Remarks

19: ‘nimda’

19: Może: *Second type*?

SILESIAN UNIVERSITY OF TECHNOLOGY

FACULTY OF AUTOMATIC CONTROL, ELECTRONICS, AND COMPUTER SCIENCE

Engineer thesis

Central online voting system

author: Wojciech Drzewiecki

supervisor: Krzysztof Simiński, PhD DSc

Gliwice, February 2020

załącznik nr 2 do zarz. nr 97/08/09

Oświadczenie

Wyrażam zgodę / Nie wyrażam zgody* na udostępnienie mojej pracy dyplomowej / rozprawy doktorskiej*.

Gliwice, dnia 5 lutego 2020

.....
(podpis)

.....
(poświadczenie wiarygodności
podpisu przez Dziekanat)

* podkreślić właściwe

Oświadczenie promotora

Oświadczam, że praca „Central online voting system” spełnia wymagania formalne pracy dyplomowej inżynierskiej.

Gliwice, dnia 5 lutego 2020

.....

(podpis promotora)

Contents

1 Introduction **1**

2 Online voting systems **3**

2.1 Introduction 3

2.2 Ideal voting system 4

2.3 Overview of similar solutions 5

2.3.1 Electronic voting in Estonia 5

3 Requirements and tools **7**

3.1 Functional requirements 7

3.2 Nonfunctional requirements 8

3.3 Use cases 9

3.3.1 Citizen use case UML 9

3.3.2 Ward administrator use case UML 10

3.3.3 Committee administrator use case UML 10

3.3.4 Administrator use case UML 11

3.4 Description of tools 12

3.4.1 Java 12

3.4.2 Maven 12

3.4.3 Spring 13

3.4.4 Thymeleaf 14

3.5 Methodology of design and implementation 14

4 External specification **17**

4.1 Types of users and cockpits 17

4.1.1 Overview . . . 17
4.1.2 Voter cockpit . . . 17
4.1.3 Administrator cockpit . . . 18
4.1.4 Ward administrator cockpit . . . 18
4.1.5 Committee administrator cockpit . . . 18
4.2 Software requirements . . . 18
4.3 Installation procedure . . . 18
4.4 Activation procedure . . . 19
4.5 User manual . . . 19
4.5.1 Voter manual . . . 19
4.5.2 Committee administrator manual . . . 20
4.5.3 Ward administrator manual . . . 20
4.6 System administration . . . 21
4.6.1 Add country . . . 21
4.6.2 Add district . . . 21
4.6.3 Add ward . . . 21
4.6.4 Grant ward administrator privileges . . . 21
4.6.5 Add committee . . . 21
4.6.6 Grant committee administrator privileges . . . 21
4.6.7 Add poll . . . 22
4.6.8 Register candidate . . . 22
4.6.9 Create election . . . 23
4.6.10 Polls . . . 23
4.7 Security issues . . . 23

5 Internal specification **25**
5.1 Concept of the system . . . 25
5.2 System architecture . . . 25
5.3 Data structure . . . 27
5.4 Implementation of the application . . . 27
5.4.1 Entity . . . 27
5.4.2 Repository . . . 28
5.4.3 Component . . . 28

5.4.4 Controller 29
5.4.5 View template 29
5.5 Applied design patterns 29

6 Verification and validation **31**
6.1 Testing paradigm 31
6.1.1 Planning 31
6.1.2 Implementation 32
6.1.3 Testing 32
6.2 Testing scope 33
6.3 Test cases 33
6.3.1 Voting test 33
6.3.2 Closed list order test 33
6.3.3 Sending ward protocol test 34
6.3.4 Calculating poll results test 34

7 Conclusions **35**

Chapter 1

Introduction

According to "V-Dem Annual Democracy Report 2019", prepared by Department of Political Science, University of Gothenburg, "Democracy is still the most common type of regime [in the world]" [6]. In the advanced stage of internet age that we are living in right now, it is weird, that almost all democratic countries carry out their elections in voting wards. As the society becomes more and more up to date with the latest technical innovations, there will be a point in the future, when some kind of remote voting becomes the most used voting method across the globe. For now the only technology that is available to vast majority of developed countries and may be considered as help in elections is the internet.

Objective of this thesis is to focus on functionalities and features of voting system capable of delivering safe and easy to use online voting. The solution has three core features: individual online voting, ward protocol sending and calculating results of an election. Every functionality is accessible through web browser communicating with application server which is processing the requests. All necessary data is held in a database and fetched when necessary. That is the most common solution available to us at the moment.

First chapter of this thesis is dedicated to problem analysis. I focus on an idea of the perfect online voting system and review similar existing solutions. In the next chapter I state functional requirements, nonfunctional requirements and few possible use cases of an online voting system. I also describe tools used in the application and propose their alternatives. After that, I focus on external

and internal specification of the application itself. External specification includes back to back guidelines for the user of the application, while internal specification includes information about implementation and overview of the backend of the system. I finish the thesis with verification and validation of the application and short conclusion chapter.

Chapter 2

Online voting systems

2.1 Introduction

Ever since internet was introduced in our lives, online election voting was a contentious issue.

First of all, online voting would be convenient—right now voting is associated with going to a nearby ward, waiting in a queue, complicated procedure of receiving a ballot and filling a ballot. Moreover, waiting for results could be decreased—now we have to wait for all the votes to be counted, protocols sent to commissions, summed up, and determination of the winners. All this can take over few dozen hours. Online voting would decrease that time hugely, at the same time being much more precise than counting by a person.

On the other hand, there is significant trust issue. There will always be a person which have access to data, and with such access that person could change elections results in a matter of seconds. Following this lead, even if nobody has access to data, we will never be sure that vote we cast is not changed somewhere in the logic of the application. We can eliminate that by developing open source application, which can be verified that it does not change something in the logic. This approach would create many more problems, of which two are: firstly, open source application can be investigated by everyone, and with given source it is much easier to find a hole in the logic from which we can benefit. Secondly, even if we see view of that open source application, we can never be sure that logic

behind it is not swapped.

You can see a pattern here—for every argument there is a counter argument, which generates another problems. There is no right answer in this debate.

2.2 Ideal voting system

So what would be features of the perfect voting system? Let's investigate it along with flow of online voting.

Before election, system administrator has to enter committees and candidates data into a system, register them for polls predicted for certain date, which he has entered at the beginning. Although he could handle registering candidates to certain committees, he cannot handle scope of managing whole committee during election. It is best if administrator granted some chosen user privileges to manage certain committee inside the system—let's call him committee administrator. It is useful for example in case of allocation seats via D'Hondt method—it is a committee itself that wants to decide on order in closed list, there should be no third party involved.

