Report SpringApp Project Group 4

Jacek Antoni Wegrzynowski, Rashaad Wells Iversen, Veronicha C. T. Pettersen

November 26, 2023

Abstract

10-15 lines with the software technology and the highlights from the project that has been undertaken.

1 Introduction

Approximately 1 page on:

- A brief introduction to the prototype implementation and topic of the project. In this project, an IoT-cloud software system has been developed. The system takes feedback from users in the form of yes/no votes on polls. The polls can be voted on either via an web interface or via an IoT-device.
- Discuss (briefly) the technology stack that has been selected, mention related technologies (if relevant), primary arguments for choice of technology stack.

  Here we can maybe add an illustration from the powerpoint
- A brief account of the results that have been obtained in the project.
- A one paragraph overview at the end, explaining how the rest of the report is / has been organised

This rest of this report is organised as follows: Section **??** gives an ....

2 Software Technology Stack

Introduce in (sufficient) depth the key concepts and architecture of the chosen software technologies. As part if this, you may consider using a running example to introduce the technology.

Emphasize the "new" software technologies that was selected by the group and which has not been covered in the course.

This part and other parts of the report probably needs to refer to figures. Figure **??** from [**?**] just illustrates how figure can be included in the report.

2.1 Angular:

We chose Angular as our web application framework to develop the frontend of the FeedApp prototype. Angular is a popular, open source TypeScript based framework that is used to create Single Page Applications (SPAs). SPAs are web applications that only loads one single page, and then changes the content of that page depending on how the user interacts with the page.

Fundamental Consepts of Angular:

Components : components are many places described as the main building blocks for Angular applications. Each component will contain one HTML-file with the UI-template of the component and one TypeScript- file that contains the components logic. It can also contain CSS or SCSS files, defining the styles of the component, as well as test files and configuration files. A component defines a specific view, as well as the functionality that goes into that view. In other words, it contains both what the user sees in the UI and the logic that goes into the component.

Services :It is also possible to share different functions and logic between different components. This can be done by creating a service, which is a TypeScript file that is used for tasks such as for example business logic and handling of data. In our FeedApp implementation, we created services dedicated to managing authentication and handling poll data.

Dependency Injection (DI) : Services can also be used to inject dependencies into multiple components, with the use of DI. This is useful for connecting the different parts of the application.

Routing : The Angular Router handles the navigation between different views as users performs different tasks. This is a key element in SPAs since instead of reloading the page every time the view is changed,

One of the main advantages of creating the application as a SPA is that it speeds up the development process.

Resourses used for writing this paragraph: https://angular.io/guide/architecture

2.2 Spring Boot

We have chosen Spring Boot as our enterprice software framework when developing our application. Spring Boot is an extension of the Spring Framework that simplifies the development process, making it possible to create a functioning web application fast. To fully understand the benefits of the Spring Boot extension, we are first going to talk a little bit about some of Springs main consepts.

Spring Configurations Explained:

Bean Definitions : In Spring, objects managed by the Spring Inversion of Control (IoC) container are referred to as beans. Configurations involve specifying these beans and managing their lifecycle within the application.

Dependency Injection (DI) : Spring's DI mechanism manages dependencies among application components. This setup is crucial for injecting required services or modules into different parts of the application.

Aspect-Oriented Programming (AOP) : Configurations in Spring also include setting up aspects for handling cross-cutting concerns like logging or transaction management.

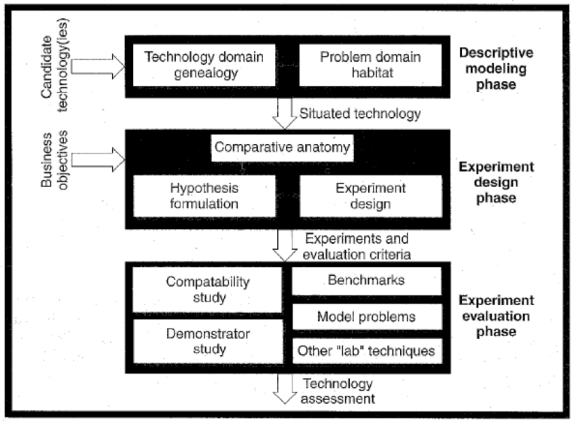

Figure 1: Software technology evaluation framework.

