Database Management Systems Project - EVoting

E-Voting

COLLABORATORS			
	TITLE : Database Management Systems Project - EVoting		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Group 3, Bahtiyar Samet Çoban, 040090553 (Şule Gündüz Öğüdücü), Coşkun Deniz, 040090578 (H. Turgut Uyar), Fikret Aktaş, 040080122 (H. Turgut Uyar), and Ozan Arkan Can, 040090573 (H. Turgut Uyar)	08.01.2013	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

List of Figures

List of Tables

General Information

Chapter 1

Introduction

1.1 Project Description

In countries which are governed with democratic regimes, voting is one of the fundamental civic duties.However, there are some difficulties such that some people gerrymander, influential people in rural areas force other people to vote for person they support, people live in areas that have the problem of terrorism are afraid of voting, traveling necessity for some people who live away from their voting districts, having a physical disability etc.

Considering all of these, there is a need for an online voting system which will be able to reduce or remove completely these difficulties. Our project aims this. With this system, governments can organise elections in a faster, more reliable, and more economical way. Results can also be determined faster and accurately. This is an easy and effective way in terms of people who will vote too.

E-Voting system is used by four types of users. These are *Admin*, *Party Administrator*, *Elector* and *Candidate*.

- *Admin:* Admin can conduct add, delete and update operations on admin, party, party administrator, elector, province, election, and independent candidate using admin page interface.
- *Party Administrator:* Party Administrator is added by admin and can conduct add, delete and update operations on candidate using party administrator page.
- *Elector:* Electors are added by admin. When they login to system as elector, they are redirected to theirs personal pages. On this page, they can update their information, or if there is an active election they can vote using vote link which will be appear during election. After election they can also see the election results using results page.
- *Candidate:* Candidates are added by party administrator. When user login as candidate, he/she is redirected to candidate page. On party page, user can see the candidates of a party by clicking on party name.

1.2 Development and Running Environment

Project has been developed with Java language in PostgreSql database. In the development phase, project has been runned on Apache Tomcat 7.0.23 server, tests have been done on this server. To maintain the connection between Java and Html, Wicket 1.4.10 interface has been used.

Project works on every operating system. It has been develeoped in Netbeans 7.2.0 IDE.

In addition to Wicket framework, CSS , JavaScript and Ajax has been used to maintain user friendlyness and good design. But this additions make the project browser-dependent. On Mozilla Firefox and Google Chrome project works properly but some CSS problems occured in Internet Explorer.

1.3 Work Share

Bahtiyar Samet Çoban

I was responsible for ELECTOR and PROVINCE tables, their database operations such as delete - add - update and their page designs. I also implemented delete operation and page for candidate. Files under my responsibility:

- Elector.java
- ElectorDM.java
- ElectorPage.java, ElectorPage.html
- ElectorEditPage.java, ElectorEditPage.html
- ElectorUpdatePage.java, ElectorUpdatePage.html
- ElectorDeletePage.java, ElectorDeletePage.html
- ElectorDisplayPageLink.java
- ElectorCollection.java
- Province.java
- ProvinceDM.java
- ProvincePage.java, ProvincePage.html
- ProvinceDisplayPageLink.java
- ProvinceCollection.java
- ProvinceEditPage.java, ProvinceEditPage.html
- ProvinceUpdatePage.java, ProvinceUpdatePage.html
- ProvinceDeletePage.java, ProvinceDeletePage.html
- CandidatePage.java, CandidatePage.html
- CandidateDeletePage.java, CandidateDeletePage.html
- style.css (partly)

Coşkun Deniz

I was responsible for USERS table, login operations, and page designs. I also implemented add operation for independent and normal candidate. Files under my responsibility:

- User.java
- UserDM.java
- PasswordGenerator.java
- MySession.java
- LoginPage.java, LoginPage.html
- BasePage.java, BasePage.html
- HomePage.java, HomePage.html
- AdminPanel.java, AdminPanel.html
- AdminBasePage.java, AdminBasePage.html
- AdminPage.java, AdminPage.html
- AdminAddPage.java, AdminAddPage.html
- AdminDeletePage.java, AdminDeletePage.html
- AdminUpdatePage.java, AdminUpdatePage.html
- CandidateAddPage.java, CandidateAddPage.html
- IndependentAddPage.java, IndependentAddPage.html

- style.css (partly)

Fikret Aktaş

I was responsible for PARTY, PARTYADMINISTRATOR and CANDIDATE table and pages of party and party administrator. Besides, I implemented all operations of party and party administrator and also candidate update operation and candidate display page. Files under my responsibility:

- Candidate.java
- CandidateDM.java
- CandidateUpdatePage.java, CandidateUpdatePage.html
- CandidateDisplayPage.java, CandidateDisplayPage.html
- CandidateDisplayPageLink.java
- Party.java
- PartyDM.java
- PartyCollection.java
- PartyDeleteForm.java
- PartyDeletePage.java, PartyDeletePage.html
- PartyDisplayPage.java, PartyDisplayPage.html
- PartyDisplayPageLink.java
- PartyEditForm.java
- PartyEditPage.java, PartyEditPage.html
- PartyPage.java, PartyPage.html
- PartyUpdatePage.java, PartyUpdatePage.html
- Party.java
- PartyAdministratorDM.java
- PartyAdministratorCollection.java
- PartyAdministratorDeleteForm.java
- PartyAdministratorDeletePage.java, PartyAdministratorDeletePage.html
- PartyAdministratorDisplayPage.java, PartyAdministratorDisplayPage.html
- PartyAdministratorDisplayPageLink.java
- PartyAdministratorEditForm.java
- PartyAdministratorEditPage.java, PartyAdministratorEditPage.html
- PartyAdministratorPage.java, PartyAdministratorPage.html
- PartyAdministratorUpdatePage.java, PartyAdministratorUpdatePage.html

Ozan Arkan Can

I was responsible for ELECTION and VOTE tables, voting operations, showing results, developing operations that are related organizing election, basic database connection and page designs. Files under my responsibility:

- Election.java
- ElectionDM.java
- AddElectionForm.java
- DeleteAndUpdateElectionForm.java
- ElectionEditorPage.java, ElectionEditorPage.html, ElectionEditorPage$ElectionAdd.html, ElectionEditorPage$ElectionEdit.html
- ResultsPage.java, ResultPage.html

- ResultForAProvince.java, ResultForAProvince.html
- Vote.java
- VoteDM.java
- VotePage.java, VotePage.html
- PostgreSqlManager.java
- DatabaseDemoCreator.java
- datetimepicker_css.js
- style.css (partly)

1.4 Setup

The EVoting Project is a database project and PostgreSql is used in project. To initialize database:

- Install postgresql 9.2 (Download from website or install from application manager(linux))
- Install PgAdmin when installing the postgresql (There will be a choice about it).
- Add a new server and settings must be like following. Password must be 123456.
- Create database that is named EVoting.

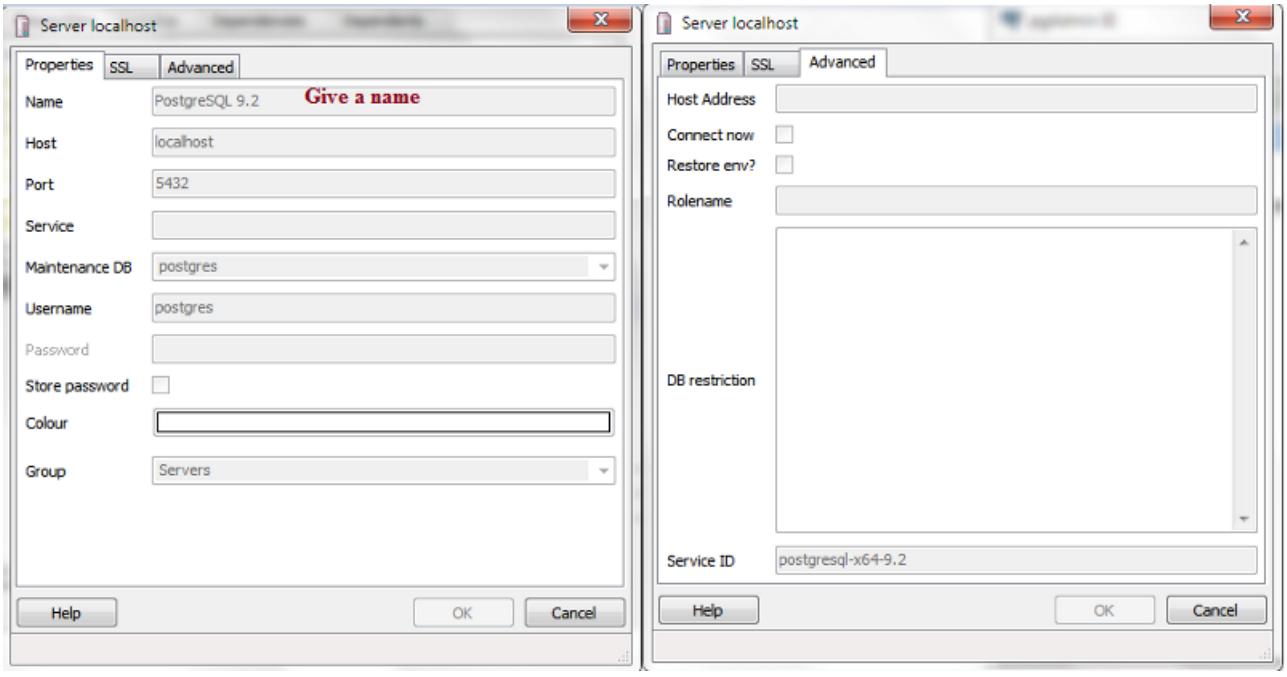


Figure 1.1: Settings for server

After the postgresql is initialized the project can be run on apache server using war file. Another way to run the project is using Netbeans. Netbeans has apache server plugin. After installing it, project can be opened from source codes and can be run from Netbeans.