First, in order to cast a vote through internet, a citizen would have to register. What online voting system wants to achieve for citizens is to be able to cast a vote without leaving home. It makes sense that citizens would register also through the internet—for example in case of some injury, if someone could go out of home, they could also vote in a ward. It creates a problem—how do we know that person on the other side of the screen is who he says he is. Solution is two step authentication process—citizen registers with citizen id (for example PESEL in Poland), he is sent a text message or an email, but also receives registered letter of registration, both with activation codes, that only together allows him to register properly. Registered letter allows to verify citizen willing to vote through internet without need of leaving the house. Because email can be generated without interference of any third party person, code sent this way secures citizen from someone taking advantage of code in letter. Those two ways combined secure citizens account from being taken advantage of.

Let's say a citizen registered successfully. On election day, he should be able to vote on exactly the same polls as he would personally, in ward. Citizen should be

able to cast a vote only once, and his vote should be detached from his account—in no way should anybody be able to tell which vote belongs to which citizen. Although registered for online voting, a citizen should still be able to vote personally in ward. However, once he voted in any of two ways, the second one should be automatically and immediately blocked. This requires for the system to work also in wards. Once user votes online or receives a ballot from commission in a ward, he should be immediately blocked for second type of voting.

Citizen easily and securely voted either through internet or personally. Now his journey ends, he can go back to living his life. What's left to do is to calculate results of election. However, system also has to take into account all votes casted in wards. Results can be calculated if and only if all wards have entered their protocols into the system. To do so, a member of a commission should be assigned with access to send the protocol to the system—let's call him a ward administrator. Once that member is a registered user, he should be able to be assigned as particular ward admin by the system administrator. Then, he should be able to file in the protocol, once the election is closed for voting.

After closing the election and collecting protocols from all wards, results of the election can be calculated. This information can be public and delivered to public on the same site that they voted on, as well as on official government electoral commission website.

2.3 Overview of similar solutions

2.3.1 Electronic voting in Estonia

In Estonia, voting is based on electronic citizen ID. It is mandatory and sufficient national identity document, which allows for secure remote authentication. To cast a vote, an Estonian needs a computer connected to the internet and equipped with card reader. They can authenticate using digital certificate included in their citizen card, and cast a vote on the internet [3].

This solution is problematic for several reasons:

- It is impossible to be easily implemented in countries without electronic citizen ID. Providing such documents for most of society of the country is a

long time process, measured in years rather than months.

- A card reader capable of reading such a card is not a common device—a citizen has to go somewhere with a card reader, if he wants to cast a vote.
- Electronic citizen ID is not an intellectual knowledge like password or token. It is a physical device that can be easily stolen—and of course blocked in some department—but it may be too late and someone may have already taken advantage from owning someone else's citizen ID.

Chapter 3

Requirements and tools

3.1 Functional requirements

To better understand scope of the project, we define functional requirements:

- A citizen of a country that holds an online election in the application should be able to register.
- The application should allow registered citizens to vote on candidates of their preference.
- A vote of a citizen should be taken into account when calculating results of an election.
- A citizen should be able to cast only one vote per election.
- A vote of a citizen should not be trackable—no person should be able to tell which citizen voted on which candidate.
- A person designated as a ward administrator—and only that person—should be able to send a ward protocol to the system.
- Protocols sent from wards should be taken into account when calculating results of an election.

- A person designated as a commission administrator—and only that person—should be able to enter a commission's closed list candidates order to the system.
- An administrator of the application should be able to perform steps that lead to creation of an election identical to one provided to him.
- An administrator should be able to trigger calculation of the results of an election.
- Election results should not be calculated with some of ward protocols missing.
- An administrator should be able to grant privileges of ward and commission administrators to designated people separately.
- An administrator should not be able to perform actions in replacement of a ward or committee administrator.

3.2 Nonfunctional requirements

After reviewing functional requirements, we can define nonfunctional requirements:

- User interface should be understandable to English speaking individuals.
- The application should be implemented in Java, with help of Spring Boot and related frameworks for an backend, and Thymeleaf template engine for an frontend.
- The application should store all necessary data in a MySQL database.
- The application should hash all sensitive data stored in a database.
- The application should offer all available functionalities on Google Chrome version 79.

