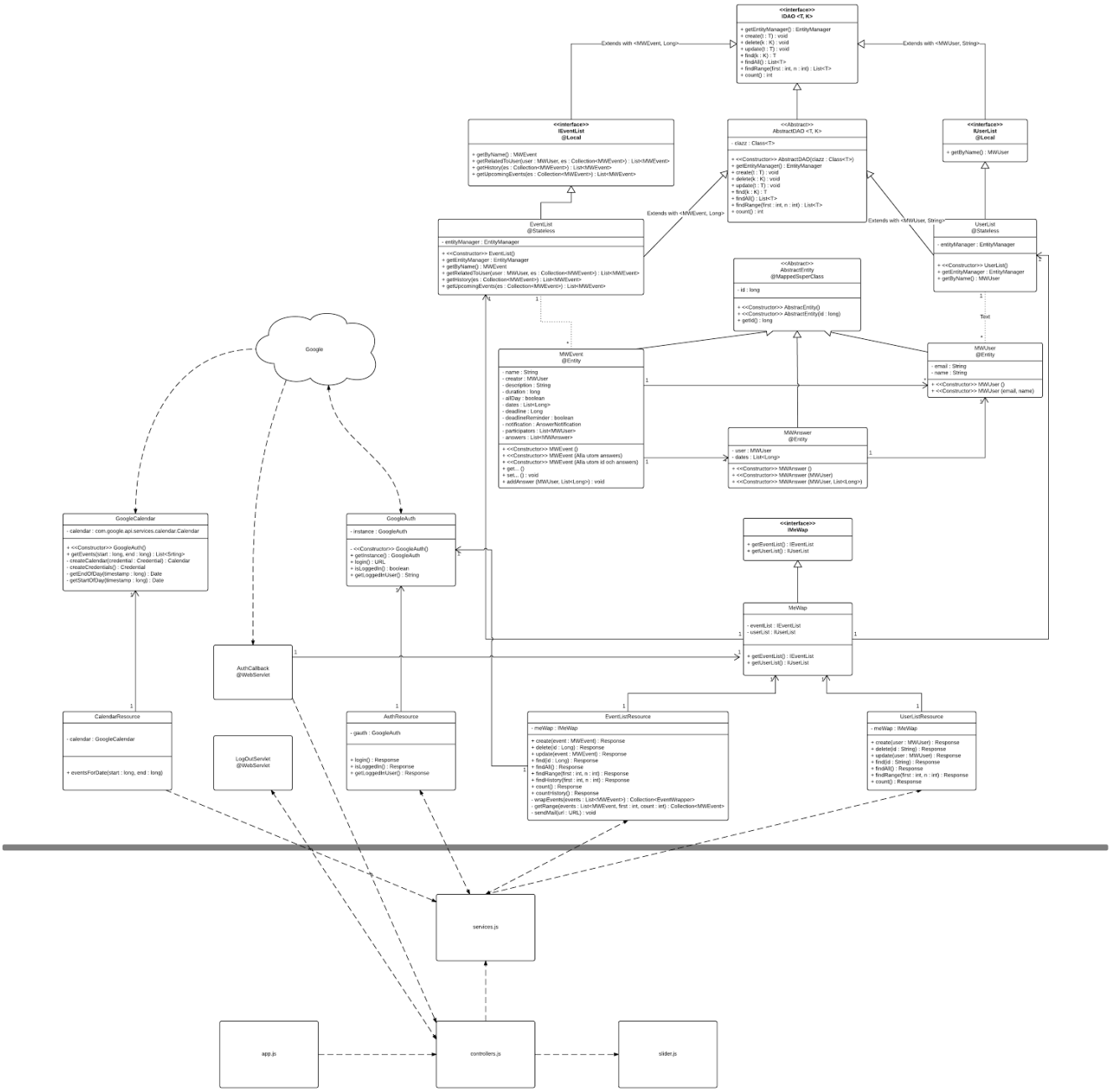




The UML-diagrams are also presented in full size in the folder “diagrams” in the repo on github.

