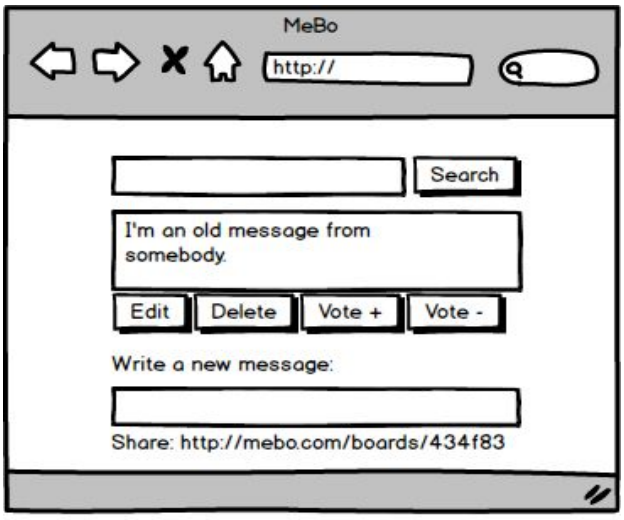
Application Design and Software Structure Report

Login-Free Message Board (MeBo)

1. Introduction

The Login-Free Message Board (MeBo) is a web application which enables users to create their own message board, share a link with other users and write message in a collaborative way. Every user can post new messages or edit existing messages - although if they were written by other users.

- Users can create boards (without registration or login)
- Boards can be shared by their unique link
- Users can write messages to the board
- Every user can edit, delete or vote on every message
- Messages can be searched and ordered



2. Design and Implementation

2.1 The REST API Specification

Method	End Point	Explanation
GET	/boards/<ID>	Returns the requested board. **Error:** ● 404 if the board was not found
POST	/boards/<ID>	Creates a new board with the given ID or returns an error is this ID already exists. **Error:** ● 404 if the board was not found
GET	/boards/<ID>/messages	Returns a list of all messages on the board. The list might be empty. **Error:** ● 404 if the board was not found
POST	/boards/<ID>/messages	Creates a new message on the given board. **Error:** ● 404 if the board was not found
PUT	/boards/<ID>/messages/<ID>	Updates an existing message. Error: ● 404 if the board or message was not found
DELETE	/boards/<ID>/messages<ID>	Creates an existing message. **Error:** ● 404 if the board or message was not found

2.2 Front-end Architecture Design

Design Decisions

- Intensive use of AngularJS 1.5 components to create independent and reusable widgets
- A service to interact with the "`/boards/`" REST API endpoint
- A service to interact with the "`/messages/`" REST API endpoint
- Package structure by type (service, component, view), not features

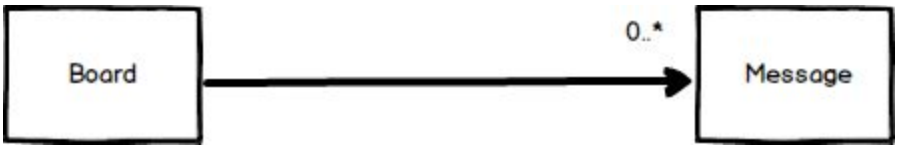
Package Structure

```
/app
      /components
            /board
            /footer
```

```
            /navigation
            /...
      /services
            /message.service.js
            /board.service.js
            /...
      /views
            /home.html
            /contact.html
            /...
/css
      /main.css
      /some-special.css
/index.html
```

2.3 Database Schemas, Design and Structure

Model



- Very simple data model
- Based on two entities: board and message
- A board can have 0 or multiple messages
- A board has some fields:
    - An unique ID
    - A creation date
- A message has some fields:
    - A text
    - A vote count
    - Optional: a title

The data is stored in a MongoJS database.

