Semester 3

Teun Visser

442651

Design Document

Pollstar

Contents

[Project Description 2](#_Toc93096798)

[User Problem: 2](#_Toc93096799)

[Pollstar as solution: 2](#_Toc93096800)

[Description: 2](#_Toc93096801)

[Architecture 3](#_Toc93096802)

[How is Solid Guaranteed: 3](#_Toc93096803)

[Single-Responsibility principle: 3](#_Toc93096804)

[Open–closed principle: 3](#_Toc93096805)

[Liskov substitution principle: 3](#_Toc93096806)

[Interface segregation principle: 3](#_Toc93096807)

[Dependency inversion principle: 3](#_Toc93096808)

[C4 Architecture diagrams 4](#_Toc93096809)

[UML Class diagram 7](#_Toc93096810)

[Design Decisions 8](#_Toc93096811)

[Database 8](#_Toc93096812)

[Front end 8](#_Toc93096813)

[Back end 8](#_Toc93096814)

[Security considerations 8](#_Toc93096815)

[User management Frontend 8](#_Toc93096816)

|  |  |  |
| --- | --- | --- |
| Version | Edits: | Time |
| .1 | Document Creation and setup | 27-09 |
| 1.0 | Actually making the document, filling it with necessary diagrams and related info | 08-10 |
| 2.0 | Solid guarantee updated  *Design decisions* | 14/01 |
|  |  |  |

# Project Description

## 

## User Problem:

Making a decision where there are several options and there is no clear answer, getting input from a lot of users will be a difficult task.

## Pollstar as solution:

A simple question should have a simple answer, set a question and the answers, and let everyone from the crowd pick his or her favorite from your provided options. This was, the most voted option will come forward, together with the info about how many people voted for it. This is also compared to the other options, and comments will be shown from the voters to elaborate on their chosen answers.

## Description:

A tool for users that need the opinion of a large amount of people, like an audience. But also possible for use in group projects, classrooms or events. This can be used for evaluation, but its main focus is to make collaborative decisions.

The application will focus on speed and ease of use, this means people won’t have to create an account to vote, and help making a decision. Instead, you will be directed to the poll immediately and be prompted to vote for your favorite option. Of course, you can only vote once to avoid manipulating results, but a user can always retract his vote and cast it on another option on the poll, as long as the poll isn’t locked.

What other people voted will only be available after you voted, this ensures our users don’t “Go with the pack”, but make a decision for themselves.

There also might be scenarios where you want to limit the sample size for your audience or get an opinion on a time limited issue. For times like these you can *lock* your vote, this will make the poll un-vote-able, meaning that new voters can’t cast a vote and people that already voted on your poll are unable to recast their vote.

These are all precautions to make sure that the pollster gets the fairest and most accurate response.

# Architecture

## How is Solid Guaranteed:

SOLID is about the principles of object-oriented programming

### Single-Responsibility principle:

This has been accounted for since the start of the UML, therefor all classes but also separate applications have one purpose, and they will be built for that purpose only.

### Open–closed principle:

I have to admit that at this point in the application this rule isn’t being upheld that well, once the simple application has been setup however, the UML will be updated and from that point on modification of the application will ensure the persistence of this rule.

### Liskov substitution principle:

This principle is tested in the unit tests to some degree, but is not recorded anywhere else, the UML has been created with this principles in mind.

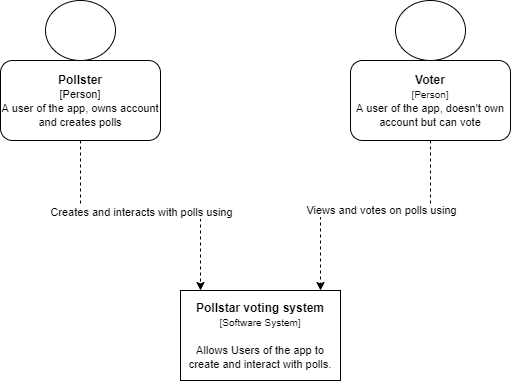
### Interface segregation principle:

I show how I apply this principle in diagram C3, the application consists of several layers, these are defined by those interfaces.

### Dependency inversion principle:

I am unable to prove dependency inversion as of right now, because of the fact that there hasn’t been an opportunity to use it, yet. However I am sure planning on implementing this principle, to ensure nothing failing later.

