

# Report group 5

PollStar - a polling App

Ariadna González de Prado  
Ben Wooldridge  
Christina Meisoll  
Erik Larsson

# 1 Introduction

Pollstar is a polling app targeted for casual voting. The intended users are friends, families, or colleagues. The intended kind of polls are of a non-critical nature. For example, could a group of coworkers decide on a present for a jubilee or a family could vote on whose home hosts this year's Christmas celebration. Future features would include different types of polls for increased flexibility and ease of use. This could mean polls where: a user can vote for multiple choices, a user can add their own choices to a poll created by another user, and finally a time/date type poll for finding a time when all users can meet. Another set of future features would be registered users who can log in and create polls that are shareable only with other registered users, this would open up for more security and privacy.

The program code can be accessed at: [Powana/DAT076-Web-App \(github.com\)](https://github.com/Powana/DAT076-Web-App)

## 2 A list of use cases

- create a standard single choice poll
  - the user is given fields to enter one question and three choices to vote on
- add or subtract choices from standard poll when creating it
  - the user is given the option to deduct choices from the original 3 or add more if needed
- edit a single choice poll
  - the user is given the option to edit a poll after it has been created, for example when there is a typo in the question or choices
- answer a single choice poll
  - the user is given the possibility to answer a poll where only one of the choices can be chosen
- comment on a poll
  - the user can add a comment to the poll, so they can add more information if needed
- show the results of a poll
  - the user can view the results of a poll
- use both ID or address to access a poll
  - when a user wants to spread a poll, they can either share the whole address that takes one directly to the poll or the pollID can be shared and one enters the ID on the applications start page to reach the poll.

## 3 *User manual*

### 3.1 How to install the app

In the client folder, run: *npm install* followed by *npm run build*. Then, in the server folder, run: *npm install*.

### 3.2 How to start the app

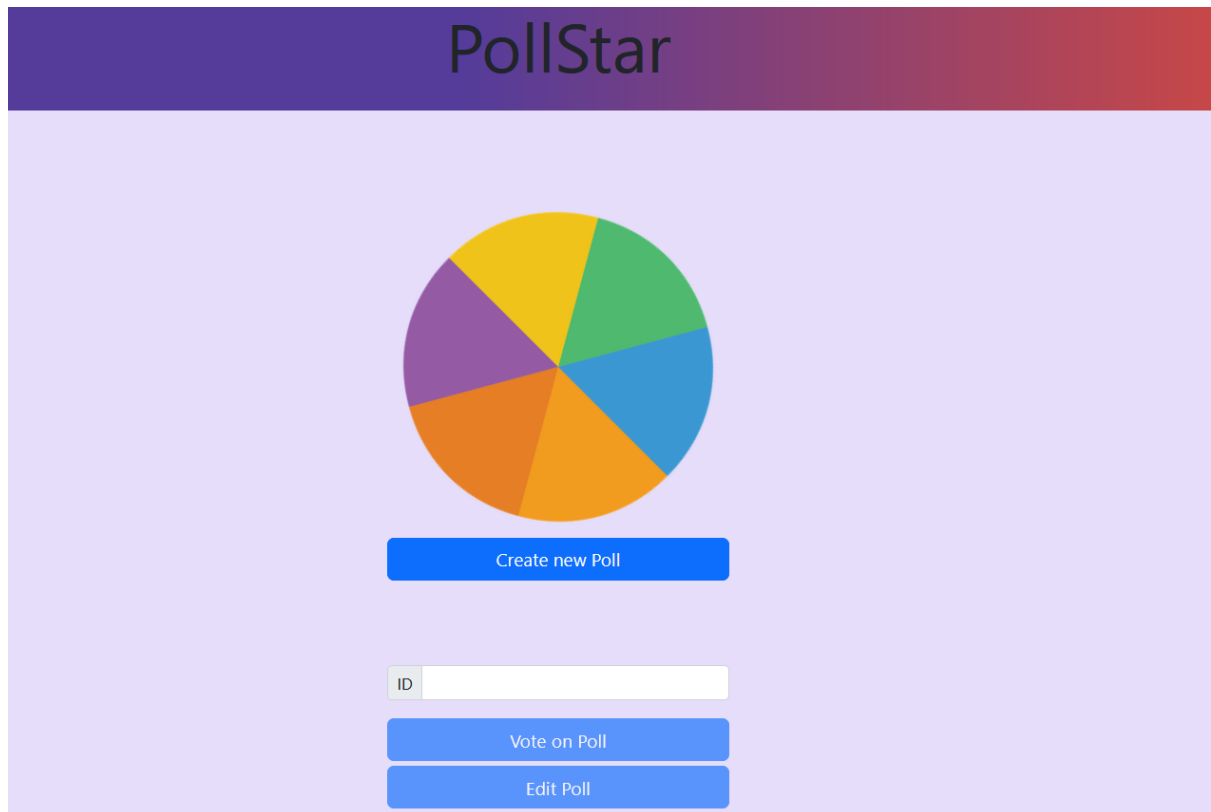
After the initial installment you can start the app by opening two command prompts. Then navigate to the applications server directory in one and to the client directory in the other. In the server directory run: *npm run dev* and in the client directory: *npm start*. Thereby it doesn't matter if you first execute your command in the server or client directory. When both commands are run the application opens in your browser.