Chapter 2

User Manual

2.1 Elector

2.1.1 Database Operations and Web Pages

2.1.1.1 Login

Electors can login to system with their TC number and password which is given from "Yüksek Seçim Kurulu".

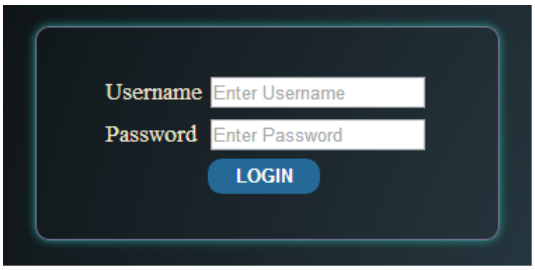


Figure 2.1: Login Page

If elector login to system successfully his/her current information will be printed to the screen. Therefore,they can look and change if they want by clicking update your information button which is on the left side of the page.

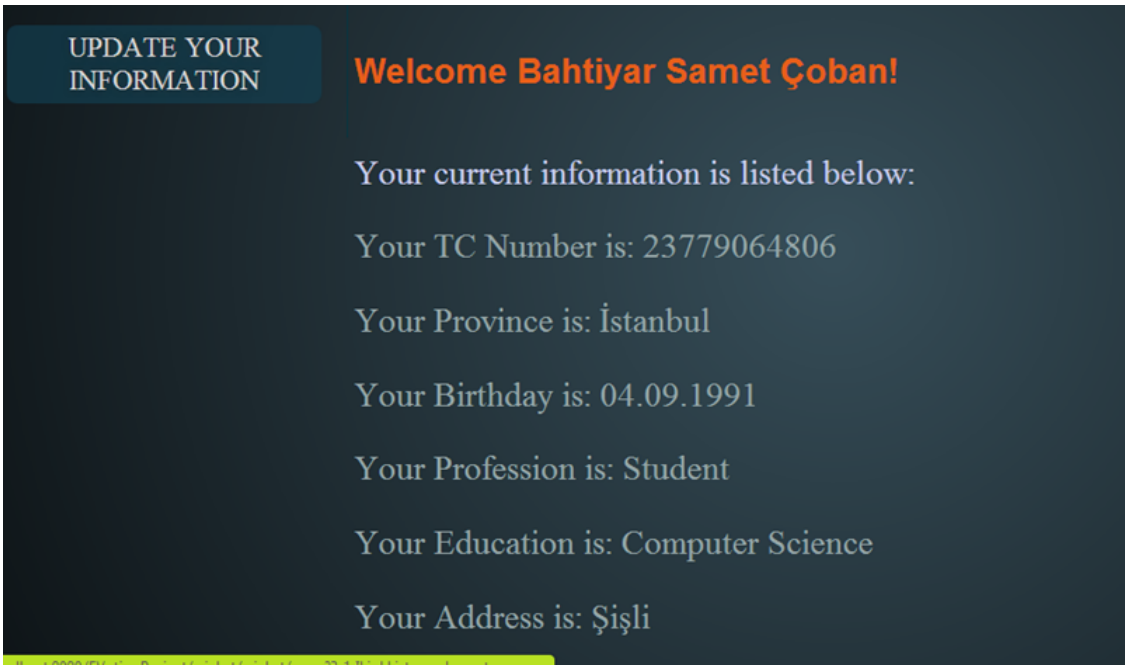


Figure 2.2: Elector Page

2.1.1.2 Elector Update

When elector clicks the "UPDATE YOUR INFORMATION" button, Elector Update Page will be come to the screen. Electors can update their informations if election is added to system but not started. Elector's TC number is added automatically. TCnumber field is designed as read only so elector can not change it.

Figure 2.3: Elector Update Page 1

After clicking GET button, elector information will be pulled from database and filled automatically into field boxes and elector

can change and update it easily.

All fields are designed as required, this means elector can not leave this fields empty. Also type checkings, max-size checkings are added. For example, elector can not give a string to Province ID. All these checkings which applied to all add- delete- update form pages are handled by a Feedback Panel which will be explained later on.

Figure 2.4: Elector Update Page 2

2.1.1.3 Elector Add

Since this is a National E-Voting project, electors can not add themselves to the system, only admin can add them. As a result of that fact this page only accessible from admin panel. Admin can add elector when election is added to system but not started.

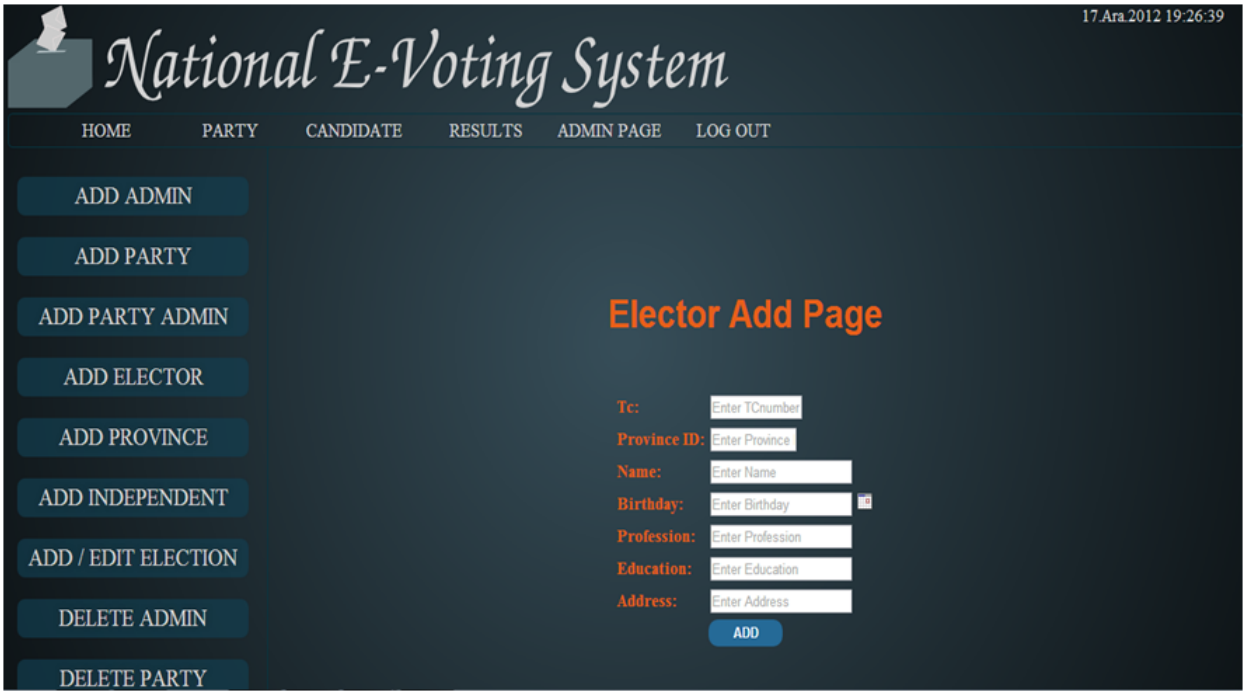


Figure 2.5: Elector Add Page

2.1.1.4 Elector Delete

Again with the same reason, electors can not delete themselves to the system, only admin can delete them. As a result of that fact this page only accessible from admin panel. Admin can delete electors with their TC number when election is added to system but not started.

Figure 2.6: Elector Delete Page

2.2 Province

2.2.1 Database Operations and Web Pages

Since Province is not a actor of this project there is no login or Province page in the system.

Province is only controllable from admin panel. That's why Add- Delete- Update pages of Province is created from Admin Base Page.

2.2.1.1 Province Add

Admin can add provinces to system if there is no time constraint such as election is started or there is less time than 2 weeks to election start. In this cases system will produce a warning. This is also same for delete and update operations.

Figure 2.7: Province Add Page

2.2.1.2 Province Update

When admin clicks the "UPDATE PROVINCE" button on the admin panel, Province Update Page will be come to the screen as showed in the figure.