## C4 Architecture diagrams



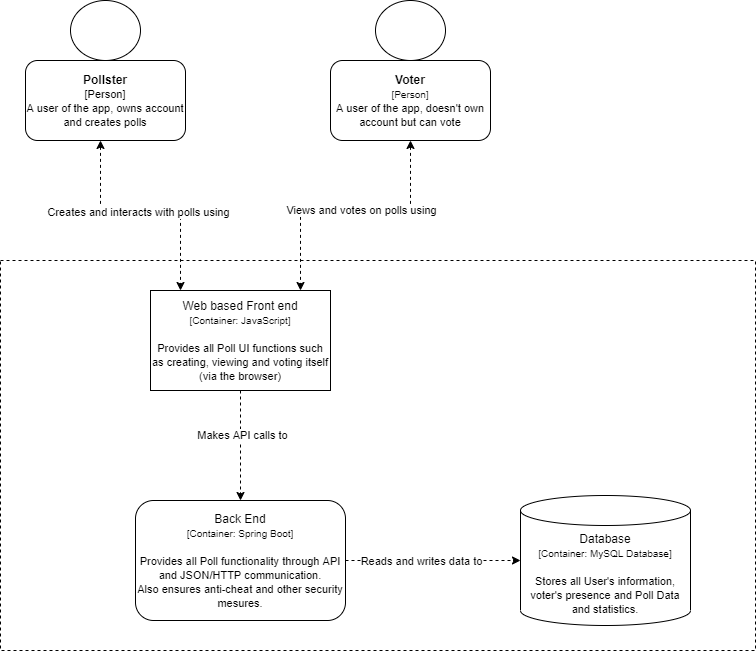
A simple C1 diagram to show user interaction with the application, as shown, there will be two main users that interact with the application:

The Pollster, this user will have to create an account to create polls.

The Voter, this user can vote on polls anonymously

There will also be some users upgraded to Admin, but they are a bit of an anomaly, they can lock and block off existing polls.

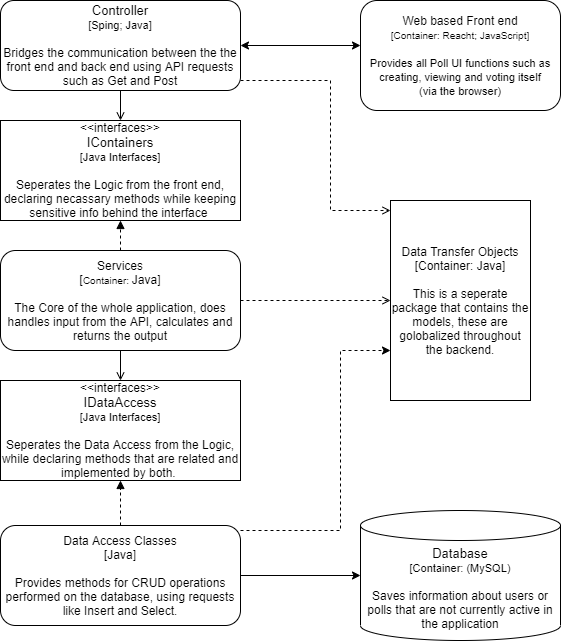
This would make them only available to the pollster that created the poll, so he can edit it and re-publish it again



The C2 Diagram, all the components I am planning to use are listed here.

The arrows represent how data is called and the path it takes from storage to end-user.