3.3 Use cases

3.3.1 Citizen use case UML

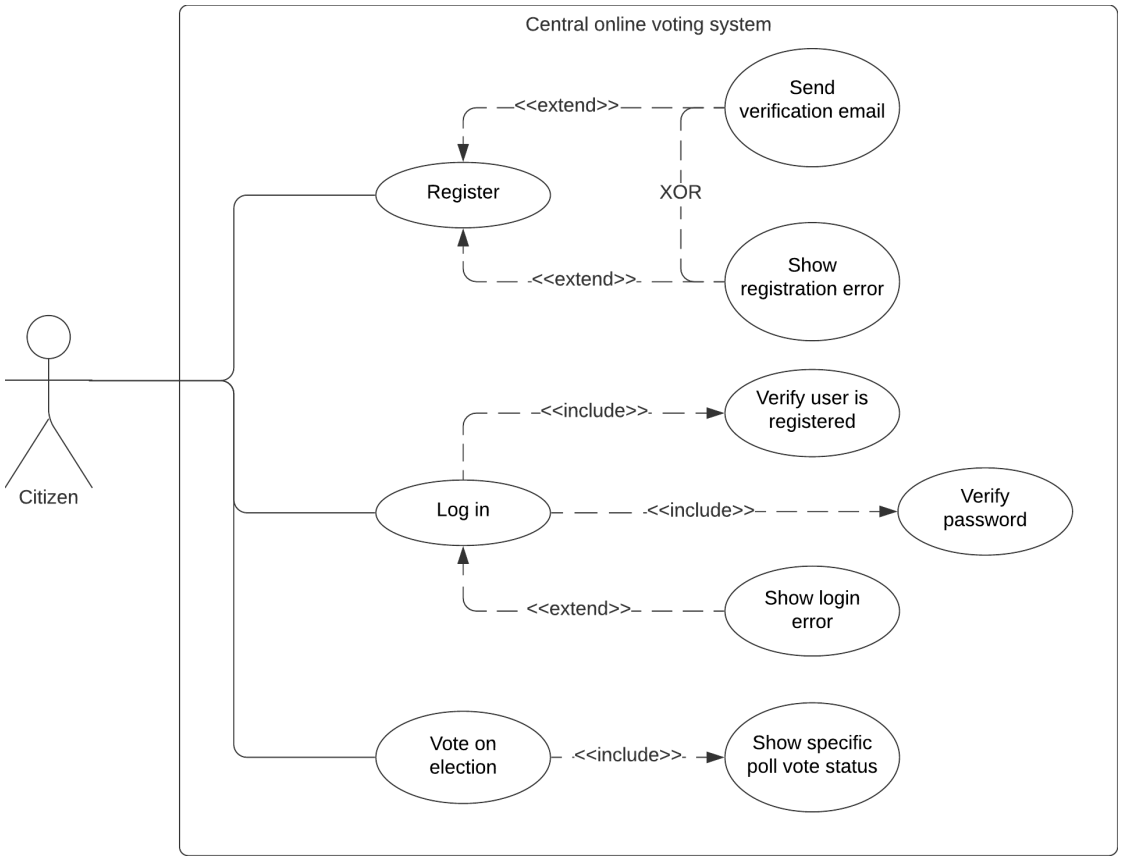

Figure 3.1: Citizen use case UML for Central online voting system

Figure 3.1 shows available use cases for a citizen. He can register and in that case he will receive an email, or he can see some error regarding registration. A citizen can try to login, which will trigger checks if he is registered and that password matches. In case of any errors, he will see a message. A citizen can vote, and after that he will see status of his vote regarding every poll in the election, no matter if vote is valid or not. However, even if a citizen casted an invalid vote, he is not able to vote again.

3.3.2 Ward administrator use case UML

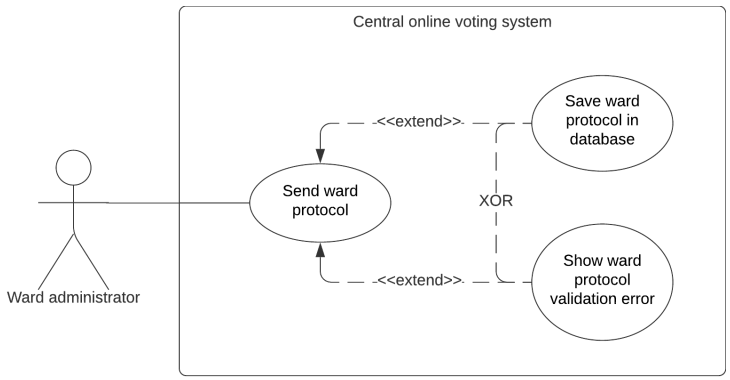

Figure 3.2: Ward administrator use case UML for Central online voting system

Figure 3.2 shows available use cases for a ward administrator. A ward administrator can try to send a ward protocol. If anything is not correct in a protocol he tried to send, he will see a validation error, and the protocol will not be sent.

3.3.3 Committee administrator use case UML

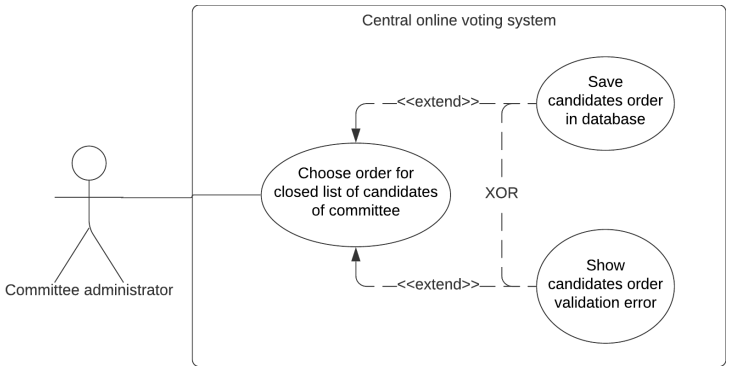

Figure 3.3: Committee administrator use case UML for Central online voting system

Figure 3.3 shows available use cases for a committee administrator. A committee administrator can send closed list candidates' order. If anything is not correct in an order of candidates, he will see a validation error, and the candidates order will not be saved.

3.3.4 Administrator use case UML

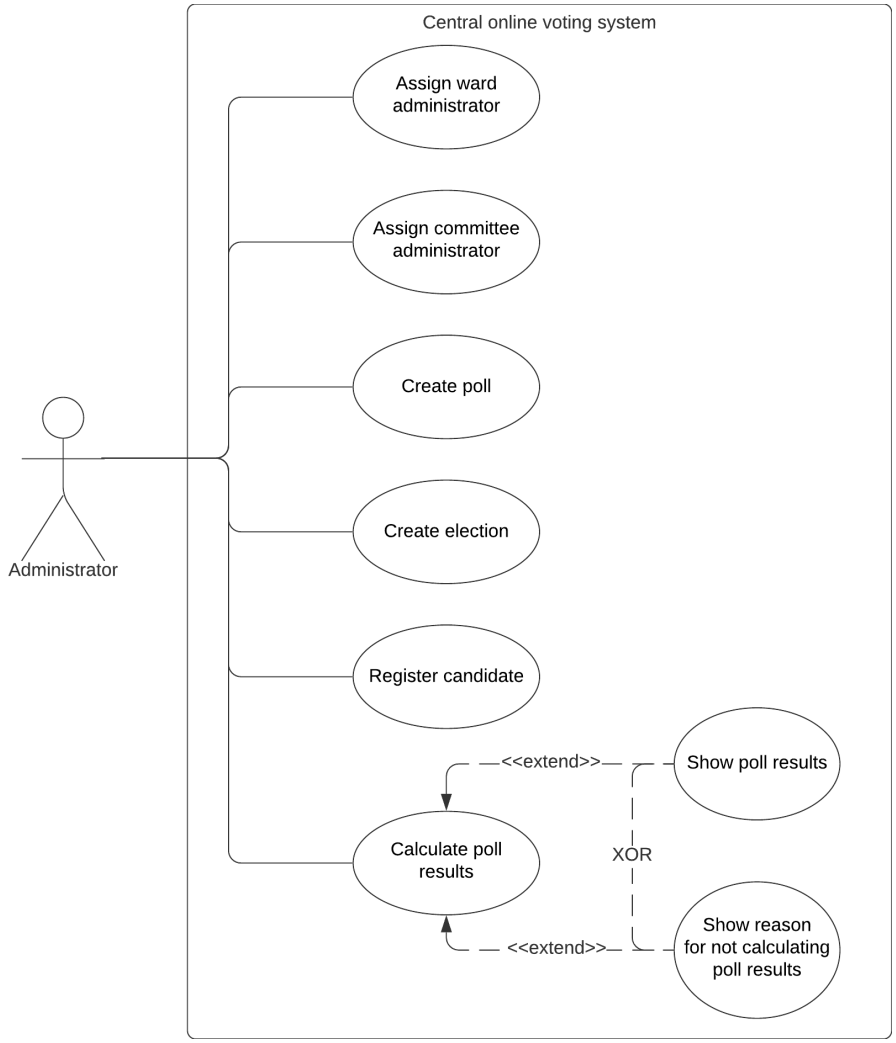

Figure 3.4: Administrator use case UML for Central online voting system

Figure 3.4 shows available use cases for an administrator. We assume that an administrator knows what he is doing, so there is no need to validate his choices. However, he has no power over how many protocols are sent from wards when he wants to calculate poll results. Thus he will get appropriate message when he tries to calculate results of poll that is not ready to be calculated.