Data Source and Transaction Management : For applications interacting with databases, configurations encompass setting up database connections and managing transactions effectively.

Deploying a Spring Application:

Packaging : The application is compiled and packaged, typically into a JAR or WAR file, ready for deployment.

Running on a Server : The packaged application is deployed on a web server. Spring Boot, with its embedded server capability, simplifies this by allowing the application to run independently without needing a separate server setup.

Spring Boot's Role in FeedApp:

Auto-Configuration : Spring Boot automatically configures the application based on the included libraries, reducing the need for extensive manual configuration.

Simplified Deployment : The embedded server feature of Spring Boot allows our application to be deployed as a standalone unit, enhancing ease of deployment and portability.

In the implementation of our FeedApp prototype, the use of SpringBoots autoconfiguration and deployment functionalities has meade it possible for us to spend more time on the applications buissness logic.

2.3 JSON Web Tokens

2.4 H2

2.5 Hibernate

2.6 Java Persistence API

2.7 Mosquitto MQTT

3 Design of the FeedApp Prototype

3.1 Architectural Overview

Our fundamental idea of the FeedApp application was to mimic some of the Kahoot application capabilities while remaining simplistic, keeping alignment with good practices in software development and prototyping. Our archtiecutre aims to meet requirements, demonstrate core functionality, and balance the effort put into development of a prototype.

The application is accessible through web browsers, ensuring compatibility with devices such as smartphones, tablets, and computers. Key features of the FeedApp are listed below:
(rewrite so that each point is described in a full sentence)

1. Ability to create and particpate in polls
2. Flexibility to cast votes via an IoT device or a web browser.
3. Real-time display of voting results and analytics.
4. Third Party access to data for analytics
5. User Authentication

3.2 Domain Model

Our domain model of the FeedApp, is represented in a Unified Modeling Language (UML) class diagram representing the fundamental objects and their relationships. By doing this we describe behavior and functions in a clear way so that we are all on the same page. Throughout the development lifecycle of the FeedApp, this domain model has undergone iterative modifications. These adaptations were essential to maintain alignment with changes made in our objectives. A primary focus during these modifications has been the preservation of simplicity within the system's architecture. By continuously refining the domain model, we have ensured that the FeedApp remains functionally efficient.

3.2.1 Users

The domain model describes an entity of a user and categorizes it into two distinct types based on account registration status. Users with registered accounts are granted full access within the FeedApp including the capability to create and participate in both public and private polls. This category of user must undergo an authentication process which is described in TODO:Section. Users without registerd accounts are only allowed to participate in publicly accessible polls. This restriction is a delibarate choice, allwoing our development efforts to maintain simpliciy in implementation.

3.2.2 Polls

A key feature of the system is the poll entity, characterized by distinct attributes such as a title, an identification number, and the option to be set as private or public. Each poll is required to have a time limit and are also designed to integrate with Internet of Things (IoT) devices. Furthermore, polls are designed to be flexible, allowing an unrestricted number of questions. The application does not impose a limit on the quantity of questions per poll. In addition to this, for private polls, there is a feature

to specify a list of authorized users who are permitted to vote. In the design of polls, the questions must be structured as binary-choice and closed-ended. This format restricts responses to one of two predetermined options, exemplified by pairs such as True/False, Yes/No, or Pancakes/Waffles. While the application does not impose time limits on individual questions, it integrates time restrictions at the poll level. This design choice, made for ensuring simplicity in implementation, focuses on the entire poll rather than individual questions.

3.2.3 IOTDevice

For this project, a physical IoT device was chosen to add the interactive dimension to the polling process. This device is intentionally simplistic, capable only of transmitting voting data without knowledge of the specific poll or question involved. It operates via a Mosquitto broker, which relays messages from the IoT device to the feed application. The integration with the IoT device is minimalistic. The device's primary function is to acknowledge a voting action and communicate this to the feed application. It is not equipped to discern details about the poll or the voting options. The absence of an on-device display is compensated for by using a command-line interface to indicate when a vote has been cast. This setup was demonstrated in the submitted video, showcasing the Mosquitto broker's role in facilitating communication between the IoT device and the feed application.