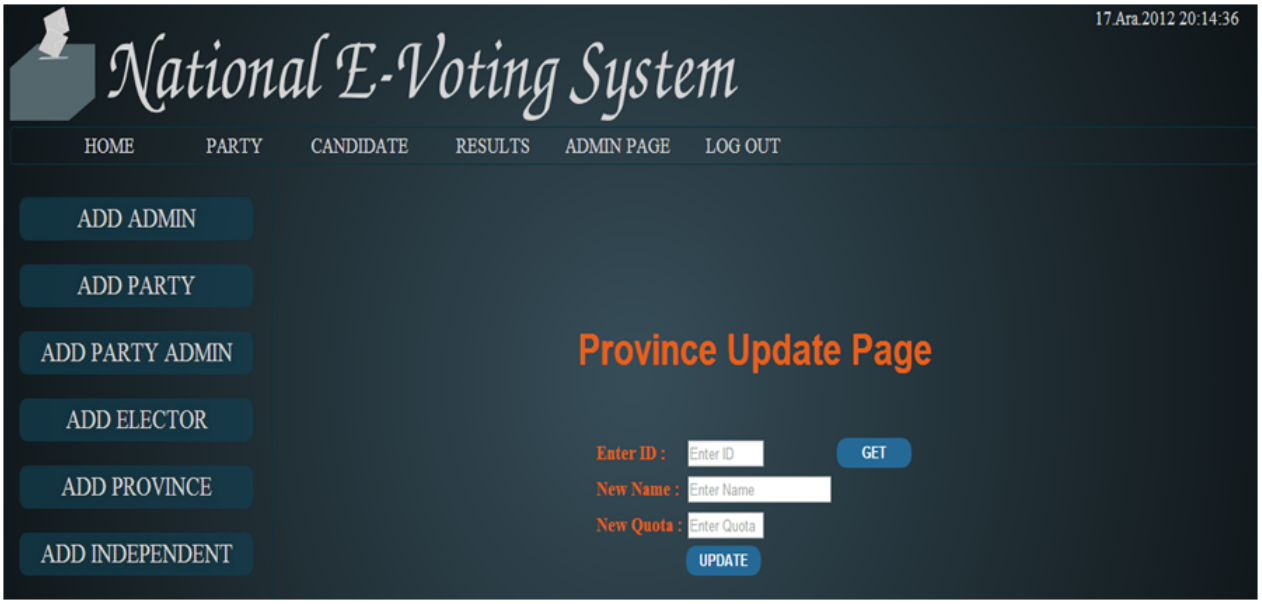


Figure 2.8: Province Update Page 1

Admin can update province information with entering Province id. After clicking GET button, province information will be pulled from database and filled automatically into field boxes and admin can change and update it easily.

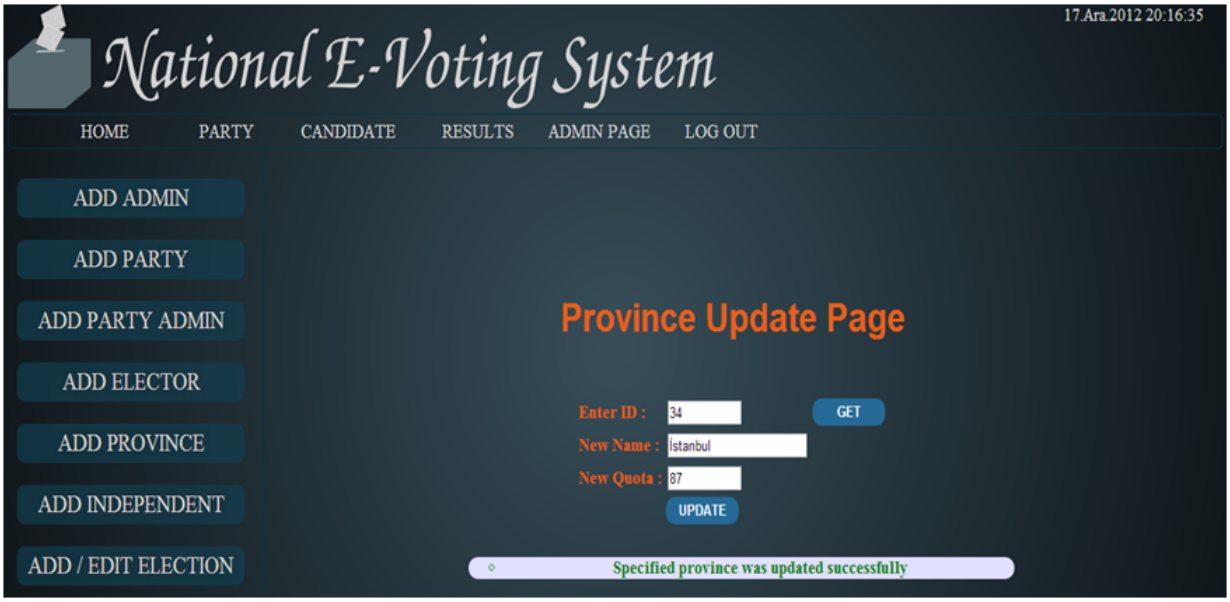


Figure 2.9: Province Update Page 2

2.2.1.3 Province Delete

Admin can delete provinces from the system with entering Province id.

Figure 2.10: Province Delete Page

2.3 Login Operations and User

2.3.1 User Login

There are four types of users in our system. These are:

- Admin
- Party Administrator
- Elector
- Candidate

If user did not log in, **LOGIN** link appears in navigation bar.

Figure 2.11: Login link in navigation bar

User can reach the login screen by clicking the login link in navigation bar. After that, user is redirected to login page which is the following login form appears.

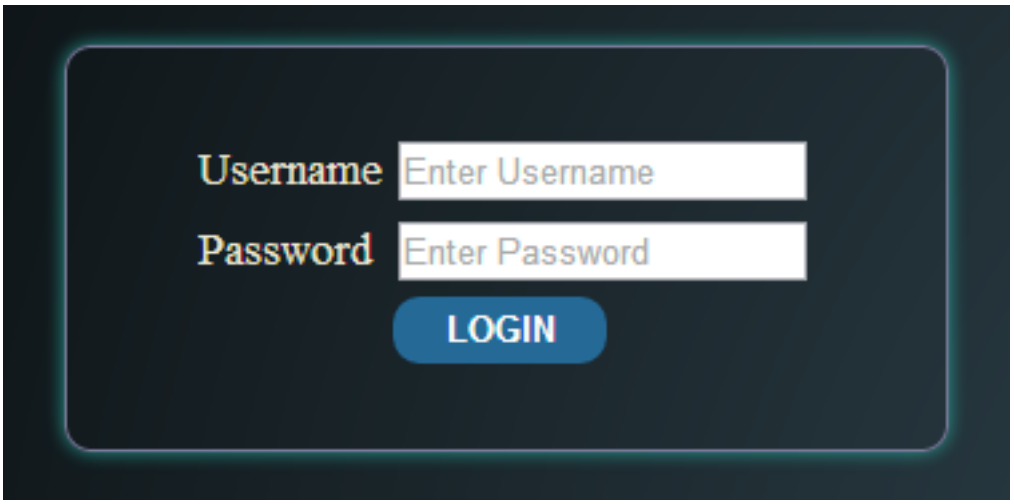


Figure 2.12: Login Form

If login is successful, user is redirected to related page according to user type.

Admin Login

If logged in user is admin, he/she is redirected to admin page. In navigation bar admin page and log out links appear.

Figure 2.13: Navigation bar with admin login

In admin page, there is an admin panel on the left which is shown below. Using this panel, admin can carry out add, delete and update operations.

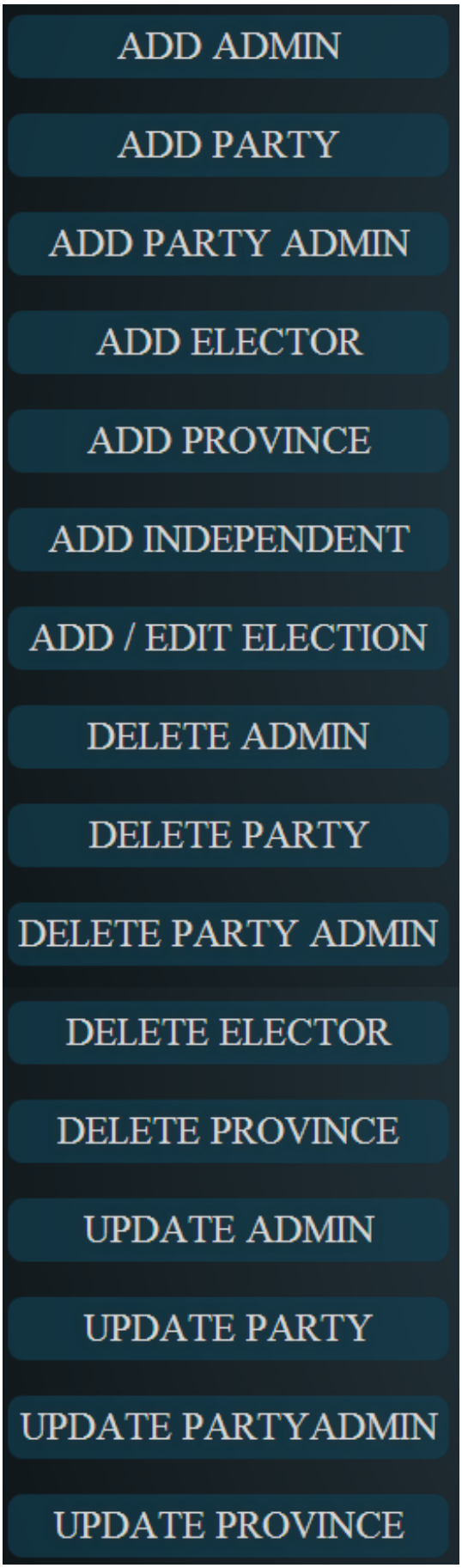


Figure 2.14: Admin Panel

Party Administrator Login

If logged in user is party administrator, he/she is redirected to party administrator page. In navigation bar party admin and log out links appear.