3.4 Description of tools

The application is developed in Java programming language with Spring Framework and Thymeleaf as template engine. Data is stored in a MySQL database, and whole application is built with Apache Maven build automation tool.

3.4.1 Java

It is an object–oriented programming language that requires Java Virtual Machine to run a program. This allows Java applications to run on almost every device, regardless of it's operational system [4]. Java offers huge open–source ecosystem.

Alternative

C# is also an object–oriented programming language, but it is used to develop software for Microsoft platforms [12].

Although there is no much difference in beginner level development between Java and C#, I find C# less convenient than Java, because I use machine with macOS operating system as my everyday work tool, and Java has huge support on Unix based system, compared to almost none support for C#.

3.4.2 Maven

Apache Maven is a tool that is capable of building and managing any Java-based project. It helps developers with dependency management, as it is able to download all necessary dependencies with almost no participation of developer. It is easy for beginners, as it is build lifecycles and syntax is easy to understand—all you need is artifact info of dependency that you want to include in your project, and it will download and include dependency in project during install build lifecycle [2].

Alternatives

Gradle It is also application build tool for Java-based projects. It is better than Maven for larger projects, thanks to better automation methods and better performance. Gradle is built with more experienced and demanding developer in mind [5].

Ant Ant is a powerful tool that allows developer to automate very complicated task, but this functionality is not cheap. It is a complicated to configure and it is being ousted by Gradle [1].

I chose Maven because I wanted to learn modern build technology that is common in the industry, but is suitable for beginners.

3.4.3 Spring

It is an open–source framework for Java platform. It focuses on delivering out–of–the–box core features for any kind of application, so developers can focus on business logic of the application. Aside from Spring Core, which provides easy dependency injection, Spring provides extensions in form of more or less independent modules, each available as it's own maven artifact [8]. Spring Boot is an extension of Spring framework, which provides out–of–the–box configuration, and requires user to provide only application specific configuration.

Alternative

Because Spring is open–source framework, considered a standard in the industry for over 10 years, there is no other framework that is as feature–packed as Spring. One could make a case for .Net framework for C# language, but it is developed and directed by Microsoft [12], whereas Spring is powered by community, and contains features required by that same community.

Spring Security

Authentication and Authorization framework for Spring based applications. Provides quick protection of API exposed by application and integration with users in database. Allows for assigning users' roles, and managing accesses based on these roles [9].

Spring Web MVC

It is designed around front controller pattern, and helps developers follow the Model–View–Controller design pattern. It coordinates flow of request processing performed by configurable delegate components [10].

Spring Data JPA

Spring implementation of JPA contract, powered by Hibernate. Reduces hugely boilerplate code that's associated with querying database, especially with Repository interface custom finder methods [7].

3.4.4 Thymeleaf

Modern server-side Java template engine resolver. It converts HTML files with Thymeleaf–specific dynamic attributes based on model attributes provided by developer through Spring MVC framework [11].

Alternative

There are not many pure alternatives when it comes to template resolvers. Java Enterprise Edition comes with JSP, which is not pleasant to use, and is considered obsolete. It definitely does not provide level of integration with Spring MVC framework as much as Thymeleaf does.

3.5 Methodology of design and implementation

Base of every application is a well designed database schema. Once schema is designed with required functionalities of application in mind, and with space for

extensibility predicted, it makes development not only go faster and more efficient, but also more pleasant for a developer.

With proper database schema, we have to consider what is the best way for user to use our application—through web browser or application? Because application is meant to be nation–wide, and there is a web browser on perhaps every computer with internet connection—which is necessary for application to work—web application is the way to go.

When designing web application, we have to decide how we want to achieve end result—presenting view to application user. There are two most common approaches when it comes to Spring application:

- Exposing REST API from Spring application, that contains all of business logic and create another application, that consumes that API and interacts with the user in any way desired.
- Keep view rendering on server side and expose pure html to the user.

I chose the second option, because it is not the most important thing for Central online voting system to be pretty and responsive, I wanted to focus on a backend side of the application.

Once view presenting is established, we want to consider how we want to design architecture of the application. I can see two approaches in choosing the architecture: basing on application requirements or basing on technology used in application. Because Spring provides a whole framework dedicated to the MVC design pattern, and also it is widely used in the industry, I chose to develop the application in that manner rather than choosing architecture basing on application requirements. My application is not large enough to be hugely impacted by bad architecture. I could only harm my project by using some fancy architecture.

Chapter 4

External specification

4.1 Types of users and cockpits

4.1.1 Overview

In order to secure the application from unauthorized access I divided the application into four main cockpits:

- Voter cockpit,
- Administrator cockpit,
- Ward administrator cockpit, and
- Committee administrator cockpit.

Each cockpit requires a specific user role to access it's website. In order to have access to anything we have to be an authenticated (active, successfully registered) user with assigned role that gives us permission for certain cockpits. A guest—not registered user—is only allowed to register or login.

4.1.2 Voter cockpit

A user with a role of a voter is able to vote for election and see election results after the election is closed. In case of voting a user is be able to vote on a poll only once without any exception.

4.1.3 Administrator cockpit

A user with a role of an administrator is able to create essential data—wards, polls, committees etc. as well as granting certain roles to certain users. An administrator is not be able to interfere in a ward or committee administration other than choosing its administrator. Moreover administrator is the one that triggers polls results calculation.

4.1.4 Ward administrator cockpit

A user with a role of a ward administrator is able to manage only his assigned ward. Under no circumstances he is able to manage other wards. In a ward administrator cockpit he is able to enter a protocol from a ward for each poll during an election.

4.1.5 Committee administrator cockpit

A user with a role of a committee administrator is able to manage only his assigned committee. Under no circumstances he is able to manage other committees. In a committee administrator cockpit he is able to choose an order of committee candidates in a closed list.

4.2 Software requirements

In order to run the application, you have to have Java 12, Maven 3.6.X installed on your computer, make sure that `java` and `mvn` executables are in PATH environment variable.

4.3 Installation procedure

In order to build project, you have to execute Maven install lifecycle `mvn install` in `APP_ROOT` directory. This lifecycle will generate `ballotbox-0.0.1-SNAPSHOT.jar` jar file in `APP_ROOT/target` directory. To run application you

have to execute `java -jar APP_ROOT/target/ballotbox-0.0.1-SNAPSHOT.jar`. Application server will listen on PORT 8080 by default.