Alternatively you can just use the *run dev* command and access the application on localhost:8080.

Finally, the application has also been Dockerized and is accessible at [pollstar.larssontech.com](http://pollstar.larssontech.com)

### 3.3 How to use the app

When PollStar is started you are met with the front page (Figure 1) that gives you access to the different ways of interacting with the application.

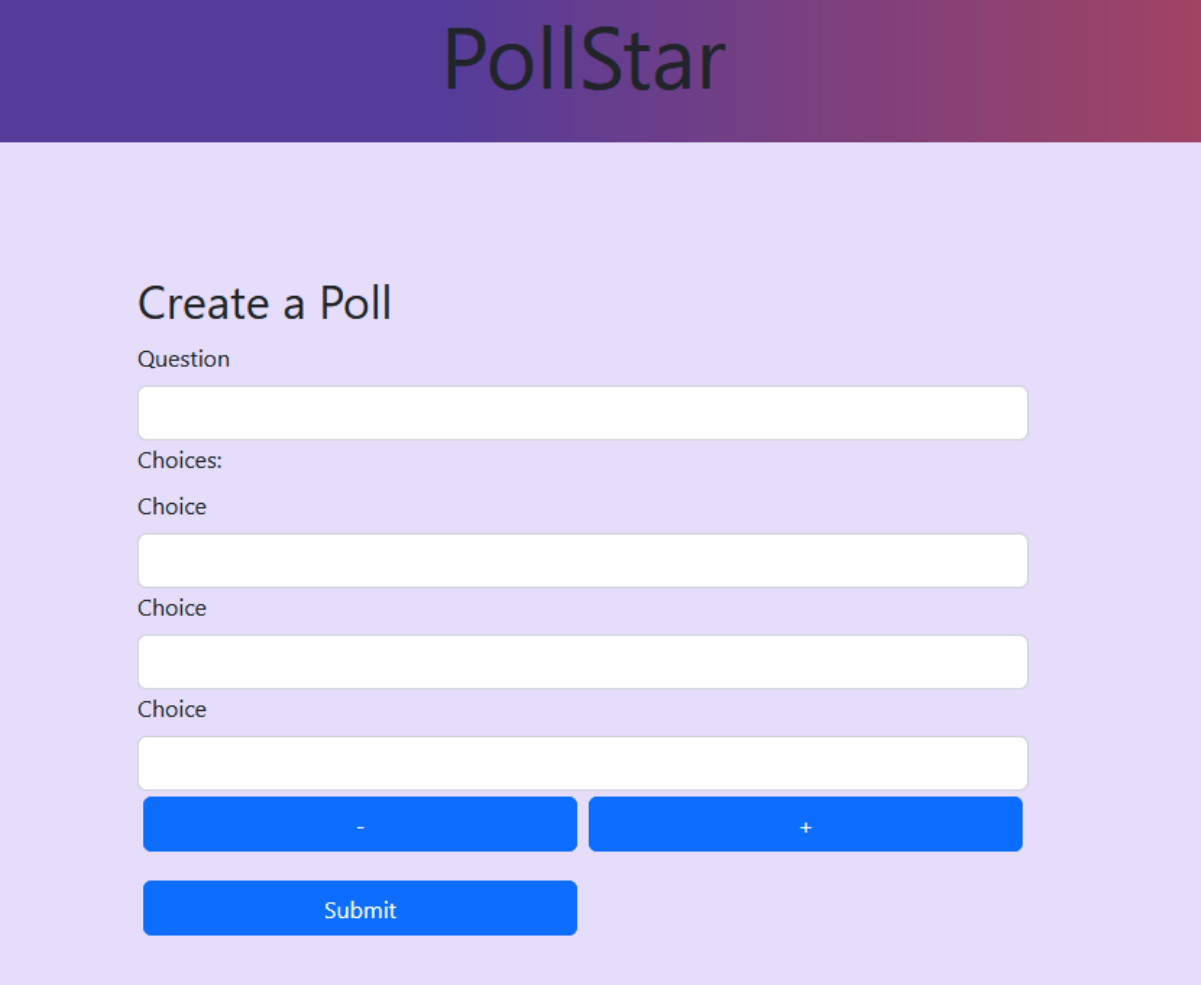


*Figure 1: PollStars front page, here can users decide whether to create a poll or access an existing one*

To create a new poll click on “Create a new Poll” and follow the instructions in section 3.2.1. If you instead want to answer an existing poll, enter the ID and click on “Vote on Poll”. Follow the instructions in section 3.2.2. Lastly, you also have the option to edit an existing poll by entering the ID and choosing the third button. For this continue in section 3.2.4