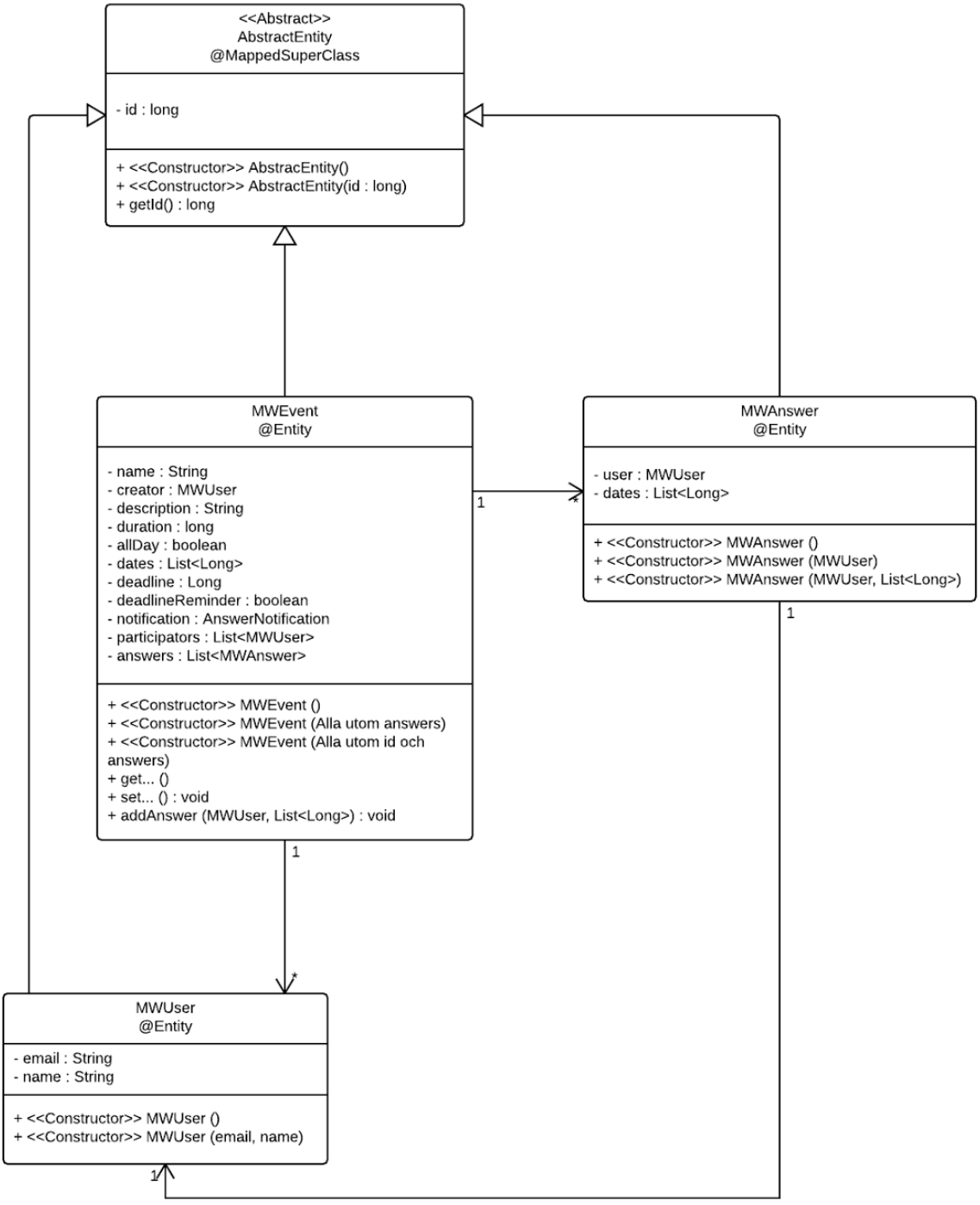
UML Documents:

- Application UML - UML of whole application, frontend, persistence, model, google communication classes etc.
- Backend UML - Of the whole backend
- Model UML - Of the model classes without heritance.
- Layers view with classes

Application UML:



Model UML:


<<Abstract>>
AbstractEntity
@MappedSuperClass
- id : long
+ <<Constructor>> AbstracEntity()
+ <<Constructor>> AbstractEntity(id : long)
+ getId() : long
MWEvent
@Entity
- name : String
- creator : MWUser
- description : String
- duration : long
- allDay : boolean
- dates : List<Long>
- deadline : Long
- deadlineReminder : boolean
- notification : AnswerNotification
- participators : List<MWUser>
- answers : List<MWAnswer>
+ <<Constructor>> MWEvent ()
+ <<Constructor>> MWEvent (Alla utom answers)
+ <<Constructor>> MWEvent (Alla utom id och answers)
+ get... ()
+ set... () : void
+ addAnswer (MWUser, List<Long>) : void
MWAnswer
@Entity
- user : MWUser
- dates : List<Long>
+ <<Constructor>> MWAnswer ()
+ <<Constructor>> MWAnswer (MWUser)
+ <<Constructor>> MWAnswer (MWUser, List<Long>)
MWUser
@Entity
- email : String
- name : String
+ <<Constructor>> MWUser ()
+ <<Constructor>> MWUser (email, name)
1
*
1
*
1
1
1

## Approaches

We have worked with a service based approached, RESTful web services with JAX-RS and AngularJS. We also used the request based approach with servlets for the login/logout function with google authentication.

## Participating software components

We used some libraries to make the development of meWap easier, the ones we are using are:

- Arquillian
    - For testing of model and persistence layer
    - In addition to only using JUnit it enables testing of persistence
- Angular.js
    - For making better controller structure and more smooth update of view
    - Angular also helped us communicate with the servers REST services
- Google Calendar Java api
    - We needed to communicate with Google's REST-api for retrieving all the events in a users calendar for a specified time interval. We let the backend use Googles Java api for their calendar service. It was poorly documented but after a lot of experimenting we got it to work.