4.4 Activation procedure

Users' roles definitions are loaded into a database on first server startup if they do not exist in the database already. Same behaviour with the default admin user with password 'nimda' ['nimda']—it is in the database on the first startup. To secure the application you have to register a proper user which is meant to be an administrator, and execute following query in the database:

```
INSERT INTO users_roles(user_id, role_id)values (ADM_ID, ROLE_ID);
```

Where ADM_ID is an id of a desired administrator in table 'user', and ROLE_ID is an id of a row with a column name equal to `"ROLE_ADMIN"` in table 'role'—1 by default. That is the procedure you should follow when you want to add an administrator to the system. After that you should delete the predefined admin user from table user, with following query:

```
DELETE FROM user WHERE username='admin';
```

4.5 User manual

4.5.1 Voter manual

A Voter can access voting page at the root endpoint of the application—HOST:PORT. After clicking show all elections , a user can choose election on which he wishes to vote in. After clicking on vote button, he can cast a vote in all polls available in this election. There are two types of voting methods. One is a single choice vote—voter can mark a checkbox next to candidate he wishes to vote on.

/////single choice vote screenshot/////

Second one [Może: *Second type*?] is a preference voting, in which voter can mark candidates by his choice of preference: 1 for a candidate of his first choice, 2 for his second favourite candidate and so on.

/////preference vote screenshot/////

Description provides user with specific instructions on how to vote on a specific poll. If user does not follow the instructions, vote is invalidated, and voter cannot vote again. After casting a vote, user gets information on status of his votes - if vote is validated or not, and if not, why.

/////polls statuses screenshot/////

After election is finished and results are calculated, voter can see results of poll after clicking on results button. He will get information about how many valid votes were casted in a poll, and who are the winners of that poll.

/////poll results screenshot/////

4.5.2 Committee administrator manual

Committee administrator can access committee management page after going to HOST:PORT/committeepanel endpoint of the application. After that, he can choose which committee that he is an administrator of he wishes to manage. Then, he can set closed list order of that committee, required for D'Hondt method.

/////closed list order screenshot/////

If he makes any error, he will be displayed appropriate message, and order will not be saved—it will be held at previous stage.

/////closed list order result screenshot/////

4.5.3 Ward administrator manual

Ward administrator can access ward management page after going to HOST:PORT/wardpanel endpoint of the application. After that, he can choose which ward thet he is an administrator of he wishes to manage. Then, he can send specific poll protocol from that ward.

/////ward management page screenshot///// /////poll protocol sending page screenshot/////

If administrator makes any error in protocol, he will be displayed appropriate message, and protocol will not be sent.

4.6 System administration

In order to access administrator cockpit, we should go under HOST:PORT/panel URL in the browser, when logged in as user with administrator rights.

/////localhost:8080/panel screenshot/////

We can see all available actions for administrator.

4.6.1 Add country

Crete a country with typed in name.

4.6.2 Add district

Create a district with typed in name, and select country that owns created district.

4.6.3 Add ward

Create instance of real ward in district - place where people come to vote. Select district that owns created ward.

4.6.4 Grant ward administrator privileges

Choose ward and existing user. Chosen user will be ward administrator of chosen ward. One user could be administrator of many wards.

4.6.5 Add committee

Create a committee with typed in name.

4.6.6 Grant committee administrator privileges

Choose committee and existing user. Chosen user will be committee administrator of chosen committee. One user could be administrator of many wards.

4.6.7 Add poll

/////add poll screenshot/////

Create poll. Type in unique name and short description. Choose date in which poll will be available to vote. Choose poll scope—citizens from which wards can place a vote in this poll. For example, if you choose country scope, citizens from all wards that belong to all districts that belong to country chosen will be able to vote in this poll. Same applies to district scope—citizens from all wards that belong to chosen district will be able to vote in this poll. Next, select voting method. It is a method in which results will be calculated, but also it will determine how voters can cast their vote. There are four available methods, each with constraints:

- Winner takes all—Number of seats to fill must be 1, voter can mark only 1 candidate. Candidate with higher number of votes overall wins.

- Two round—Number of seats to fill must be 1, voter can mark only 1 candidate. Poll is resolved only if one candidate has over half of all votes.

- Instant runoff voting—Number of seats to fill must be less than number of candidates participating (type in desired number), voter can mark maximum of available candidates (type in desired number). Voter will vote on candidates based on his number of preference for each candidate—1 for highest preference, 2 for slightly lower and so on.

- D'Hondt method—Number of seats to fill must be less than number of candidates participating (type in desired number), voter can mark only 1 candidate. Results of poll will be calculated using D'Hondt method, using closed lists from committees, provided by committee administrators.

4.6.8 Register candidate

Select an existing user and register him as a candidate in a poll, as candidate from chosen committee.

4.6.9 Create election

Create an election from existing polls. Voter will have only one chance to vote on polls contained in an election.

4.6.10 Polls

/////polls screenshot/////

Opens available polls menu. For each poll we have two options: Summary and Results.

Summary

Shows if poll is active (can be voted on), and how many votes were casted over the internet.

Results

Calculates and shows results of chosen election. Results will not be calculated if poll is still active, or if any ward did not send the protocol. Once results are calculated they are stored in database and will not be calculated again—there is no need to.

4.7 Security issues

Application is protected against unauthorized access with help of Spring Security. It handles authentication and authorization following only guidelines of the developer. Moreover, I handle business logic security directly in API calls which may affect data written to database. Whenever there is a call that should be protected by some business logic, before it performs anything, it checks whether user that makes that call can perform desired action. If not, it returns with message why action was not performed. For example, see trying to vote in election that you, as a voter, have already voted in.

Chapter 5

Internal specification

5.1 Concept of the system

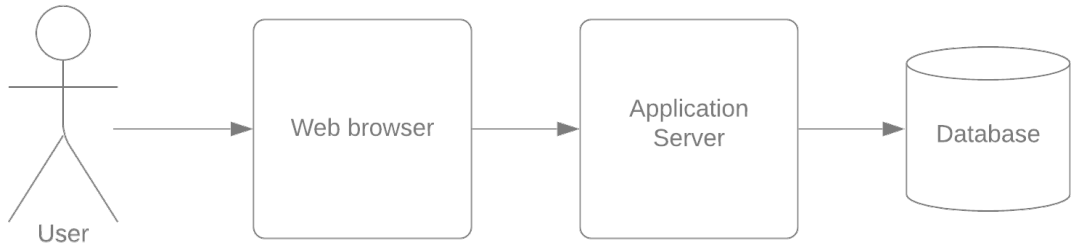

Figure 5.1: Concept of the system

A user communicates with the system via a web browser. A web browser sends HTTP requests to the application server which constructs the response using resources available in a database.