### 3.3.1 Create a Poll

On this page you can create a new poll as shown in figure 2.

The image shows a web interface for creating a poll. At the top is a dark purple header with the word "PollStar" in white. Below the header is a light purple section titled "Create a Poll". This section contains a form with the following elements: a label "Question" above a single-line text input field; a label "Choices:" above three stacked single-line text input fields, each preceded by a "Choice" label; a minus button "-" and a plus button "+" below the choice fields; and a "Submit" button at the bottom.

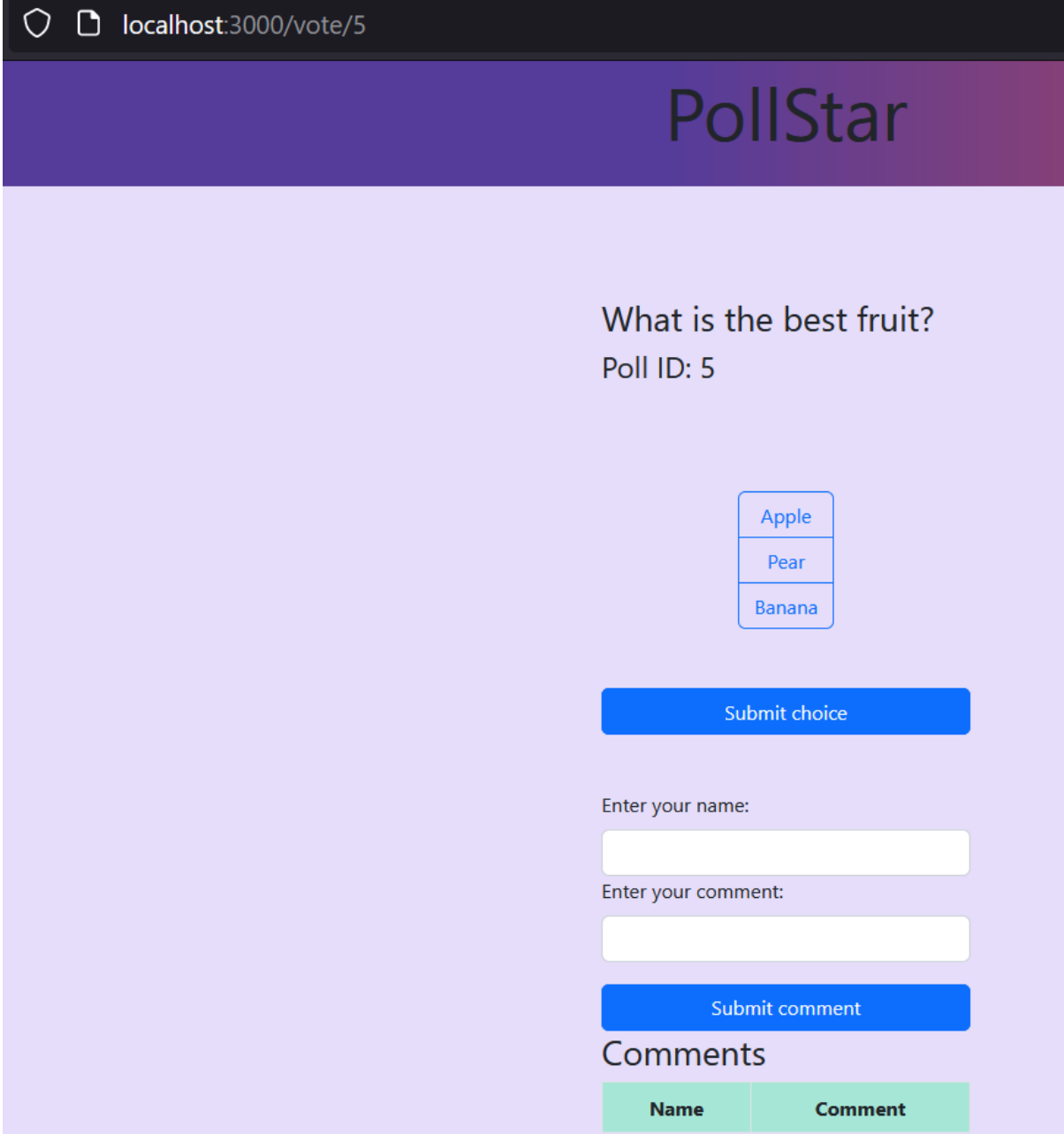
*Figure 2: Create a poll, with options to enter the poll's content.*

