IT-UNIVERSITETET I KØBENHAVN

Frontpage for written work/project

ITU student	EBUSS student
☐ Course ☐ Thesis ☑ Project (with project agreement)	☐ Course ☐ Thesis ☐ Project ☐ Summerproject ☐ 4-weeks project ☐ 12-weeks project ☐ 16-weeks project

Title of course, project or thesis:

Analysis, Design and Software Architecture

Course manager/Supervisor(s):

Joseph Roland Kiniry

Name(s):	Birthday and year:	ITU-mail:
1. Jens Dahl Møllerhøj	24/05/1990	jdmo@
2. Michael Valentin Erichsen	20/06-1991	mier@
3. Morten Hyllekilde Andersen	14/07-1988	mhyl@
4.		@
5.		@

Courses with e-portfolio:

Link to e-portfolio:

DIGITAL VOTER LIST

BDSA, E2011, IT-UNIVERSITY OF COPENHAGEN

Authors:

Jens Dahl Møllerhøj (jdmo@itu.dk),
Morten Hyllekilde Andersen (mhyl@itu.dk)
&
Michael Valentin (mier@itu.dk)

Abstract

Voter lists are used to write voter cards and determine eligible voters at the voting venues. Traditionally voter lists has been centrally printed and manually checked at the voting venues. The efficiency of this procedure can be dramatically increased with the use of digital technology. This however, introduces a number of questions regarding security and data consistency. This project focuses on creating a simple but highly usable digital voter list system that could be used at an election. This project is not about making a system for digital voting (also known as e-voting), but only for maintaining the voter lists digitally, for greater consistency, overview and efficiency.

Table of contents:

Mandatory Requirements

The finished system MUST do the following:

1. Generate voter cards from a given set of eligible voters
2. Authenticate a voter based on his/her voter card and/or personal information
3. Register when a voter has been handed a ballot and store this information
4. Prevent that a voter can be handed more than one ballot
5. Be able to authenticate and register voters at multiple machines in various venues.
6. Have an interactive user interface, for the authentication and registration of voters

Secondary Requirements

The system MAY do the following:

1. **Usability:**
    a. Make the user interface easy to use for non-technical users (election representatives)
    b. Voters can retrieve their ballot at any voting place within their voting district
2. **Persistence:**
    a. Not loose data in the event of a typical system failure (eg. a computer crashes)
    b. Not loose data in the event of a network failure
3. **Scalability:**
    a. Make sure the system can handle a large number of voters. At least 5 million, but preferably more
    b. Make sure the system can handle a large number of simultaneous users (election representatives). At least 7.000 but preferably more
4. **Security:**
    a. Make sure the system uses proper security measures and crypto-technology to establish confidence that the system is secure
    b. Filter the citizens based on multiple lists and criterion to determine eligible voters
    c. Analyze the resulting data, to detect suspicious voters and fraud
    d. Make better authentication questions for voters (eg. generate a secure question or set of questions based on the voters personal details)
    e. Status on the digital voter list prior to election and after
5. **Analysis:**
    a. Analysis of the turnout, both nationally and for specific turnout results
    b. A public API for the media to access
    c. Visualize the turnout results

Overview

This project is a part of the course *Analysis, Design and Software Architecture* fall 2011 at the IT University of Copenhagen. The aim of the project is to design and implement a digital voter list, for use at an election, while making use of the methods and techniques taught in the related course. The project was developed in the period 21/11-14/12, and is programmed in C# using Visual Studio 2010.

Team members

All team members are part of the bachelor in software development program at the IT university and enrolled in the course *Analysis, Design and Software Architecture*. The development team has been working closely together through the whole process, while some team members have focused more intensely on specific parts of the program. Below is a description of each team-member and their specific focus in the project.[1]

Jens Dahl Møllerhøj

Jens' primary focus has been on testing and making sure that the program delivers expected results. He has also been in charge of developing the separate data-generation-project[2] and the data transformation process of processing raw data into data that is ready for the rest of the system to use. Jens has also made a lot of work on the MySql-implementation of the Data Access Object, including transactions and contracts.

Morten Hyllekilde

Morten has been in charge of most of the graphical user interface, including the layout and setup of most windows and working on the related controllers. Morten has also been in charge of handling the printing of voter cards, including the generation of machine readable barcodes.

