Note:
Please consider the following for your submission:

The project needs to be submitted through moodle (there will be a page called *Project Submission*). Create a zip compressed folder and put all of your code inside of it. Your front-end code should be in a subfolder called *app* and your back-end code in a subfolder called *server*.

For the back-end, you have to write a node.js server application running on port 5300. The final server project must immediately run with the command *npm start* in the *server* directory of your project. Thus, make sure *npm install* and your build automation are automatically triggered and that all packages are specified in the JSON files (test this before the submission). Make sure that no installed dependencies are added to the server folder, i.e. exclude the *node_modules* folder. Assume that a PostgreSQL database called *webdev* is running on *localhost:5432* and user *postgres* with password *root* has full access to it. The database structure is exactly the one defined in the provided file *database.sql*.

The front-end can be built using either AngularJS or Angular (latest stable version at time of submission). If you choose AngularJS, please submit a zip-file containing all files. A click on index.html within the *app* folder must be sufficient to see the website. If you choose Angular, you can work on the project skeleton provided by angular-cli. However, your project has to start when typing *npm start* within the *app* folder, again including all build automation etc. Your website should then be accessible via *http://localhost:4200/*.

In case you do not meet these requirements, you fail this project. We will not debug your code until it works.

You have to submit the projects on your own, group submissions are not allowed. Overall this project will define 70% of your grade for this course. Those 70% are divided into 150 points split across the tasks as defined below.

The goal of this project is to build an interactive web application that aims to improve the communication between students and lecturers during university courses or lectures. The system should be designed as a dynamic web application that offers the following functionalities:

1. When visiting the website for the first time, users can select a username. This username should be posted to the server to receive the created student object including the studentID. The studentID should be stored locally as a cookie so that the client knows who is currently logged in. *Note: This is a strong simplification of what should be done in a productive system and violates all security standards.* (**10 pt**)
2. Once the user has such a cookie, she is able to access the web application and it can fetch the username for visualization. (**5 pt**)
3. The main page should display all available lectures. For this project it is sufficient to show the ones which we already added to the database and never change these. (**5 pt**)
4. When selecting a lecture, the user is directed to the page of the lecture. (**2 pt**)
5. This page consists of 4 sections: a live chat, a Q&A section, a mood visualization, and a live stream. The design and layout of this page is up to you but should make sense and will be graded. (**5 pt layout, 5 pt design, 10 pt responsiveness**)
6. The users can communicate via a live chat. The chat does not need to be stored in the database. This chat should be realized using websockets only. (**20 pt**)

7. Users can ask questions. These questions are shown in a dedicated "Questions" section. (**5 pt**) Whenever a new question is added by some user, the "Questions" section automatically updates in real-time (using websockets). (**5 pt**) Questions are editable by the author and changes are reflected on all other clients. (**5 pt**)

8. Every user can provide an answer to a question. (**5 pt**) Multiple answers per question are allowed. Answers are editable by the author. (**5 pt**) Again, the answers of a user are propagated to all other users using websockets. (**5 pt**)

9. Every question and every answer can be voted up or down once per user. (**5 pt**) Whenever a user voted, the Q&A section automatically updates in real-time (using websockets). (**5 pt**) Each client should see whether she already voted for a specific question. (**5 pt**)

10. The questions and their answers are automatically ordered by the difference of up and down votes (highest to lowest). (**10 pt**) This again should update whenever new votes have been made or new questions and answers have been asked (using websockets). (**5 pt**)

11. Users have the opportunity to express their current mood and change it as often as they want using a smiley-based interface (three different smileys to indicate their current mood). The three smileys should correspond to the mood values -1, 0, and 1. Each user should always be able to see, which mood they have currently selected. (**10 pt**)

12. There should be a visualization of the aggregated mood of all users (e.g. using a tag-cloud-like representation: Each smiley changes its size based on the relative amount of students that selected the associated mood). (**10 pt**) Whenever a user changes his/her mood, the mood visualization automatically updates in real-time (using websockets). (**5 pt**)

13. There should also be enough space for a potential live stream video of the lecture. For this project, you should simply use a thumbnail for this. (**3 pt**)

The front-end must be designed responsively and data-binding must be used. That means, your system MUST use ((AngularJS or Angular) and (Bootstrap))! Since you already use bootstrap, please make sure to use the standard template or some other template you find on the web (e.g. on wrapbootstrap.com). Responsiveness and design are graded as well.

In order to realize the system with the above-mentioned requirements, a server back-end needs to be implemented as well. This server back-end should offer a REST API (for client requests) as well as a websocket-based communication channel (in order to distribute changes event-based) and store and load the data to/from a database.

You can find the specification of the REST API that you should implement at `https://app.swaggerhub.com/apis/WebDevelopmentCourse/web-dev_practical_project_server_api/1.0.0#`.

The websocket specification can be found in Tabelle 1.

As a database you should use PostgreSQL. Please create a database with the following specifics:

```
CREATE DATABASE webdev
 WITH OWNER = postgres
      ENCODING = 'UTF8'
      TABLESPACE = pg_default
      LC_COLLATE = 'English_United␣States.1252'
      LC_CTYPE = 'English_United␣States.1252'
      CONNECTION LIMIT = -1;
```

We have created a database.sql script and added it as accompanying file to this project. Please execute it after creating your database. It will create all necessary tables and add some sample lectures.

Tabelle 1: Websocket specification

Event Name	Example Value
onNewQuestion (server sends)	{lectureID: <intValue>, questionID: <intValue>, voteRatio: <intValue>, author: {studentID: <intValue>, username: "<stringValue>"}, textContent: "<stringValue>"}
onNewAnswer (server sends)	{answerID: <intValue>, questionID: <intValue>, voteRatio: <intValue>, author: {studentID: <intValue>, username: "<stringValue>"}, textContent: "<stringValue>"}
onQuestionVoteRatioChanged (server sends)	{questionID: <intValue>, voteRatio: <intValue>}
onAnswerVoteRatioChanged (server sends)	{answerID: <intValue>, voteRatio: <intValue>}
onQuestionTextContentChanged (server sends)	{questionID: <intValue>, textContent: "<stringValue>"}
onAnswerTextChanged (server sends)	{answerID: <intValue>, textContent: "<stringValue>"}
onRegisterForChat (client sends)	{lectureID: <intValue>, studentID: <intValue>}
onSendMessage (client sends)	{lectureID: <intValue>, studentID: <intValue>, message: "<stringValue>"}
onReceiveMessage (client receives)	{lectureID: <intValue>, username: "<stringValue>", message: "<stringValue>"}
onMoodChanged (server sends)	{lectureID: <intValue>, mood: { positive: <intValue>, neutral: <intValue>, negative: <intValue>}}

Make sure to **accurately** implement the REST API and websocket specification and to use **exactly** the provided database scheme. Please also validate the received input and make sure your system responds with proper error messages and continues to work after invalid requests. We will test your project using our own database with this schema, so if you use any other schema or database name or login credentials these tests will fail and you will receive 0 points. The REST API and websockets will also be tested automatically, therefore make sure not to change any specifications. Even though we will test this automatically, write clearly structured code as this might reduce your points if you fail to submit readable code.

Submission due Saturday, March 31th 2018.