5.2 System architecture

Figure 5.2 shows architecture of the application. A user interacts with a web browser and his actions trigger communication of the browser with the application server via HTTP. Those requests are passed through security filters of Spring Security, which check if user is authenticated and authorized to fetch the resource

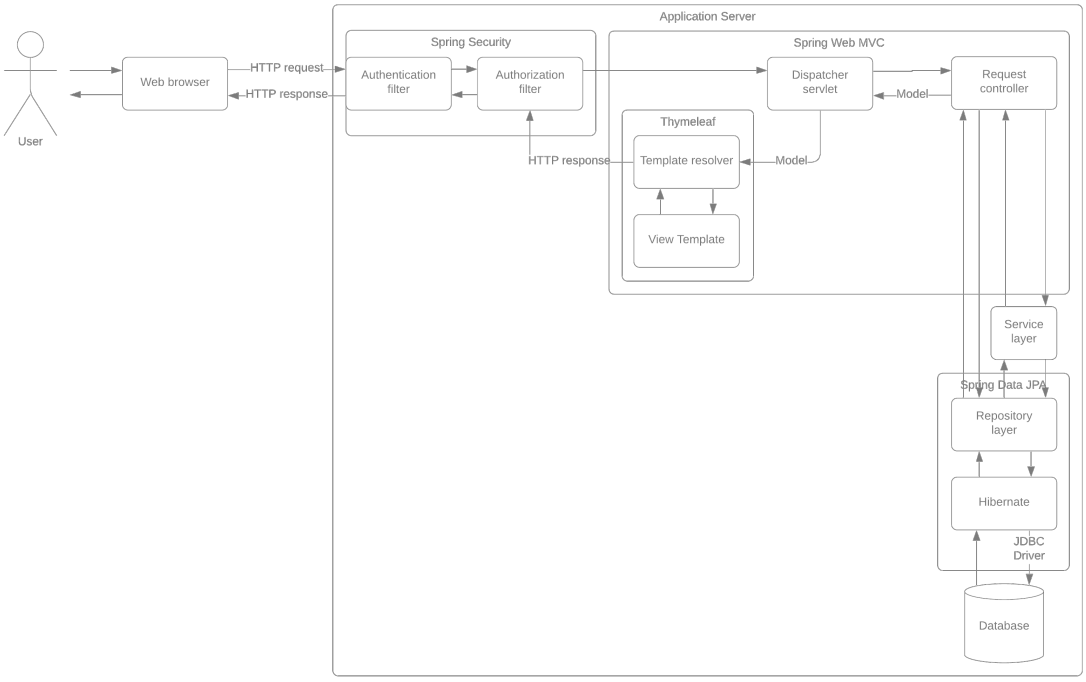

Figure 5.2: Application architecture

he requests. After that, all requests hit dispatcher servlet, which delegates request to it's specific request controller. Repository layer provides data available directly in the database, while service layer performs some additional logic, which is not available in the repository logic (some simple filtering of fetched data). Repository talks to database with help of Hibernate, JPA implementation used by Spring Data JPA. Controller connects all data provided to it by service or repository layer into business logic, and constructs a view model for HTTP request. This view model is passed to template engine, which applies this view model to specific view template (information regarding which template is also mentioned in view model), and constructs HTTP response for this template. This response comes back to security filters, which send response back to web browser.

5.3 Data structure

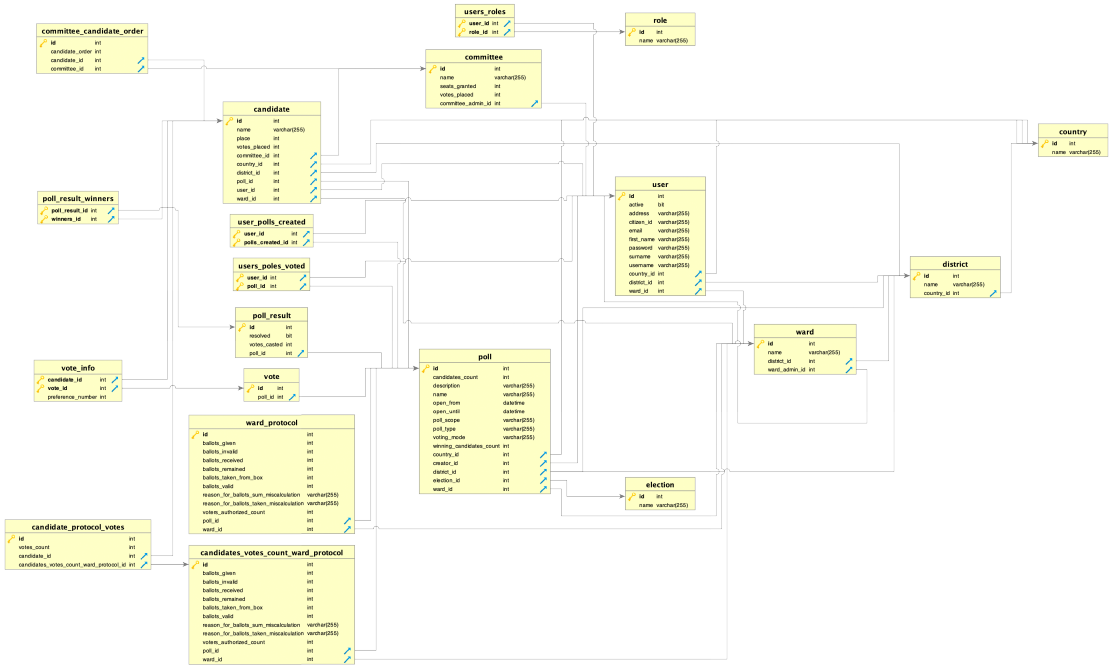

Figure 5.3: Entity-Relationship Diagram

5.4 Implementation of the application

5.4.1 Entity

An entity class is a representation of a table in a database. It can be treated as implementation of business data model, which is used to achieve some business result.

In Figure 1 we can see example code for an entity. Apart from standard Java syntax, there are some annotations related to JPA:

- `@Entity`—marks class as entity. It will be represented in database as a table.
- `@Id`—marks class field as primary key for table.

- @GeneratedValue—determines strategy for generating this value when writing to database in case of null.
- @ManyToOne—marks class field as many to one relationship with another entity. It will store primary key of target entity in table of this entity. There is a twin annotation to this one, @OneToMany for entity on the other side of this relation.

5.4.2 Repository

A repository is an interface that allows a developer for interaction with objects stored in a database. Spring Data JPA provides out–of–the–box implementation of a CRUD repository. A @Repository annotation allows Spring to identify such interface, create implementation of it and hold it in the context. It is easily extensible with help of custom finder methods, for example see figure 2 in the appendices.