Michael Valentin

Michael has been in charge of the overall architecture, and has outlined most of the core classes. He has worked a lot on implementing the MySql Data Access Object with transactions and contracts. He has also helped a lot with the testing-process and worked on the GUI and especially the controllers.

[1] Every file in the source-code is commented with authorship statements, as to indicate who have worked on what.

[2] A simulation of data, that could be collected from other sources, that we unfortunately don't have access to.

Project details

In the following we will point out specific parts of the project and our solution, that we think are interesting and discuss them briefly.

The KMD digital voter list

The Danish government hired the firm KMD to develop a similar digital voter list system. There have been made some suggestions from outsiders (e.g. Joe Roland Kiniry from the IT University of Copenhagen) to how the system could be improved, and this was our inspiration for choosing this particular project. We had hoped that KMD would have provided us with a reasonable amount of details regarding their solution, but unfortunately they decided not to make any material available to us; why we have been doing our system without any insight on the system that KMD did. The goal of our project was to demonstrate how we would handle the task of developing a system to accomplish the same tasks, but with a potentially different approach similar system.

Security concerns

It is of great importance that a system to be used at an election is believed to be secure. We chose to look at the security from two aspects. One is when citizens or election representatives would try and commit election fraud, by voting more than one time or similar. The other is digital security, where somebody would try to compromise the system by feeding it malicious data, manipulating requests, responses or similar.
For digital security, we assume that our development environment and the connection to the database can be made secure by applying modern cryptographic technology and other security measures. This however, is not a part of our project.
For the manual security, we assume that the election representatives can be trusted to some reasonable extend, however we also assume that people turning up the voting venues can't be trusted. That is why our system implements a number of security measures, which are described below.

Layers of authentication

When a voter turns up at the voting venue it is important to determine their identity. We do this with a simple three step process, with an option for manual validation if the automated process fails.

1. The voter must present his/her valid voter-card, which the election representative uses to look up the voter.

2. The voter must enter the last four digits of his/her CPR-number, to verify his/her identity.

3. The voter must answer to one personal question.

The first step of the process has great deal of security, as the voter card with its unique key is sent to the voters address by the postal system. This makes fraud more difficult. The second step is important to verify the voter's identity. Should the voter have picked up a bunch of voter cards from a (irresponsible) printing store, he wouldn't know the correct CPR-number. However, these days CPR-number

information can be obtained through various sources, and that is why we have our third step, which includes a random, personal question, that is simple for an authentic voter to answer, while the large set of possible questions makes fraud even harder.

Should this validation process fail, there should always be at least one privileged election representative at every voting venue, who can perform a manual validation and registration of the voter.

Contracted transactions

To ensure that no users can ever be accepted to vote twice (for example), we make all of our queries to the database happen in transactions, so that nothing is returned before we are absolutely sure that the requested operation can happen without conflicts. This also allows for code contracts, for better transparency and stability in the system.

The database part of the system gave us a lot of unexpected troubles. We were not able to run PEX on the data access object, and the transactions were harder to implement than we had predicted. In retrospect, we found that our limited development experience; especially using design by contract, combined with complex issues in making a database critical application, have cost us a lot of time, resulting in a less polished product. However we feel like we've learned a lot from this process and gained a lot of valuable experience.

Efficiency

Our system is very efficient when using a local database and fairly efficient for typical operations when using a remote database. We're pretty confident that efficiency could be greatly improved from the code, by making it possible to instantiate multiple objects with rows from a single query to the database. However, we didn't have the time to implement this feature. Moreover, there should be a great room for improvement of efficiency by having a better and more stable connection to the database.

Election administration

The administration part of the system is for the administrators of the election to handle the most important tasks related to the database. This allows them to import data of all the citizen, update the voter card and print them. The administration system should have had many more functions and possibilities as outlined in our secondary requirements, however we didn't have the time to implement these features. Moreover the administration part of the system shouldn't be shipped with the software for the voting venues.