Here you enter the question you want the poll to ask in the first field. In the following fields, you enter the options you want people to have when choosing an answer for the poll.

The default is 3 fields for your choices but you can adjust the amount. Pressing the minus button removes one from the choice fields. Pressing the plus-button on the contrary adds a new choice field. You have to have between 2 and 10 choices. When you are satisfied with your input you click "submit", which creates your poll and leads you to its page (Figure 3).

### 3.3.2 Voting on a poll

When you create a new poll you are forwarded to the voting page directly. Otherwise, you have gotten the address directly or the ID and entered it on the front page (Figure 1)



The screenshot shows a web browser at the address `localhost:3000/vote/5`. The page has a purple header with the text "PollStar". The main content area is light purple and contains the poll question "What is the best fruit?" and the poll ID "Poll ID: 5". Below the question are three buttons labeled "Apple", "Pear", and "Banana". A blue button labeled "Submit choice" is positioned below the choices. Underneath is a form for comments, starting with the label "Enter your name:" followed by a text input field, then "Enter your comment:" followed by another text input field. A blue button labeled "Submit comment" is below the comment field. At the bottom, the word "Comments" is displayed above a table with two columns: "Name" and "Comment".

Name	Comment
------	---------

*Figure 3: Displaying a poll. Here is an example containing the question “What is the best fruit?” and three choices: “Apple”, “Pear” and “Banana”. Beneath the Comment area.*

The voting page (Figure 3) shows The selected poll’s question with the poll’s ID below. Thereunder you find the given choices among which you can select one and a button for submitting your choice.

When you want to register your selected answer you click on the one desired by you. This will switch colors between blue and white to further visualize your chosen option. Then you click on “Submit choice” which leads you to the poll result (Figure 4).