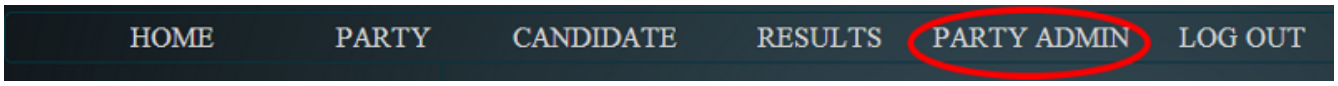


Figure 2.15: Navigation bar with party admin login

Elector Login

If logged in user is an elector, he/she is redirected to elector page. In navigation bar elector and log out links appear.

Figure 2.16: Navigation bar with elector login

If there is an active election shown in announcements field on the left,

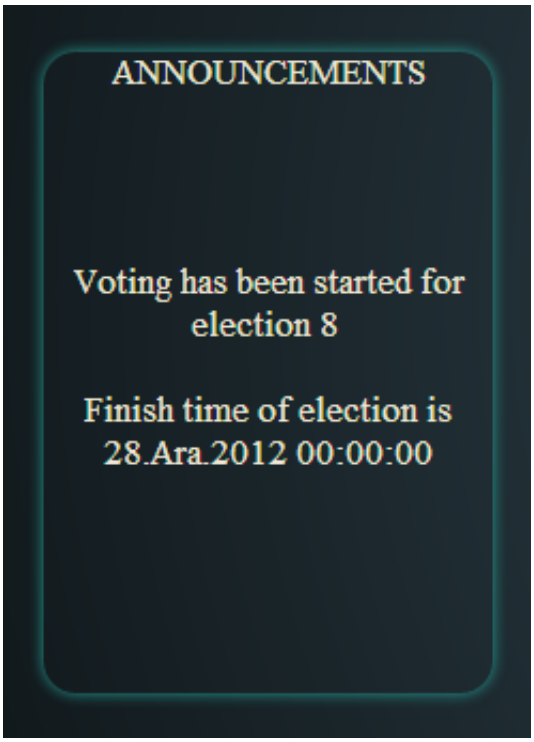


Figure 2.17: Announcements field with active election

VOTE link also appears in navigation bar.

Figure 2.18: Navigation bar with active election

Using this link, elector can reach voting page.

If user tries to log in with invalid credentials or leave fields blank, error messages are shown like below.

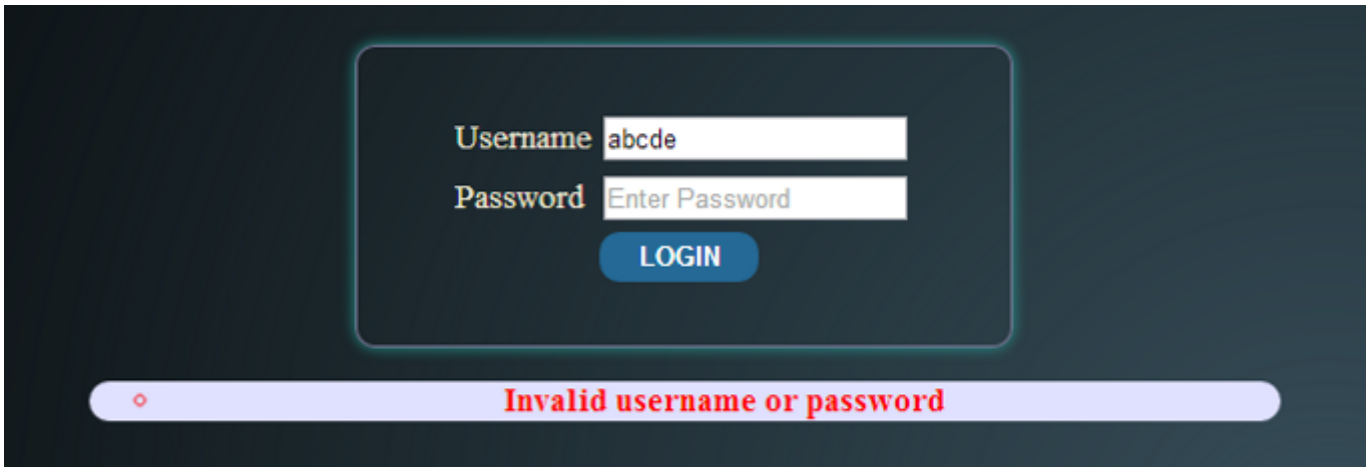


Figure 2.19: Error message for invalid username or password

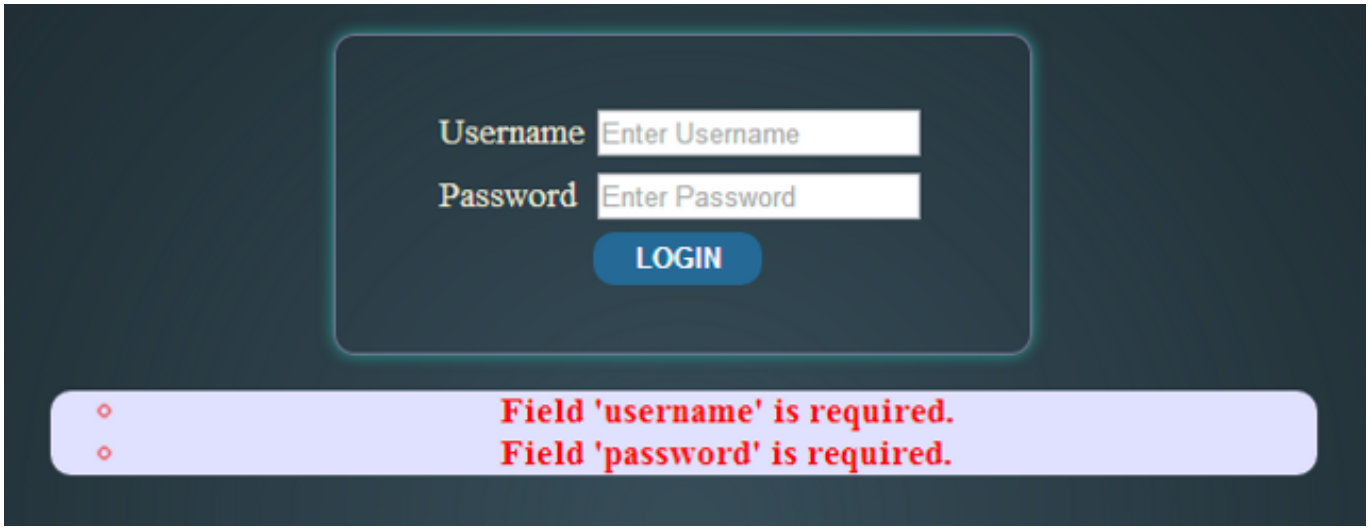


Figure 2.20: Error message for blank fields

2.3.2 Responsibilities of Admin

Admin generally manages the user database. Admin

can add

- Admin
- Party
- Party Administrator
- Independent Candidate
- Elector

- Province
- Election

Admin can add another admin using ‘ADD ADMIN’ link in admin panel.

Figure 2.21: Admin adding interface

After adding admin successfully, an information message is shown.

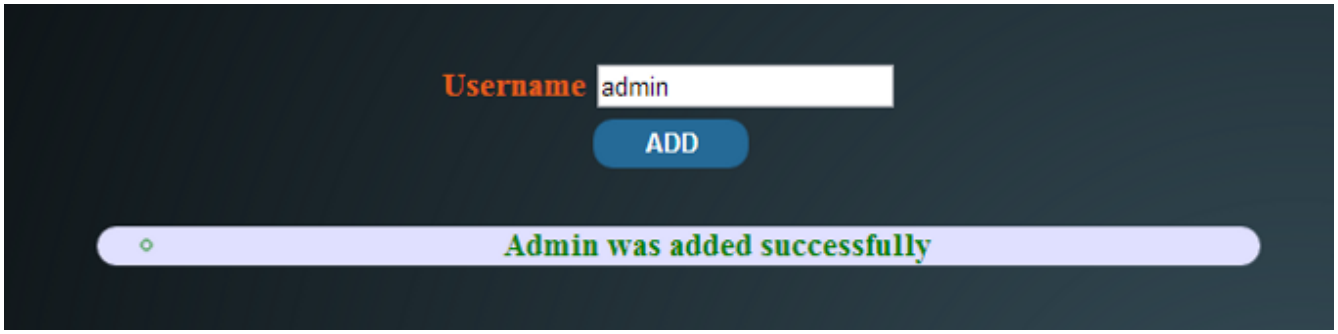


Figure 2.22: Successful adding information

Independent candidates are also added by admin using ‘ADD INDEPENDENT’ link in admin panel.