Design and specification

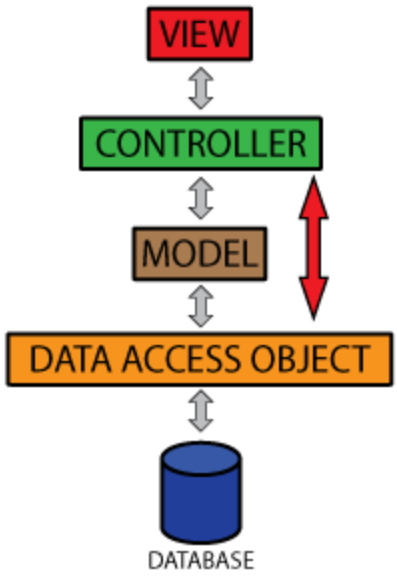
We sketched up our model[3] using informal bon. The classes and clusters that we made in the design phase have not changed much during the development. In the design phrase, we hit a point where our

[3] Meant as the Model in a Model-View-Controller architecture

limited knowledge of making large software systems made it impossible for us to predict what kind of problems we would run into. Furthermore we found it hard to make proper contracts and specification for database related methods, as we had not looked much into this topic yet.

Layered architecture

Our architecture is supposed to be layered such that we have a controller listening for user input from a view and asking our model if any additional information is necessary. The model then makes a database call through a data access object if necessary. The illustration explains what we wanted to do.



However we didn't follow this pattern at all time, in particular we made a lot of calls directly from our controllers to our Data Access Object, which is not according to the supposed layered architecture.

Restricting method access

To make the system secure at code level, we wanted to implement restrictions on which methods a user could call. To accomplish this without cluttering the Data Access Object implementation with permission checking, we introduced a permission proxy, which all DAO calls must be routed through. This proxy utilizes a set enums that resembles permissions for system actions, which are assigned to users. The permission proxy detects if the user has the necessary permissions and denies the call if necessary.

Iteration over database data

Since we were not able to return iterate over data from the database outside of the DAO, all methods that required this functionality had to be placed in the data access object itself. This is a design problem because the data access object contains critical business logic that should have been placed in the model. As this is a very universal problem, we would have hoped to explore it in greater detail, but we didn't have the time.

Security without contracts

To make sure that the system isn't completely insecure (though hardly reliable) we break the *non-redundancy principle*[4]. In order to assure security, even when the project is compiled without contracts, we insert redundant checks for permissions and similar conditions.

[4]Bertrand Meyer : Object-Oriented Software Contruction (p. 343) : *Under no circumstances shall the body of a routine ever test for the routine's precondition*

Things we wanted to do…

There were a lot of things that we wanted to do, while we didn't have the time to do them. A lot of these are described in the secondary requirements as stated earlier, but some new once have appeared during the design process. Some of the most important of these are:

A logging system

A logging system for logging all events, including who did it, when and where would impose a great measure of security and allow for later statistics and/or investigations to detect fraud. This would also be a great way log possible security issues, when errors occur in the data access object.[5]

A cache layer

Our system could have been greatly improved, both in terms of performance and reliability, by introducing a cache layer with a local database at each voting venue. This would allow the election process to continue even though the connection to the global database couldn't be established. However this would require some careful thought on synchronizing and an implementation of a new layer that we did not have the time to do.

Improved Data Access Object and overall architecture

From the experience we have gained through the project there is some important changes that we would have done to our overall architecture and especially the data access object. This includes a better separation of concerns and a greater consistency in the way we do things.

Dictionary

The dictionary is a mapping of all domain-specific words to an explanation, enabling the reader to better understand the contents of this document.

Classifier	Explanation
Person	A human being
Citizen	A human being relevant to the election; eg. a Citizen of Denmark.
Voter	A citizen that is eligible to vote
Eligibility to vote	A citizens eligibility to vote depends on certain criterion determined by law[6]
Election	A person working for the election host at the election, helping with registering

[5] We actually have these security logs scheduled with ToDo's in our source-code, but we didn't have the time to implement them.

[6] See: http://da.wikipedia.org/wiki/Valgret

Representative	people and handing out ballots.
Election Official	A highly trusted election representative, that has permission to perform manual validation of voters.
Election administrator	An employee of the election host, with responsibility for updating data, printing voter cards and similar.
DAO	Data Acces Object. All communication with the database goes through this object.
Quiz	Quiz elements are attached to voters and contains generated answers and questions for the user to ask.
Normal registration	The process of voting with a voting card, entering a correct CPR-pin and answering a security question
Manual registration	The process of voting with a manual validation of documentation, possibly combined with related security questions.
Election Event	The event representing a specific election.
Eligible voter	Voters who have permission to vote in order of the rules decided by the Danish Government.