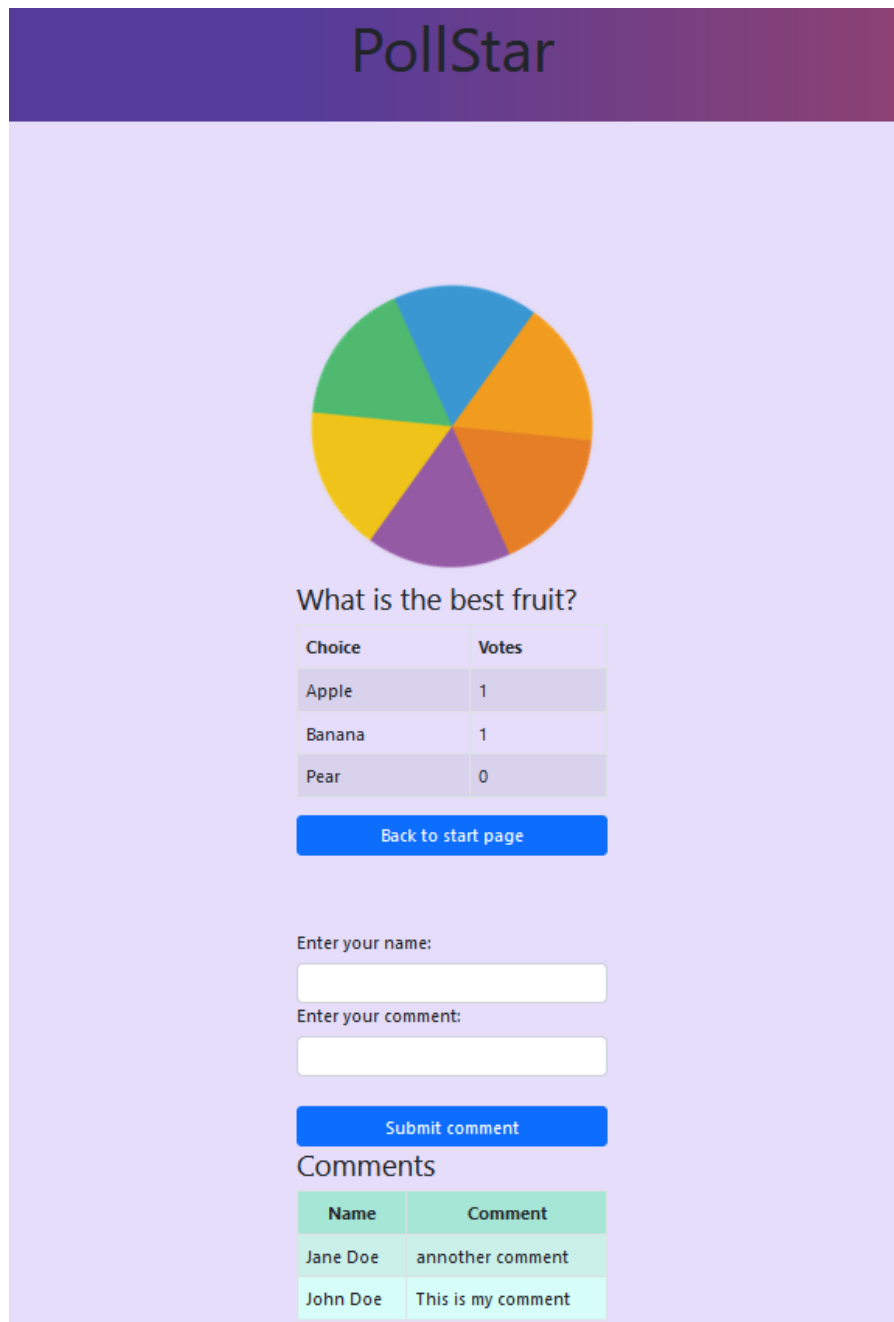
You might want to add a comment to the poll as well. In that case, you can enter your name or pseudonym and what you have to say in the two input fields below and submit your comment. After submitting it it will show up in the Comments table below.

#### 3.3.2.1 Spread a poll

In order to spread a poll you can either spread the full address so that those whom you share the poll with access it directly by link, or you send out the poll's ID. The ID is displayed below the question on the Voting page (figure 3).