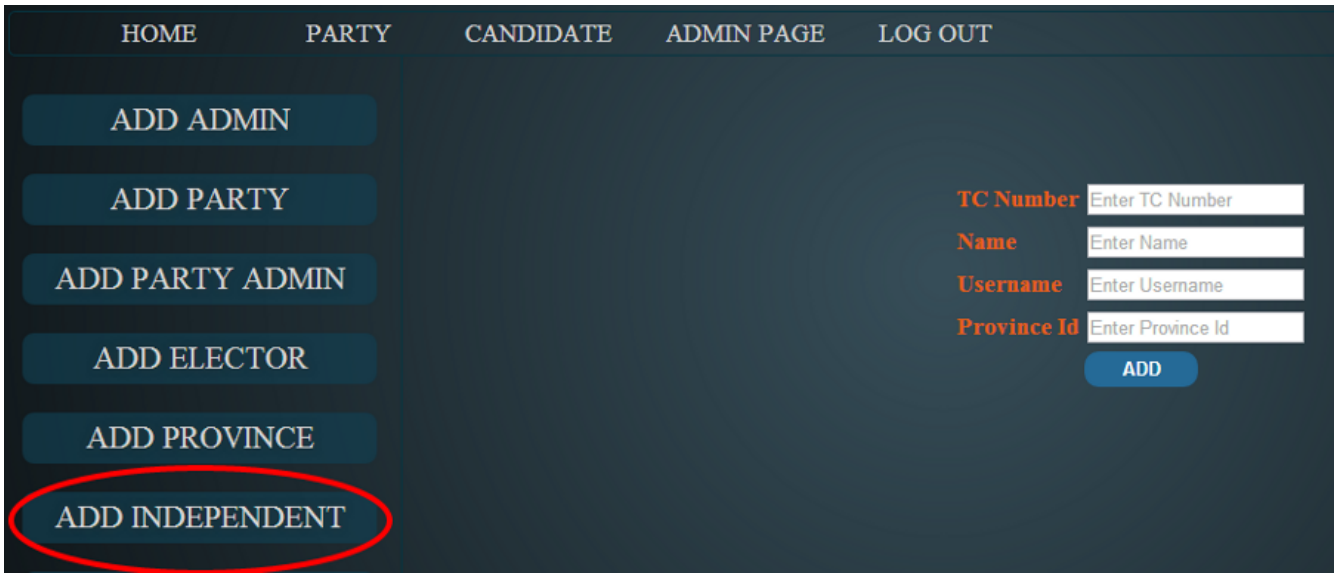


Figure 2.23: Independent candidate adding interface

After adding independent candidate successfully, an information message is shown.

Figure 2.24: Independent candidate successful adding information

Independent candidate information in database after adding.

	tcnumber [PK] character	name character var	username character var	partytitle character	isindependen boolean	electionid integer	provinceid integer
1	15448788321	CandidateA	acandidate		TRUE	1	16
*							

Figure 2.25: Independent candidate information in CANDIDATE table after add

Independent candidate is also added to USERS table.

	username [PK] character varying(80)	password character varying(80)	usertype integer
1	acandidate	48aae7037ba3	3
2	admin	5490514e5ef2	0
3	party	admin	1
4	root	root	0
*			

Figure 2.26: Independent candidate information in USERS table after add

can delete

- Admin
- Party
- Party Administrator
- Elector
- Province
- Election

Admin can delete admin using ‘DELETE ADMIN’ link in admin panel. After deleting admin successfully, an information message is shown.

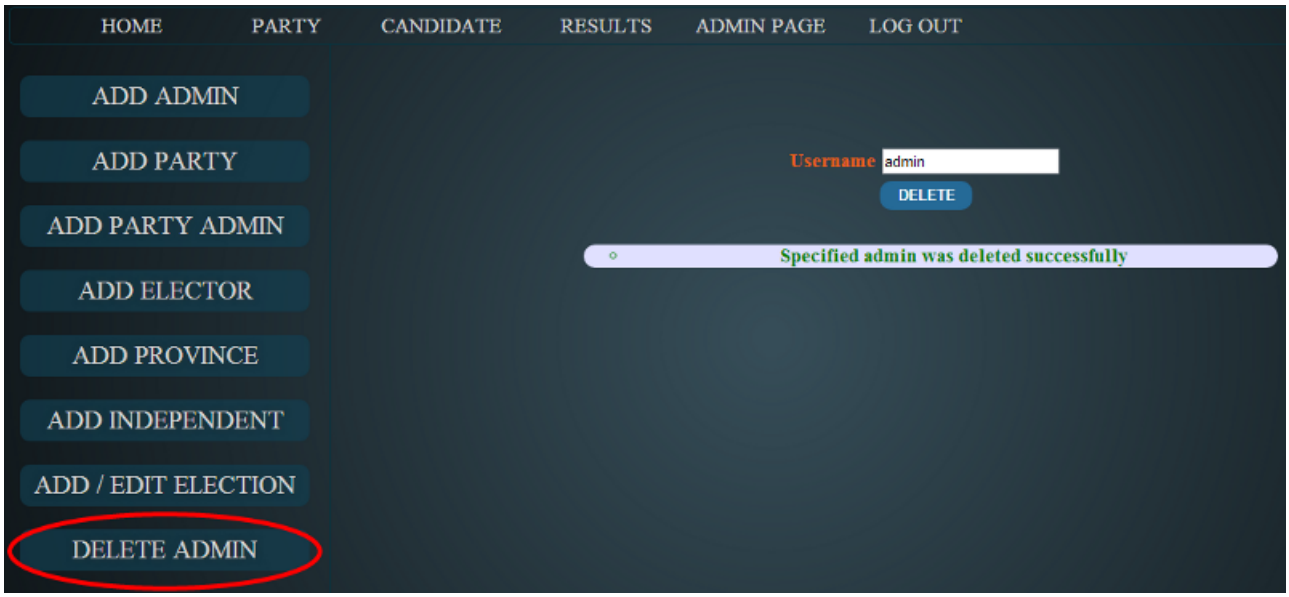


Figure 2.27: Admin deleting interface

can update

- Admin

- Party
- Party Administrator
- Province
- Election

Admin can update admin using ‘UPDATE ADMIN’ link in admin panel.

Admin information before update

	username [PK] character var	password character varying(80)	usertype integer
5	23974264808	23974264808	2
6	47832256971	47832256971	2
7	53269764808	53269764808	2
8	53269798658	53269798658	2
9	87412064806	87412064806	2
10	91235861232	91235861232	2
11	98653211232	98653211232	2
12	admin	d719c1b5571c	0
13	ahmetakkaya	ahmetakkaya	3
14	ahmetdemir	ahmetdemir	3

Figure 2.28: Admin information in database before update

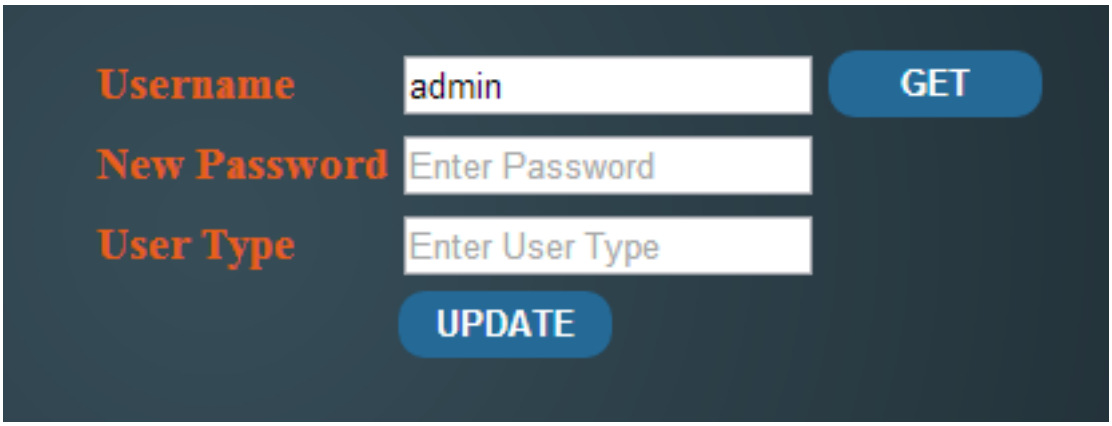


Figure 2.29: Admin updating interface

If admin types username and press get button, information about specified admin is taken from database and related fields are filled. Username can not be changed, since it is primary key.