5.4.3 Component

In order to help Spring identify dependency classes eligible for DI, we can annotate such classes with any derivative of @Component annotation. When spring is scanning for components, @Component annotated class will be treated as component, thus an object of such class will be created and held in the context. There are several derivatives of @Component annotation, which serve the same purpose, but are named differently for simpler understanding of class role and behavior. Examples of @Component specialized annotations:

- @Service
- @Controller
- @Repository

If needed, those dependencies can be injected using @Autowired annotation. It looks for dependency of matching type as field under annotation, and assigns desired object to class field after owner object construction. Example of @Service annotated class can be seen on figure 3 in the appendices.

5.4.4 Controller

A controller is a class that can process HTTP requests. It is annotated with `@Controller` annotation, along with `@RequestMapping` annotation, which value determines what URL this controller responds to. As HTTP has few methods that determine type of request under certain URL, controller class method annotations like `@GetMapping` and `@PostMapping` map HTTP method to class method, which that request triggers.

There are many arguments that mapping method can take that give developers additional tools for processing HTTP request. This application uses `Model` class as method argument to create a model passed to template. In case of POST HTTP method, we can use `BindingResult` class in method argument to get information about form that was sent with HTTP request. Controller decides which template should be used by returning template location in `String` type. This location is relative to folder `APP_ROOT/src/main/resources/templates` by default. An example of Controller class can be seen on figure 4 in the appendices.

5.4.5 View template

A view template is a HTML file with additional tags and attributes which are parsed by Thymeleaf after populating template with a model. This means that any errors in template file are runtime error, as template is parsed at the end of HTTP response generation flow 5.2.

5.5 Applied design patterns

While many design patterns may be useful in big projects, there was no need to introduce any sophisticated methods of organizing application layers and communication. I kept it simple, and it worked for me. Here are two design patterns visible directly in the application code:

- MVC
- DI

Chapter 6

Verification and validation

6.1 Testing paradigm

We can divide development process into three parts: planning, implementation and testing. Every phase is involved in verification and validation of a solution.

6.1.1 Planning

When planning a new feature, I first thought about it from user point of view—how user could use it, given already implemented features or database schema and relations. For example, when planning ward administrator cockpit, user has to be able to choose ward to manage, because he can be administrator of many wards. That's [That is] why we need to add page in between, so user can choose specific ward. When new feature requires some complicated logic, first I try to break it down on paper, I try to understand it better.

Then, I go into more technical stuff—can I use logic available in repository, or should I create service responsible for logic required? Are there any additional data structures that could be helpful? How should I divide responsibility between classes?

After I have some understanding of the problem, I try to think if there is a better and simpler way of achieving the same result—many times I found myself overcomplicating things.

After that I have solid overview of the problem, and I can start implementing it.

6.1.2 Implementation

It usually involves creating a controller for new functionality, and placing reference to it in some root template, so I can trigger this logic. In preparation phase, I have to load database with exemplary data that may be necessary to debug the feature. After that, I make things I prepared in previous phase come to life—I start coding. Once I have skeleton of a feature, I check if all assumptions I made are correct—if everything works as I expected, although not fully implemented. Step by step I come to solution that works for at least one data set.

6.1.3 Testing

All of test cases were performed manually. For client side part of testing, they were performed during development phase of implementation. It happens naturally that during development of some part, you have to navigate through the application simultaneously testing if everything works as expected.

For business side part of testing, we can distinguish few parts of it:

- Database communication correctness—with every interaction with database, I checked if correct data is written to the database in correct form, and also if I get fetched values as expected.

- Calculation correctness—when implementing methods that calculate results of polls, I tried to think of an example that contains all edge cases of a method. After populating the database with exemplary data, I calculated results of that example by hand, and compared them with application results. If the results were the same, I repeated the scenario with data generated randomly. If the result was still the same as results calculated by hand, I assumed the solution is sufficient.

6.2 Testing scope

Every part of application that is exposed to typical user is tested and error protected. That means that functionalities in voter, ward administrator and committee administrator cockpits are fault tolerant. Because we assume that administrator knows what he is doing, administrator cockpit is error prone. Only poll results calculation is protected, because it involves data entered by third party, and administrator cannot be sure that this data was sent.

6.3 Test cases

6.3.1 Voting test

In voter cockpit user enters data when he casts a vote. Apart from casting a proper vote, he can cast empty vote (no candidates marked) and he can cast too many votes (more candidates than necessary marked). Both of these scenarios invalidate vote.

When it comes to preference voting, he also can cast empty vote and vote with too many candidates marked, but also he can repeat preference numbers and preference number can exceed limit value, which also invalidates a vote.

Those scenarios are tested and application displays appropriate messages to user.

6.3.2 Closed list order test

As a committee administrator we can set order on closed list of a committee. There should be a continuity in order (no numbers missed), and no candidate should be left without a number. When committee administrator tries to save improper closed list order, he should be displayed appropriate message.

This scenario is tested and application displays appropriate messages to user.

6.3.3 Sending ward protocol test

As a ward administrator we can send protocol from that ward. In case of any inconsistency in entered numbers, ward administrator should state reason why there is such inconsistency.

This scenario is tested

6.3.4 Calculating poll results test

As an administrator we can calculate polls results only when poll is finished and all wards have sent their protocols. Otherwise results should not be calculated and administrator should be displayed appropriate message.

This scenario is tested and application displays appropriate messages to user.

Chapter 7

Conclusions

Objective of the thesis was to achieve voting system in accordance with requirements presented in chapter 3. Almost all of the functional and nonfunctional requirements are met using tools described in that chapter.

As to development itself, proposed architecture passed the exam—implementing features was smooth, and it was easy to extend any functionality of the application. However, not everything met my expectations. I found it annoying to work with view templates in Thymeleaf. Presenting data from backend is quite easy, but sending forms filled by user to the controller is counterintuitive and it required more learning than I expected when choosing this technology.

Although everything works as expected, frontend navigation does not work according to UX spirit. It definitely has to be more intuitive for the user. However, that was not in the scope of this project. Another thing that should be considered in terms of further development is more information presented to the user. For example it is possible to present to the administrator which wards have, and which wards have not sent their protocols. That would be useful information when it comes to calculating election results (although it is already impossible to calculate results without every protocol).

Bibliography

[1] The Apache Software Foundation. The apache ant project.

[2] The Apache Software Foundation. Apache maven project.

[3] S. Heiberg and J. Willemson. Verifiable internet voting in Estonia. In *2014 6th International Conference on Electronic Voting: Verifying the Vote (EVOTE)*, pages 1–8, 2014.

[4] C.S. Horstmann and G. Cornell. *Core Java, Volume I–Fundamentals*. Sun Core Series. HELION S.A., 2008.