### 3.3.3 View results

The section for viewing results also again gives the option to add a comment again and displays submitted comments



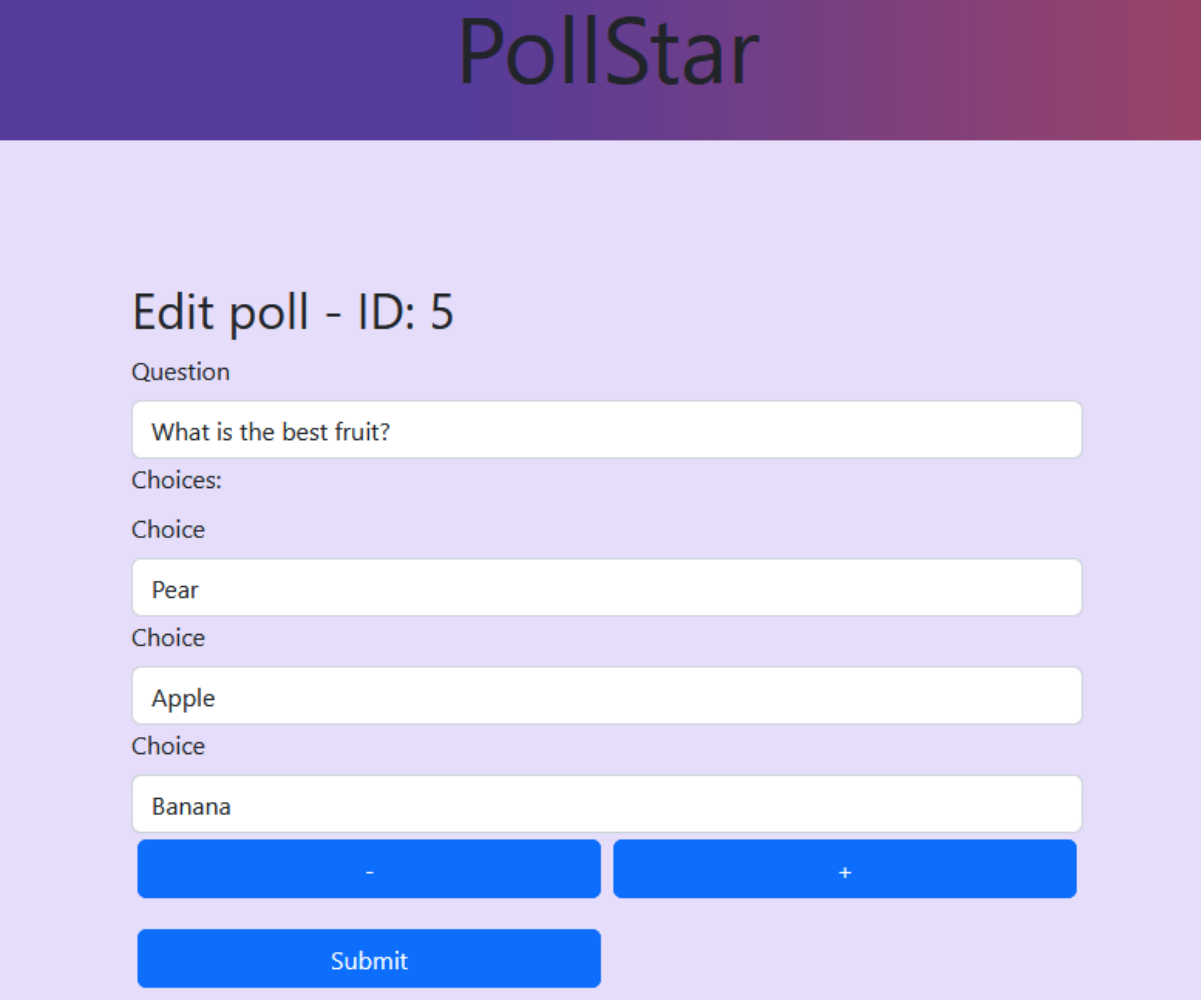
*Figure 4: The result page, showing the polls question, different choices, and how many votes each has received.*

Here you can see the poll's result including your vote as well as all previously done for this poll (Figure 4). Here you again have the option to add a comment and in case there were comments added to the poll they are also displayed beneath the poll result. Moreover, you can go back to the front page (Figure 1) by choosing the "Back to start page" button.



### 3.3.4 Edit an existing poll

When editing a poll (figure 5) you have the similar options as when creating one (figure 2).