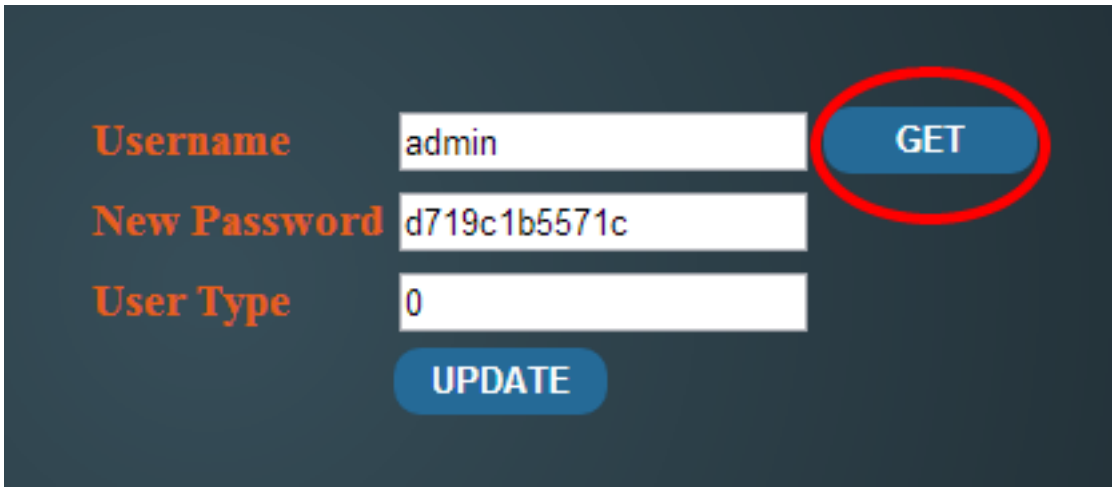


Figure 2.30: Getting admin's current information

Admin information after update

	username [PK] character var	password character varying(80)	usertype integer
5	23974264808	23974264808	2
6	47832256971	47832256971	2
7	53269764808	53269764808	2
8	53269798658	53269798658	2
9	87412064806	87412064806	2
10	91235861232	91235861232	2
11	98653211232	98653211232	2
12	admin	1234567	0
13	ahmetakkaya	ahmetakkaya	3
14	ahmetdemir	ahmetdemir	3

Figure 2.31: Admin information in database after update

If requested admin is not in database, an error message is shown like below.

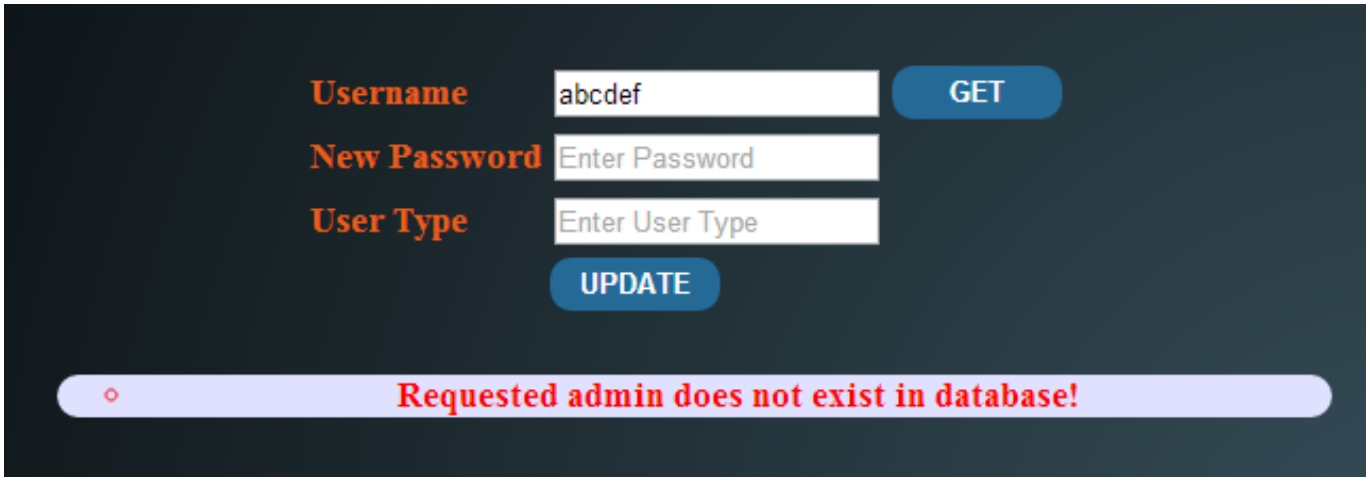


Figure 2.32: Wrong admin update

2.4 Party Page

Visitors may want to learn general information about parties such as name of party leader or principles of parties etc. Party page is created in order to meet this request. Visitors of the "National E-Voting System" can be see parties which join in next election.

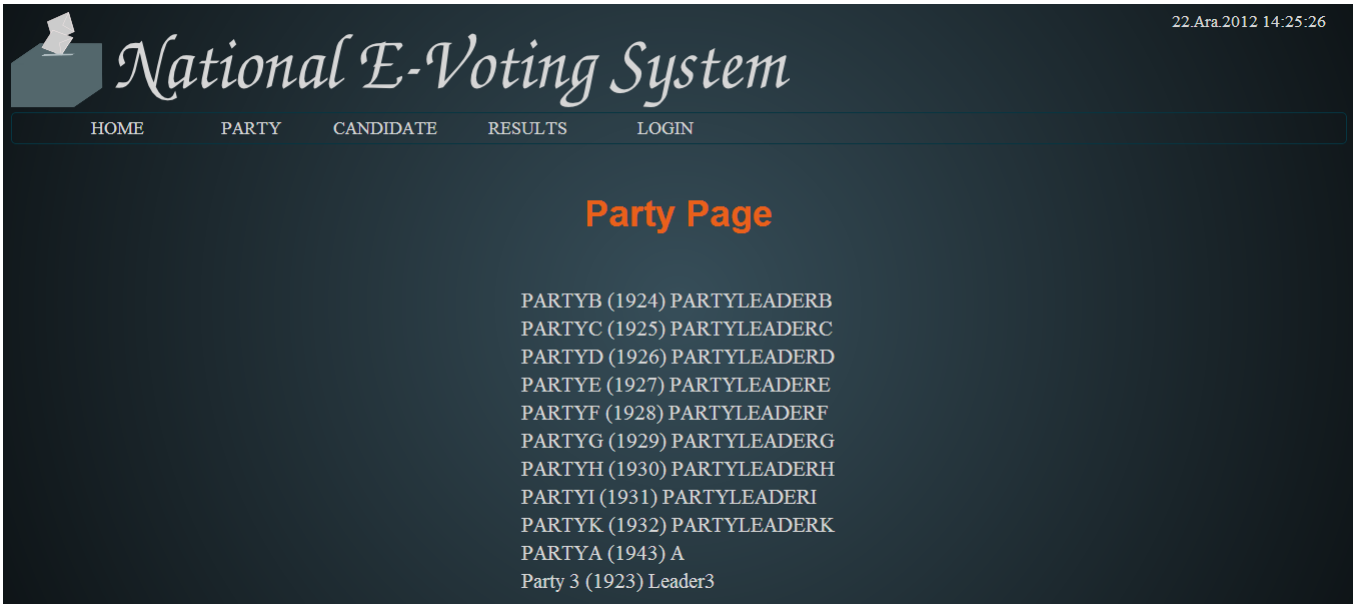


Figure 2.33: Party Page.

In this page, parties are sorted as a link. This links contain information of party name, founded year of party and name of party administrator. If visitors click any party, party information page is shown.

Figure 2.34: Party Information Page.

In party information page, visitors can learn founded year of party, name of party leader, election id of party, party principles and candidates of this party. In this page, candidates are sorted as a link. If visitors click any candidate, candidate information page is shown.

Figure 2.35: Candidate Information Page.

In candidate information page, visitors can learn information about candidate such as candidate's TC number, name, user name, party title, information of independence and province id.

2.5 Party Operations

Party operations contain add, delete and update operations for party informations. Admin can add, delete or update parties.

2.5.1 Party Edit Operation

Admin who want to add a party, need to click "ADD PARTY" button.

Figure 2.36: Party Edit Page.

A party can be added to only next election. If there is less than two weeks to start date of election, party can not be added. For adding a party, admin must fill the field of "Title of Party", "Year of Party" and "Party Leader". This information must be complete and accurate. For example, "Year of Party" is not a string. On the other hand, "Party Principle" information can be null. After adding a party, party information page is shown.

Figure 2.37: Party Information Page (After adding).

In this page, admin can see party information which is added new. In this way, the administrator can check the accuracy of the information. Since there is no candidate of this party, this part is empty.

2.5.2 Party Delete Operation

Administrator can delete a party with "DELETE PARTY" button.

Figure 2.38: Party Delete Page .

Party which administrator want to delete, can be deleted by entering the name of the party. When a party is deleted, candidates and administrators of this party are deleted. Admin can delete only parties of next election. Besides, if the election is terminated, parties can not be deleted. After deleting operation, party page is shown.