[5] Gradle Inc. Gradle build tool. [last access: 2020-01-24].

[6] Anna Lührmann, Lisa Gastaldi, Sandra Grahn, Staffan I. Lindberg, Laura Maxwell, Valeriya Mechkova, Richard Morgan, Natalia Stepanova, and Shreeya Pillai. Democracy facing global challenges : V-dem annual democracy report 2019. Report, Department of Political Science, University of Gothenburg, Gothenburg, May 2019.

[7] Inc. Pivotal Software. Spring data JPA.

[8] Inc. Pivotal Software. Spring framework overview.

[9] Inc. Pivotal Software. Spring security.

[10] Inc. Pivotal Software. Web mvc framework.

[11] The Thymeleaf Team. Thymeleaf.

[12] T. Thai and H. Lam. *.NET Framework Essentials.* Essentials Series. O'Reilly Media, 2003.

Appendices

List of abbreviations and symbols

API	Application Programming Interface
APP_ROOT	Root directory of ballotbox project (with pom.xml file)
CRUD	Create, Read, Update, and Delete
DI	Dependency Injection
HOST	IP of host machine of application server
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JPA	Java Persistence API
JSP	JavaServer Pages
MVC	Model–View–Controller
PORT	Port on which application server listens
REST	Representational State Transfer
UML	Unified Modeling Language
URL	Uniform Resource Locator
UX	User experience

Listings

```
1  //TODO: change root package name
2  package com.drzewo97.ballotbox.core.model.ward;
3
4  import com.drzewo97.ballotbox.core.model.candidate.
       Candidate;
5  import com.drzewo97.ballotbox.core.model.district.
       District;
6  import com.drzewo97.ballotbox.core.model.user.User;
7
8  import javax.persistence.*;
9  import java.util.Set;
10
11 @Entity
12 public class Ward {
13
14 @Id
15 @GeneratedValue(strategy = GenerationType.IDENTITY)
16 private Integer id;
17
18 private String name;
19
20 @ManyToOne
21 private District district;
22
23 @OneToMany(mappedBy = "ward")
24 private Set<Candidate> candidates;
25
26 @ManyToOne
27 private User wardAdmin;
28
29 ...
30
31 get methods
32 set methods
33 }
```

Figure 1: The Ward entity.

```
1  package com.drzewo97.ballotbox.core.model.ward;
2
3  import com.drzewo97.ballotbox.core.model.country.Country
       ;
4  import com.drzewo97.ballotbox.core.model.district.
       District;
5  import org.springframework.data.repository.
       CrudRepository;
6
7  import java.util.Set;
8
9  public interface WardRepository extends CrudRepository<
       Ward, Integer> {
10   Boolean existsByNameAndDistrict_Name(String name,
         String districtName);
11   Set<Ward> findAllByWardAdminIsNull();
12   Set<Ward> findAllByWardAdminUsername(String username);
13   Set<Ward> findAllByDistrict(District district);
14   Set<Ward> findAllByDistrictCountry(Country country);
15 }
```

Figure 2: The Ward repository.

```
1  package com.drzewo97.ballotbox.core.service.ward;
2
3  import com.drzewo97.ballotbox.core.model.poll.Poll;
4  import com.drzewo97.ballotbox.core.model.ward.Ward;
5  import com.drzewo97.ballotbox.core.model.ward.
       WardRepository;
6  import org.springframework.beans.factory.annotation.
       Autowired;
7  import org.springframework.stereotype.Service;
8
9  import java.util.HashSet;
10 import java.util.Set;
11
12 @Service
13 public class WardServiceImpl implements WardService {
14
15   @Autowired
16   private WardRepository wardRepository;
17
18   @Override
19   public Boolean isWardAdmin(String username, Integer
         wardId) {
20     Set<Ward> adminsWards = wardRepository.
           findAllByWardAdminUsername(username);
21     return adminsWards.stream().filter(ward -> ward.
           getId().equals(wardId)).count() > 0;
22   }
23
24   /**
25    * Get all wards that could and should provide
           protocols for this poll
26    * @param poll
27    * @return
28    */
29   @Override
30   public Set<Ward> findAllEligibleForPollProtocol(Poll
         poll) {
31     Set<Ward> returner = new HashSet<>();
32
33     switch (poll.getPollScope()){
34       case COUNTRY:
35         returner.addAll(wardRepository.
               findAllByDistrictCountry(poll.getCountry()));
36         break;
37       case DISTRICT:
38         returner.addAll(wardRepository.findAllByDistrict
               (poll.getDistrict()));
39         break;
40       case WARD:
41         returner.add(poll.getWard());
42         break;
```

```
1 import org.springframework.web.bind.annotation.
    ModelAttribute;
2 import org.springframework.web.bind.annotation.
    PostMapping;
3 import org.springframework.web.bind.annotation.
    RequestMapping;
4
5 import java.util.stream.Collectors;
6 import java.util.stream.StreamSupport;
7
8 @Controller
9 @RequestMapping(path = "/panel/ward/create")
10 public class WardCreationController {
11   @Autowired
12   private WardRepository wardRepository;
13
14   @Autowired
15   private DistrictRepository districtRepository;
16
17   @ModelAttribute("ward")
18   private Ward ward() {
19     return new Ward();
20   }
21
22   @GetMapping
23   private String showWardCreate(Model model){
24     model.addAttribute("districts", StreamSupport.stream
         (districtRepository.findAll().spliterator(),
         false).collect(Collectors.toList()));
25     return "panel/ward_create";
26   }
27
28   @PostMapping
29   private String createWard(@ModelAttribute("ward") Ward
       ward, BindingResult result){
30     if(wardRepository.existsByNameAndDistrict_Name(ward.
         getName(), ward.getDistrict().getName())){
31       result.rejectValue("name", "name.exist", "Ward␣
           already␣exists.");
32     }
33
34     if(result.hasErrors()){
35       return "panel/ward_create";
36     }
37
38     wardRepository.save(ward);
39
40     return "redirect:create?success";
41   }
42 }
```

Contents of attached CD

The thesis is accompanied by a CD containing:

- thesis (LaTeX source files and final `pdf` file),
- source code of the application,
- test data.

List of Figures

3.1 Citizen use case UML for Central online voting system 9
3.2 Ward administrator use case UML for Central online voting system 10
3.3 Committee administrator use case UML for Central online voting system 10
3.4 Administrator use case UML for Central online voting system . . . 11

5.1 Concept of the system 25
5.2 Application architecture 26
5.3 Entity-Relationship Diagram 27

1 The Ward entity. XII
2 The Ward repository XIII
3 The Ward service XIV
4 The Ward creation controller XV

List of Tables