The screenshot shows the 'PollStar' logo at the top in a dark purple header. Below it, the page title is 'Edit poll - ID: 5'. The form contains a 'Question' field with the text 'What is the best fruit?'. Underneath, there are three 'Choice' fields with the text 'Pear', 'Apple', and 'Banana'. Between the choice fields are two blue buttons: a minus sign '-' and a plus sign '+'. At the bottom of the form is a large blue 'Submit' button.

*Figure 5: The edit page shows the previously entered question and choices and makes them editable.*

The Question and choices are prefilled with the previously entered text. The question and choices can be changed and choices can be added or taken away. When the input is to your satisfaction you push "submit" and are forwarded to the voting page. See section 3.3.2.

## 4 Design

### 4.1 Technologies

Frontend:

- React
- React Router Dom
- React Bootstrap
- Axios
- Jest
- React DOM

Backend:

- Node
- Express
- PostgreSQL
- Sequelize
- Jest

### 4.2 Construction

#### 4.2.1 Frontend

The frontend was built using a page and component structure. This means that all different pages, with their respective functionality, are defined in their own file. The same goes for smaller, custom components. The pages were built using a mix of Bootstrap components and our custom developed components (which also use some Bootstrap components). Across the application, the Bootstrap components used were forms, buttons and tables.

The figure below (see Figure 5) describes the hierarchy of the application. Inside the App element we define our BrowserRouter from React Router DOM with the different routes for our pages: Create, Vote, Result and Edit. The Vote, Result and Edit routes make use of a URL parameter (`/:id`) to get content for a specific poll. This is done through the use of the `useParams` hook for getting the id and appending it to the axios calls. The `useNavigate` hook is also used to automatically redirect the user to the next page. An example of this is when the user submits a vote on the Voting page and they are redirected to the corresponding Result page for that poll. The Edit page is the Create component with an 'editMode' property

set to true. This could be further refactored to make future changes less likely to break both pages.

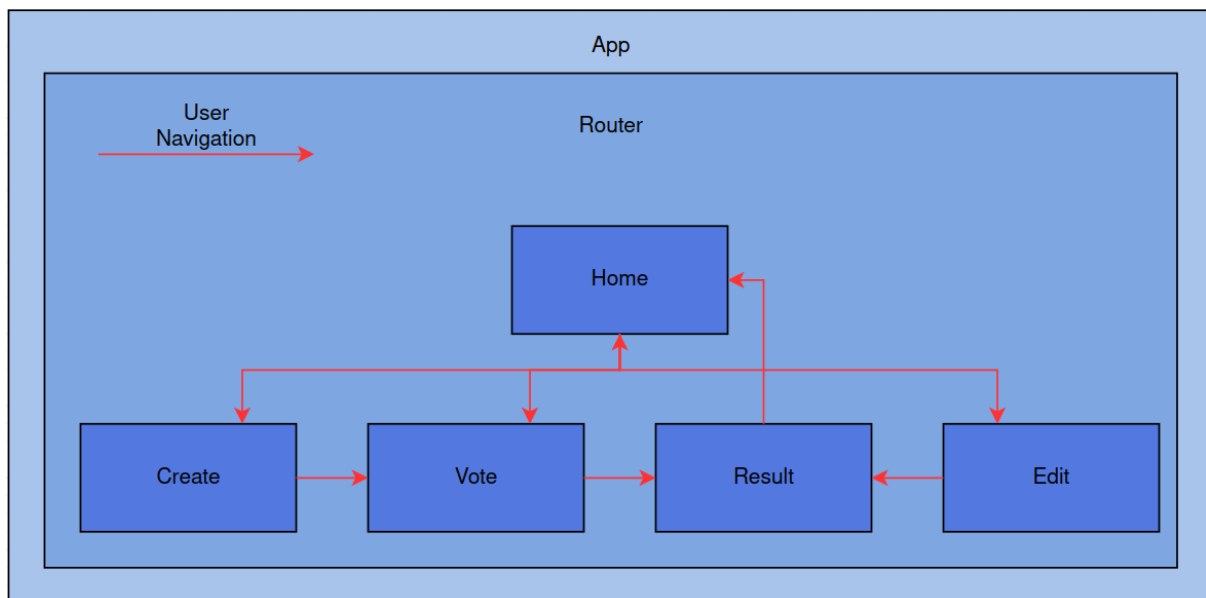


Figure 5: Layout for the pages in the frontend.