Other software components used:

- Glassfish
    - The web server used in meWap
- Maven
    - Used for the project management, managing dependencies and building of the project.

We thought about using some other libraries in addition to the ones we used. But we made the choice it would be easier or would make a better end product if we implemented them ourselves. The ones we thought about using were:

- Bootstrap
    - It is a CSS library which would have speeded up the styling of the views a bit. We decided to write our own CSS because it would give the end product a much better looking appearance.
- SocialAuth
    - We were advised to use SocialAuth for authentication with Google. But after hours of trying to get it to work with either CDI or a servlet we gave up. The main problem was that it is poorly documented. We also tried Google Java Api which turned out to be even worse in documentation. Finally we decided to create Google authentication from zero, with get requests to their rest-api.
- Image slider
    - For the image slider on the startpage we thought about using a prebuilt one. But we decided to build our own with javascript and css. We used pure javascript instead of angular because one of us had previous knowledge in js and we thought it would be a good idea to learn more about what angular is built on.

## Packages

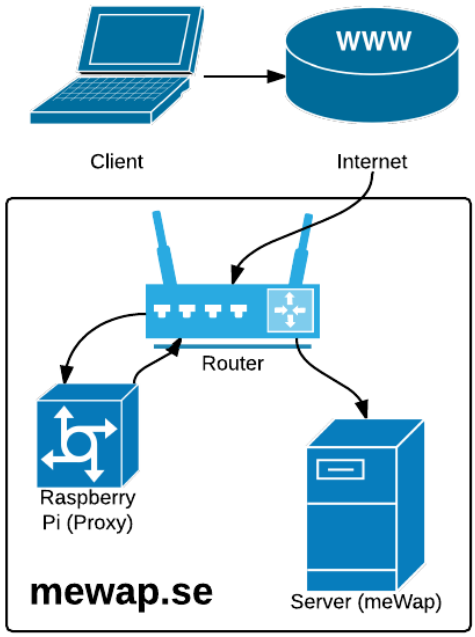
- mewap
  - Only contains ApplicationConfig.java because it didn't fit in any package
- mewap.persistence
  - Persistence classes. Model classes extends and implements these classes to enable them to be persisted
- mewap.model
  - Contains the model, i.e. what is actually visible in the application
- mewap.calendar
  - Contains REST-service and google-communication classes for Google Calendar
- mewap.auth
  - Contains REST-service and google-communication classes for Google OAuth2

## Physical setup (tiers)

For debugging and testing we ran the application on our own computers which means that theres only one tier, both the client and server are on the same computer. The application were located on http://localhost:8080/meWap/.

For deployment we used a more advanced setup. We want the users to be able to access it easily, therefore we got the domain mewap.se. First the adress for using the application was http://mewap.se:8080/meWap/ which isn't very user friendly. We solved it by letting a raspberry pi act as proxy using Nginx. The result is that the application is available at http://mewap.se.

The image to the right is a visual presentation of the final setup.




## Other

### Email functionality

We wanted to implement email functionality to enable us to send emails to invite users to events and notify the creator of events when participators answer. We didn't want to install and setup an smtp-server. One of the group members already had a server with an smtp-server installed which we used. We figured the simplest way to send emails from that server was to make a small php-script which sends the emails and from our java backend make a simple http-request with query-parameters to that script. This was a simple but not optimal solution. The reason it is not optimal is that the emails are sent from mewap@oskarnyberg.com which is not a very representative address. A future implementation using a local smtp-server would enable us to send from something like noreply@mewap.se.

### Curl test

To test our REST services we used curl. After a while we got tired of running the same curl command over and over, which led us to writing a shell-script which runs all curl commands and prints the result. It doesn't do any assertions or makes comparisons but it only prints relevant data and which makes the testing easier for the developer.

### Time zone (Winter/summer time)

A quite funny problem we ran into was the change from summer time to winter time which occurred between the second last day and the last day of the project. It was a problem which we would have had to solve even if there wasn't a time zone change but we would never have noticed that there were something wrong if this project were placed other parts of the year. The way this problem affected was that the javascript date-object adjusts after timezones which made manipulation of dates by some hours change the timezone.

### Date placing algorithms

To receive a simple presentation of the dates the creator have chosen, we have implemented two algorithms. One to arrange all day events and one to arrange the events that will take place in a specific time interval. The available dates and times are represented in a matrix where the first row represents a time or day and the first column represents a date or a weeknumber. Our first thought was to generate two matrices, both arranged as described above and calculate which one of the matrices was the smallest and return. We decided not to implement the part with the comparison of the two matrices because we received a good result by arranging all day events by week number and time interval events by single dates.