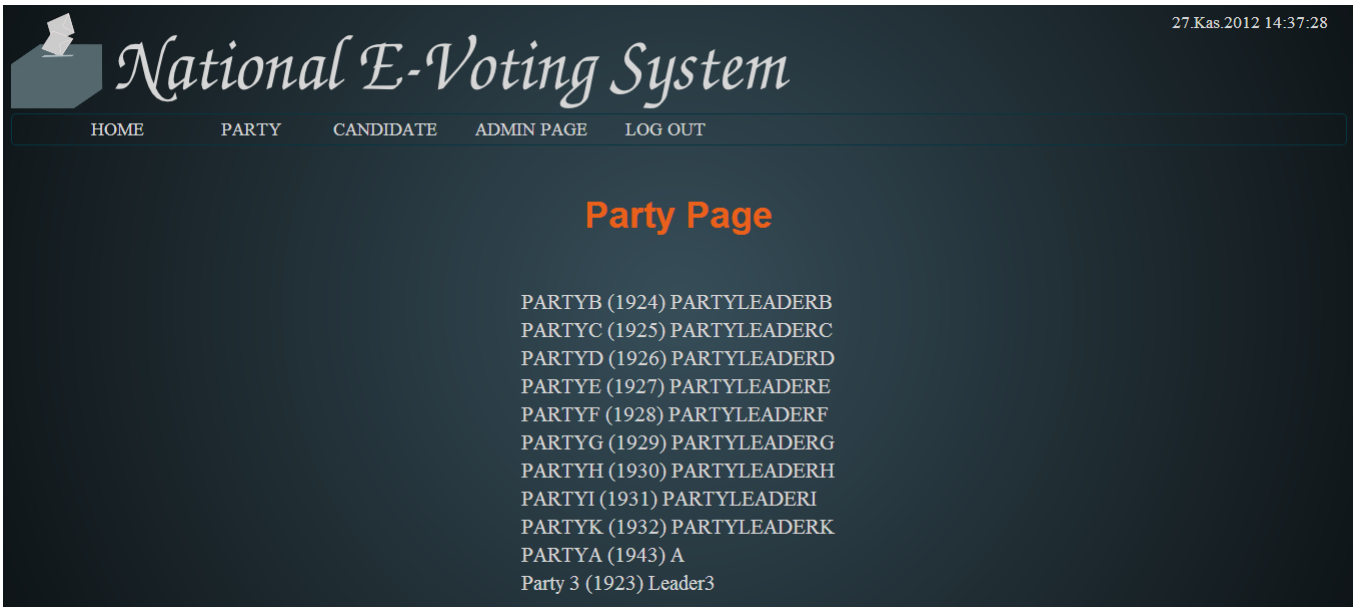


Figure 2.39: Party Delete Page .

In this page, administrator can see deleting operation which is whether successful or not.

2.5.3 Party Update Operation

Parties can be updated by administrator.

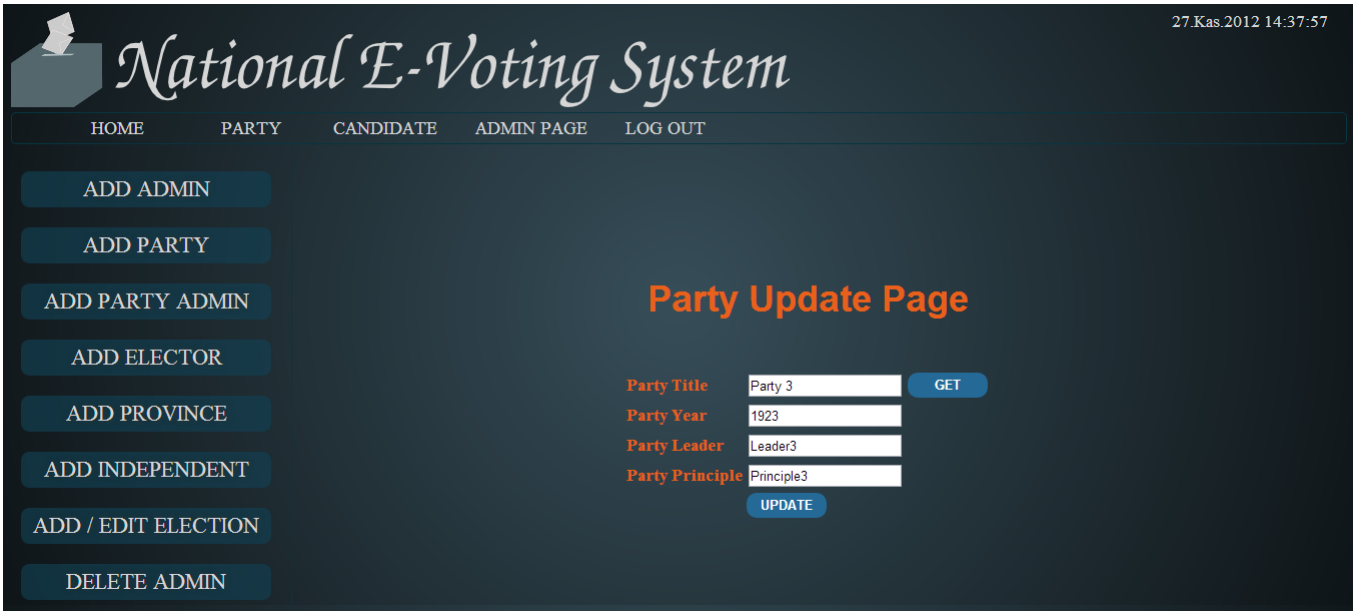


Figure 2.40: Party Update Page.

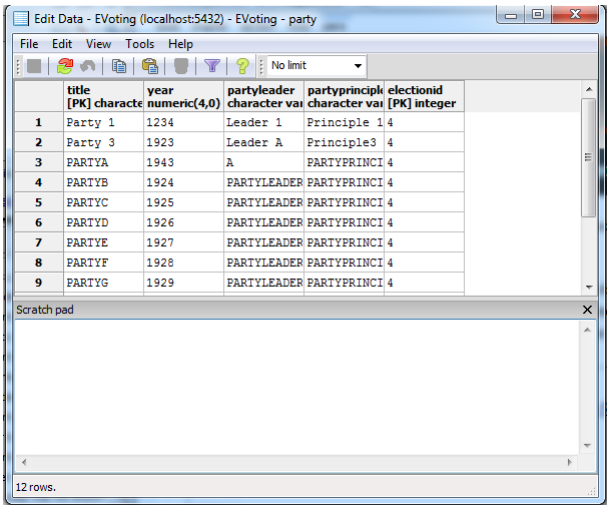
"Party Title" field is filled with party title and then information of this party is taken from database with "GET" button. After this step, administrator can update party title, party year, party leader or party principle. Admin can update only parties of next election. Besides, if there is less than two weeks to start date of election, party can not be updated.

Figure 2.41: Party Update Page (after updating)

After updating party information, "Specified party was updated successfully" is shown.

Note

This step could not been shown in demo. Because election id was changed to two in demo, but my all parties were in election four. Therefore, party could not been updated. Besides, when trying to change the election, a problem occurred in Apache Tomcat. Due to the limited time we must pass this step.



	title [PK] characte	year numeric(4,0)	partyleader character var	partyprincipl character var	electionid [PK] integer
1	Party 1	1234	Leader 1	Principle 1	4
2	Party 3	1923	Leader A	Principle3	4
3	PARTYA	1943	A	PARTYPRINCI	4
4	PARTYB	1924	PARTYLEADER	PARTYPRINCI	4
5	PARTYC	1925	PARTYLEADER	PARTYPRINCI	4
6	PARTYD	1926	PARTYLEADER	PARTYPRINCI	4
7	PARTYE	1927	PARTYLEADER	PARTYPRINCI	4
8	PARTYF	1928	PARTYLEADER	PARTYPRINCI	4
9	PARTYG	1929	PARTYLEADER	PARTYPRINCI	4

Figure 2.42: Party Database

This picture was added for showing that party update operation is successful. "Party 3" is updated successfully.

2.6 Party Administrator Operation

2.6.1 Party Administrator Edit Operation

Party operations contain add, delete and update operations for party informations. Admin can add, delete or update party administrator. If admin want to add a party administrator, "ADD PARTY ADMIN" button is clicked.

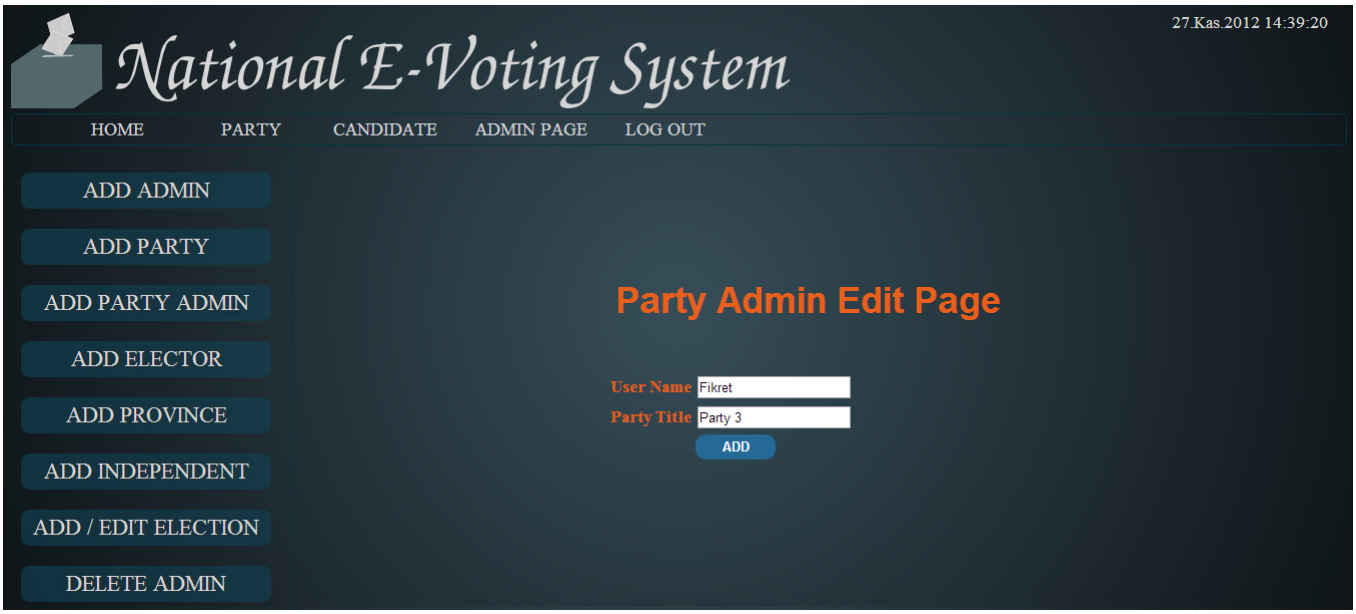


Figure 2.43: Party Administrator Edit Page

A party is needed for adding a party administrator. A party administrator can be added to only next election. Party administrator cannot be added if these party is not in this election. When a party administrator is added, this party administrator is added "PARTYADMINISTRATOR" table and also "USERS" table. In "USERS" table, password and type is assigned automatically. For adding a party administrator, admin must fill all fields. After adding a party administrator, "party administrator was added successfully" is shown.