#### 4.2.2 Backend

The backend uses the standard setup with an express router that passes requests to a service. The service imports our different models: Poll, TextChoice and Comment. The models extend the Model from sequelize-typescript and are defined in a simple way using annotations (see Figure 6). This allows us to easily query the database to create, update and get our polls (which include the choices and comments). As an example, this is how we get a poll by using an id:

```
const foundPoll = await Poll.findOne({
  where: {id: pollID}, include: [TextChoice, Comment]
});
```

```
import { Table, Model, Column, HasMany } from "sequelize-typescript";
import { TextChoice } from "../choice.model";
import { Comment } from "../comment.model";

@Table
export class Poll extends Model {
  @Column
  question!: string;

  @HasMany(() => TextChoice)
  choices!: TextChoice[];

  @HasMany(() => Comment)
  comments!: Comment[];
}
```

Figure 6: How our Poll class is defined and added to database using annotations

## 4.3 API Specification

In our use cases we require support for a few API requests. We need to be able to create a poll, get it, edit it, and finally vote for a choice in the poll. Our result is a mix of a default route '/' and a route with a dynamic id,('/:id'. All functionality could have been moved to either the default route or the id route since the id can be passed along in the body or in the request parameters. The following requests are possible (the GET for all polls is not used):

Name	Route	Type	Request	Response	Error
Get all polls	/poll	GET	-	200: List of all polls	400: No polls have been created. 500: Server error
Create poll	/poll	POST	{question: string, choices: Array<string>}	201: The created poll	400: Invalid payload. 500: Server error
Vote on poll	/poll	PUT	{pollID: number, choice: number}	201: True	400: Invalid payload. 500: Server error
Get a poll	/poll/:id	GET	URL id parameter	200: The poll with the corresponding id	400: No polls have been created. 500: Server error
Edit a poll	/poll/:id	PUT	URL id parameter, {question: string, choices: Array<TextChoice>}	200: The edited poll	400: Poll was not updated 500: Server error
Add a comment	/poll/:id	POST	URL id parameter, {name: string, text: string}	200: The poll with the comment added	400: Comment was not added 500: Server error

## 4.4 Testing

There is a small amount of testing in the frontend and backend. The testing is done with React DOM and its test utils.

There are currently backend integration tests related to PollService, where all main functionality is tested. Arriving at functional backend integration tests took some time, mainly because of the fact that significant changes were made when implementing the database. Two approaches were quickly attempted. One where we simply tried to use the database as usual (even though this would give us improper unit tests) but did not manage to get it running with the tests (this was later rectified and the tests can now potentially be run

against the production database). The other approach was to mock the database which looked promising at first. We encountered the same problems as others ([Issue 1](#) , [Issue 2](#)) and ended up choosing a simpler approach where we decoupled from our postgres database and used a testing database with sqlite. It seems like the database does not need to be decoupled from the service layer since there is a mode called “validateOnly”, essentially mocking the database. This approach seemed the most promising, however, it does not seem to generate the unique IDs in the same way as when it’s run with a live database. Ultimately we ended up choosing an approach with minimal mocking, running in a non validateOnly mode, and testing against a real sqlite database.

## 5 Responsibilities

In the beginning the work was not split between different people, instead we all developed the same features on our own in order to develop a good understanding of the basics.

Therefore the features stemming from Lab 1 and Lab 2 can not be assigned to specific group members. Instead, we combined the best solutions we developed independently.

Other tasks that were assigned to specific members under the later development:

Group member	GitHub user name	Responsibilities
Ariadna	ariadnadeprado	comments edit page styling
Ben	Powana	implementing the database edit page create page most of the backend backend tests
Christina	chrmeis	<ul style="list-style-type: none"><li>• add and subtract choices</li><li>• structure based on app.tsx,</li><li>• user-friendliness, displaying the needed information for spreading the poll</li><li>• placement of Comments</li><li>• writing report/presentation</li></ul>
Erik	erra-ch	comments result page voting page routing /:id parameter routes, services merging different branches, adding small fixes writing report/presentation styling