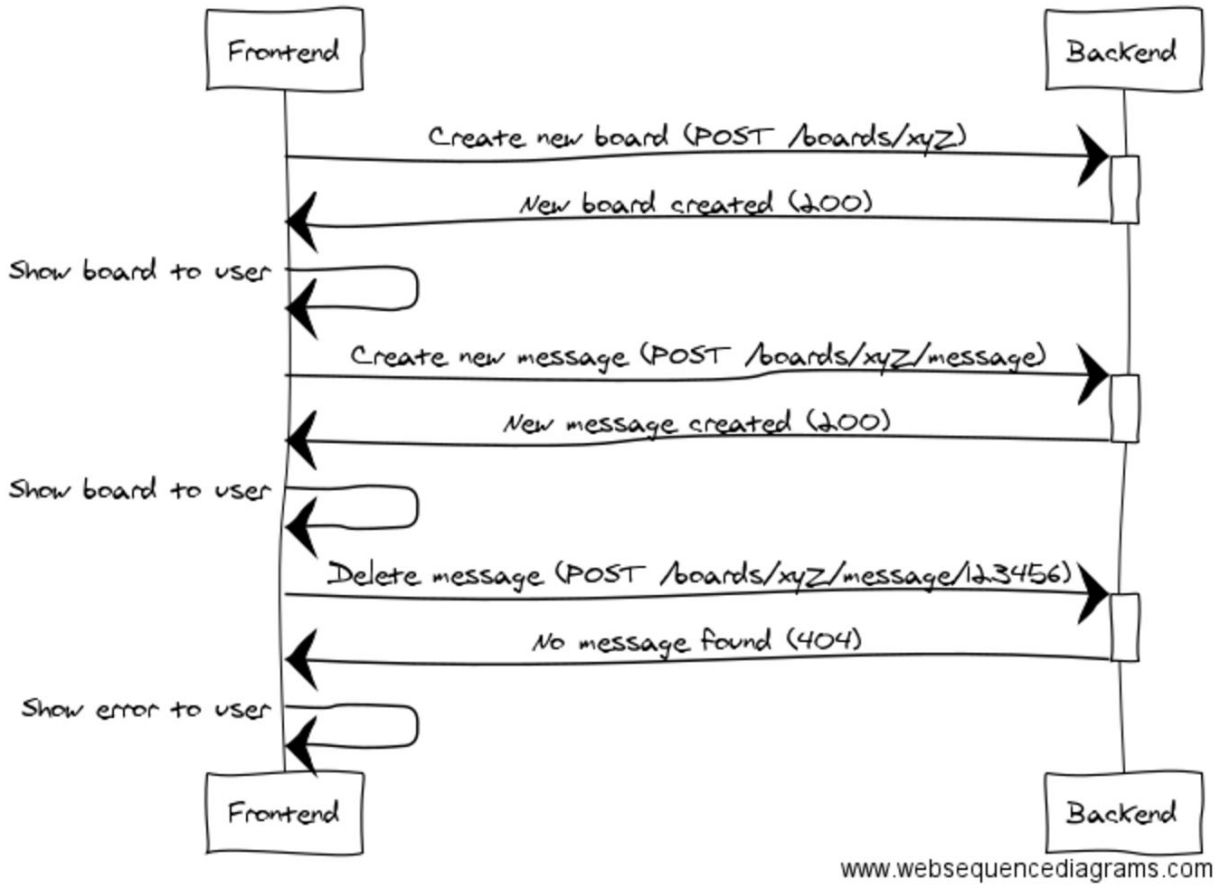
MongoDB Schema

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var Message = new Schema({
  id: { type: String, required: true } ,
  text: String,
  votes: Integer
  creationDate: {
    type: Date,
    default: Date.now
  },
);

var Board = new Schema({
  id: { type: String, required: true } ,
  messages: [Message],
  creationDate: {
    type: Date,
    default: Date.now
  },
);
```

2.4 Communication



3. Conclusions

- All three parts of the application (database, backend and frontend) use the same model
- Any domain object in the backend is represented by a REST API endpoint
- Every user-action is represented by an HTTP action on one of these endpoints
- It's not possible to provoke an error by using the frontend

4. References

- Tool used to draw the UI mock-ups:
  https://balsamiq.com/products/mockups

- Tool used to draw UML activity diagrams:
  https://www.websequencediagrams.com