Example

The section below contains examples on how to use the system, including installation instructions.

Login: jdmo **Password:** 12345 (Use this to enter the program as administrator)
Login: elec **Password:** 12345 (Use this to enter the program as election representative)

Installation

The system can be executed from the .exe files. However the host unit must have the 4.0 .net-framework and have the MySql .net/connector installed. This could be included in an installer to make the process easier for the user.

Using the local database

To experience the program without the connectivity issues and latency of a remote database, the local version of the program (DigitalVoterList_local.exe) queries a local database, which must be installed on the local computer and accessible on *localhost* with user *root* and password *abcd1234*. In the database the schema defined in *db/mysql_dump.sql* must be available as under the name px3.

Domain examples

Below are examples of the use of our solution, inspired by our BON scenario charts.

Example 1

Action	**Descriptions**
[Voter] Receives a voter card	Jens Jensen receives a voter card by mail with information about which voting venue he should use when voting and the date.
[Voter] Attempts to vote	He turns up at the voting venue and asks a user to register him, so he can vote.
[User] Retrieve data	The user scans Jens Jensen's voter card and fetches his information about the database.
[Voter] Types password	Jens Jensen types his password which is the last four digits of his cpr-number.
[User] Approving cpr-number and answer question	The user sees that the cpr-number is approved by the system. He reads up the question "When is your birthday?" which is fetched randomly from the database.
[Voter] Answers question	Jens Jensen answers the question:"21st of March 1976".
[User] Approving question and register	The user matches the given answer with the one in the database and approves it. He then registers Jens Jensen in the database without any errors and hands him a ballot.

Example 2

Action	**Description**
[Voter] Receives a voter card	Lars Larsen receives her voter card by mail with information about the election.
[Voter] Loses voter card	Lars Larsen misplaces her voter card, so she asks the administration to send a new one.
[Election Administration] Prints voter card	The election administration marks the old voter card as invalid, prints a new one and sends it to Lars Larsen
[Voter] Receives voter card	Lars Larsen receives the new voter card with information about the election.
[Voter] Attempts to vote	Lars Larsen turns up at the voting venue specified on his voter card and asks the a user to register him.

[User] Retrieve data	The user can't scan the barcode, so he is forced to use to enter the id-key on the voter card manually. The data is then fetched from the database.
[Voter] Attempts to type password	Lars Larsen attempts to type his password three times, but fails to remember the last four digits of his cpr-number.
[User] Sends to manual registering	The user is obliged to send Lars Larsen to the election representative to register manually.
[Election Representative] Retrieve data	The election representative retrieve the data about Lars Larsen by entering the voter card id-key and asks him a question about his personal data.
[Voter] Answer question	Lars Larsen answers the question
[Election Representative] Approve answer and register	The election representative approves the answer, registers Lars Larsen in the database and hands him a ballot.

System demo

To give you an idea about the various windows and functions in our graphical user interface, we've included a video-screen-cast (program_demo.mp4) with our project. This substitutes a written walk trough of the program.

Bibliography

Most of our project is done with course material and various documentation, however the following has been a great inspiration:

- Betrand Meyer, Object-Oriented Software Construction, 2. edt, Prentice Hall 1997
- http://dev.mysql.com/doc/refman/5.6/en/connector-net.html
- http://dev.mysql.com/doc/refman/5.6/en/index.html
- http://da.wikipedia.org/wiki/Valgret
- Walden & Nerson, Seamless Object-Oriented Software Architecture, Prentice Hall 1995

Revision History

- 21/11 Approval of the Project Overview
- 30/11 System design produced
- 8/12 MySQL DAO done without transactions, we decide to trash it, and work on a new version which includes transactions in order to be able to write proper contracts.
- 12/12 Working system produced and demonstrated
- 13/12 Submission of Project Overview and project

Version control repository:
https://github.com/signifly/PX3-BDSA-E2011/

Related Documents

- BON Specification (`Specification/DigitalVoterList.bon`)
- BON Spec HTML rendering (`Specification/DigitalVoterList.bon.html`)
- Screencast of program demo (`DigitalVoterListWalkthrough.avi`)
- ER-diagram of the database (`Specification/DigitalVoterList_ER-Diagram.png`)
- Program source code (`Source/*`)
- Example voter cards (`ExampleVoterCards.pdf`)