3.2.4 IOTDisplay

The project's infrastructure includes an Internet of Things (IoT) device, functioning independently on a dedicated hardware platform. This device establishes a wireless connection with a Windows-based HP laptop, which serves a dual purpose: as a control unit and as a display for the IoT device's operations. The design stipulates that both the IoT device and the laptop must share the same Wi-Fi network for seamless communication.

3.2.5 IOTDevice Integration with FeedApp

Further enhancing the system's architecture is the incorporation of a Linux virtual machine (VM) on the HP laptop. This VM is designated as the application server, managing the core functionalities of the feedback feed application. Notably, the VM's connectivity is not restricted to the same Wi-Fi network as the IoT device, offering flexibility in network configurations. It communicates with the IoT device through a Mosquitto broker, which acts as an intermediary in message transmission. The IP address of the Mosquitto broker is hardcoded into the system for consistent connectivity.

The application leverages a REST API to handle incoming data from the IoT device. When a user interacts with the IoT device, such as by pressing a button, this action triggers a message that is relayed by the Mosquitto broker to the feed application. The application's REST API is configured to recognize this input, determining the active poll and the specific question being addressed by the IoT device. While the mechanism for transitioning between different poll questions was not fully established, the conceptual framework suggests that the REST API would facilitate this progression by incrementally recording votes and navigating to subsequent questions.

3.2.6 Analytics

Once the a vote is cast on the last quesiton in the poll, the application is designed to immediately reflect the analytical data offering insight into collective responses. The system is designed to facilitate the display of analytics for all polls, regardless of their current status (active or inactive). This

functionality is implemented through a standardized display template, which is accessible via web browsers. The template ensures uniformity in the visualization of analytics, providing a coherent and consistent user experience. In addtion, the system's architecture allows for third-party applications to access poll-related data. This capability enables external entities to conduct their own analytical assessments. However, it is important to note that the scope of data accessible to third-party applications is confined to poll-level information. Granular data pertaining to individual questions within the polls is not available for external analysis.

3.3 FrontEnd Design

To get an overview of how we wanted to implement the user interface, an application flow diagram was modeled. The diagram displays how the user navigates through the different frames in the front end of the application:

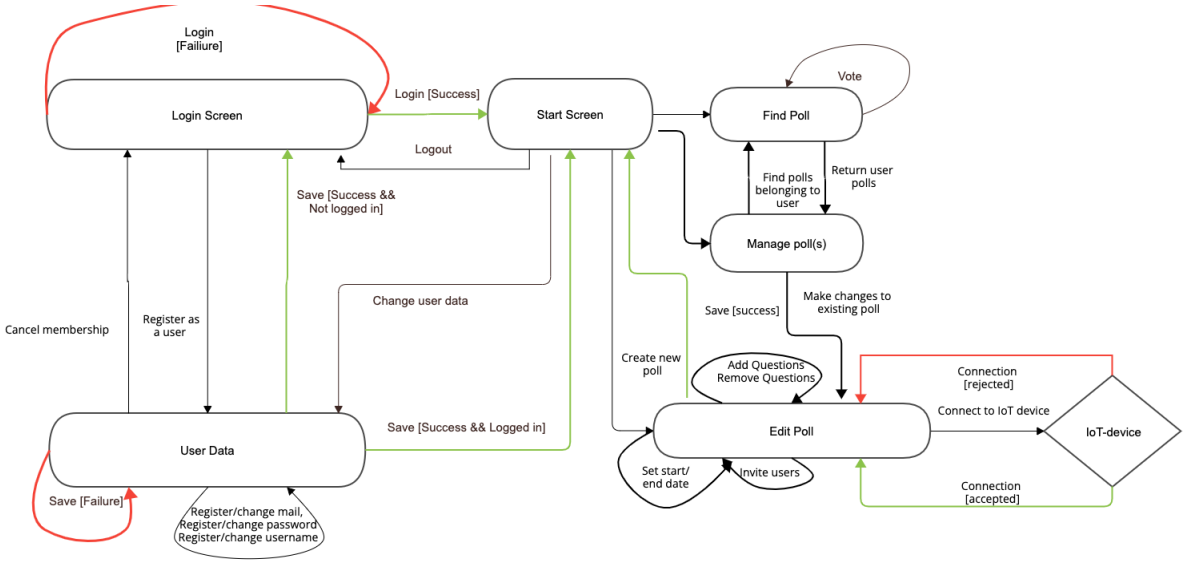

Figure 2: Application Flow Diagram

Each screen has in the diagram been modelled as a state, and for every state, the user is able to do some action or provide some input. These actions or inputs are modelled as transitions, and are in the figure above illustrated with arrows. Transitions that results in errors are colored red, and transitions that does not result in errors are colored green. Our application consists of six screens. We have a login screen, where the user can either login, or create a new user. Once the user is logged in, he is directed to a start screen, where he gets the option to do multiple actions. He can search for a poll, which then can be voted on, or he can view and manage his own polls. He can create new polls, and in this state he gets the option to pair the poll with an IoT device.

The example below shows how you may include code. There are similar styles for many other langages - in case you do not use Java in your project. You can wrap the listing into a figure in case you need to refer to it. How to create a figure was shown in Section **??**.

```
1 public class BoksVolum {
2
3   public static void main(String[] args) {
```

```
4
5       int b, h, d;
6       String btext, htext, dtext;
7
8       [ ... ]
9
10      int volum = b * h * d;
11
12      String respons =
13            "Volum [" + htext + "," + btext + "," + dtext + "] = " + volum;
14
15    }
16  }
```

4 Prototype Implementation

4.1 Frontend

The following components has been implemented for the front end:

- Login: this is the first view that the user are presented with. It's main purpose is to authenticate users, and help them access the application. It contains input fields for username and password. When the user presses "Log in", he is taken to the main-page of the application. If the user is not registered, he can easily do so by pressing the "Register"- button.
- Register: here the user is presented a schema, where he can register and that way access the application. He needs to add an email address, a username and a password. Once the information is submitted, the user is sent back to the login page.
- Main-page: This component contains buttons that lets the user easily navigate to all parts of the application quickly.
- Find-poll: Here the user can search for a poll via its id, or by a poll-name. All the polls that matches the input will then be displayed with the option to vote on it.
- Create-poll: Here the user can create a new poll. He can decide the questions that should be displayed, when the poll is active, if it is private and invite users to participate in it. Once created, a poll-id is given to the user. This can for example be given to other users and used to search for the poll.
- Vote: Here the user gets to submit votes to active polls.

A authentication service, a poll service and a voting service has also been created to handle logic that can be applied to multiple components. The authentication service handles operations such as logging the user in to the application and searching for users. The poll service handles logic depicting the polls such as finding polls, creating polls and changing polls. The Voting service handles the logic connected to the voting.

There are also a few features that we modelled into our application in the first phase of this project, but due to the time constraints are yet to be implemented. These include Poll management and User

Config	Property	States	Edges	Peak	E-Time	C-Time	T-Time
22-2	A	7,944	22,419	6.6 %	7 ms	42.9%	485.7%
22-2	A	7,944	22,419	6.6 %	7 ms	42.9%	471.4%
30-2	B	14,672	41,611	4.9 %	14 ms	42.9%	464.3%
30-2	C	14,672	41,611	4.9 %	15 ms	40.0%	420.0%
10-3	D	24,052	98,671	19.8 %	35 ms	31.4%	285.7%
10-3	E	24,052	98,671	19.8 %	35 ms	34.3%	308.6%

Table 1: Selected experimental results on the communication protocol example.

management. Currently the application allows users to create and participate in polls. However, the management of the polls is not yet functional. The same goes for the management of the user settings. The buttons has been created in the main-page component, but they are not directing the users to new views yet.

5 Test-bed Environment and Experiments

About 2 pages that:

Explains how the prototype has been tested the test-bed environment.

Explains what experiments have been done and the results.

For some reports you may have to include a table with experimental results are other kinds of tables that for instance compares technologies. Table **??** gives an example of how to create a table.

6 Conclusions

Concludes on the project, including the technology, its maturity, learning curve, and quality of the documentation.

The references used throughput the report should constitute a well chosen set of references, suitable for someone interesting in learning about the technology.