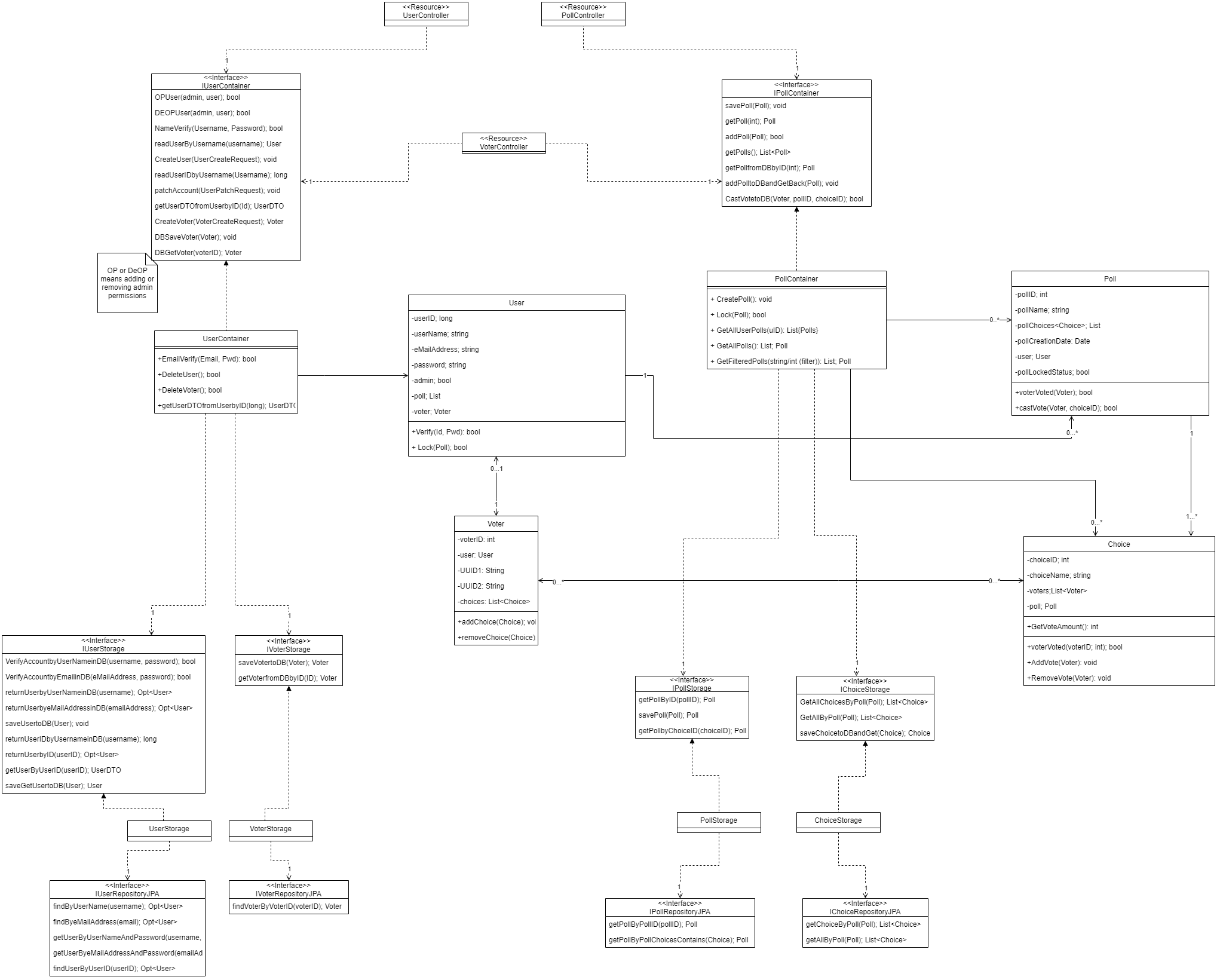
Here in this C3 Part, the data flow in the whole application is explained.



There are multiple ways of tackling one of these diagrams, but showing the components like this helps me define my application the best.

## UML Class diagram

Doubling as a placeholder C4 Diagram but also my UML, my UML:

(it is also included in the documentation folder as Pollstar UML (rev2) if the size is too small)

Construction on this UML has been started quite early in the application, it has been changed a lot and is still subject to change according to the agile way of working :)

## Design Decisions

### Database

The chose database is MySQL; because much of the handled data is intertwined, a linked database would be best suited.

The database is operated in the backend using JPA, this links with the database and automatically manages the tables, in the beginning it was hard to get used too, but a lot of time was gained by automatically handling queries.

Data transfer objects slipped my mind… In all the testing and building I was doing I forgot to implement and use them well. They are now used for keeping sensitive data away from endpoints, but are not well suited combined with JPA. They are not directly faster than the whole object because all the info is pulled from the database, and then formatted to a DTO. It’s fine for this project’s scope, but I will properly reflect on this choice.

### Front end

The front is built using react, it is still giving me some headaches because some solutions are beating around the bush compared to other languages, but it was a good choice because there are good tutorials, and it wasn’t that hard to pick up.

### Back end

Java is comparable to past experiences with coding so picking it up was quite quick, java is also lightweight but has a high scalability. For managing the java, gradle was used, it’s easy to modify and update dependencies. and because of the provided resource of Fontys, setting it up with automated testing. Springboot was used for ease of use in the application, it’s integration with Lombok gives a very organized feeling and this also made implementing the REST Api an easy task.

### Security considerations

JWT is used to verify is a user is logged, who said user is and what he is allowed to do, implementation was quite hard a I was met with a lot of problems while trying to mend it in shape, but there it all went very quickly and makes good sense when I understood the basic concept.

DTO’s are lacking in my application, a giant fail on my part as a developer. I will make sure to use them a little bit to hide sensitive data. But a more thorough implementation would improve security and performance by quite a lot.

Extra:

### User management Frontend

There is only one user in my application, but in the frontend a visitor is given a unique id to identify him, this key is used for identification and to prevent malicious intent on results. It is now handled with a cookie, I think there are certainly better ways, however for the scope of this project I am quite happy with how it performs in my demo-ing.