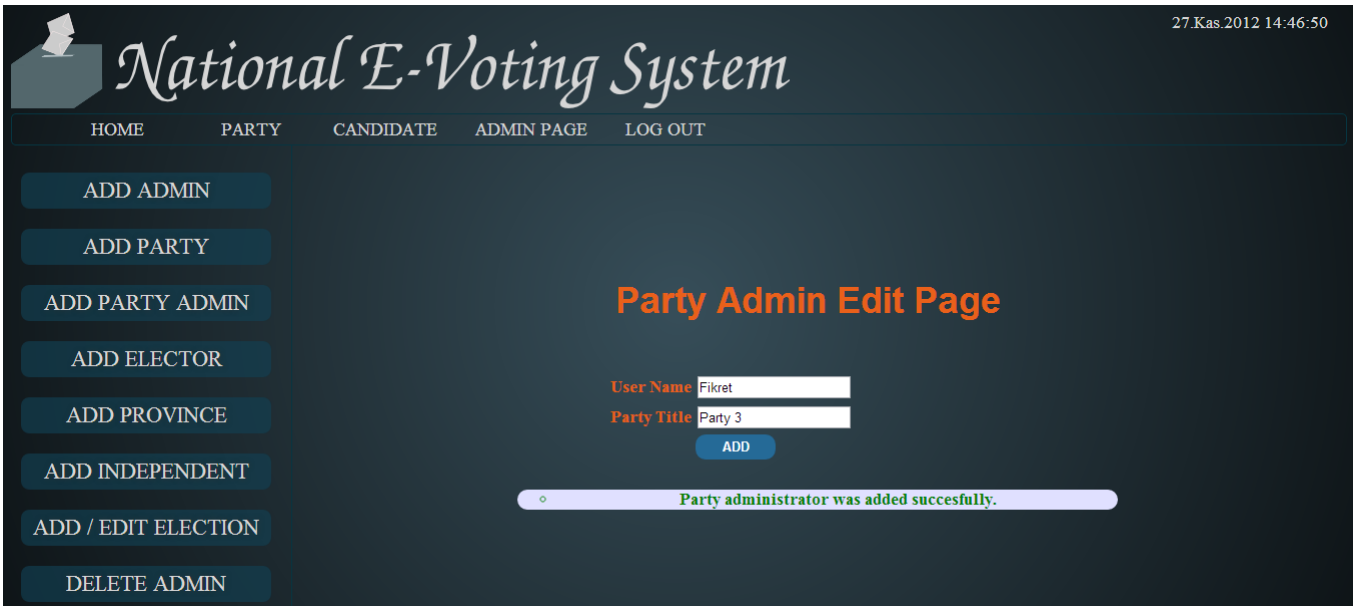


Figure 2.44: Party Administrator Edit Page (after adding)

Note
This step could not been shown in demo. Because election id was changed to two in demo, but my all parties were in election four. And Party administrator cannot be added if these party is not in this election. Therefore, party administrator was not added. Besides, when trying to change the election, a problem occurred in Apache Tomcat. Due to the limited time we must pass this step.

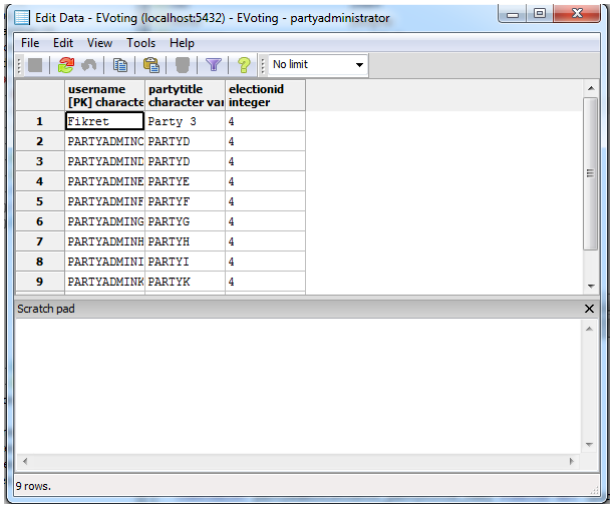


Figure 2.45: Party Administrator Database

This picture added for showing that party administrator add operation is successful. "Fikret" is added successfully.

2.6.2 Party Administrator Delete Operation

Admin can delete a party administrator with "DELETE PARTY ADMIN" button.

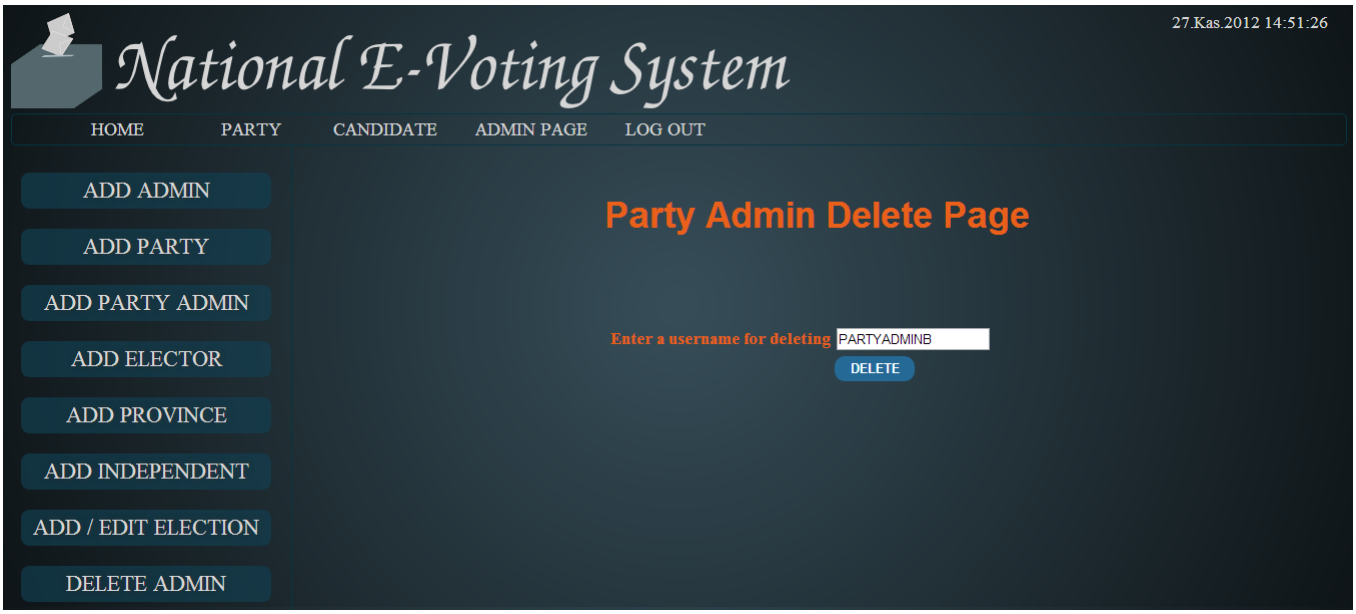
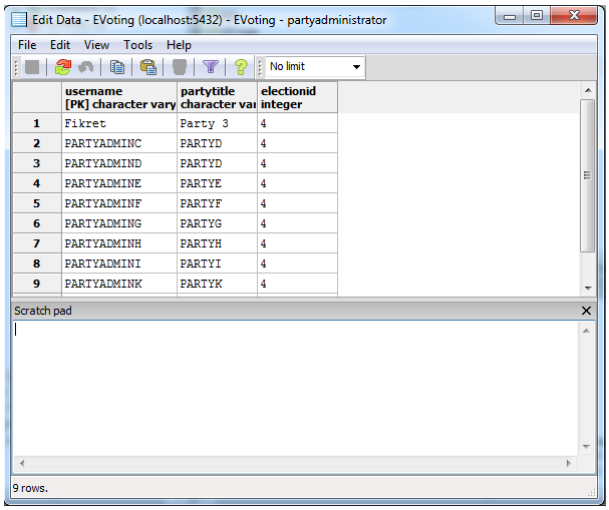


Figure 2.46: Party Administrator Delete Page

Party administrator can be deleted by entering the user name of the party administrator. When a party administrator is deleted, party administrator is deleted party "PARTYADMINISTRATOR" and "USERS" table in database. After deleting a party administrator, "party administrator was deleted successfully" information is shown.

Figure 2.47: Party Administrator Delete Page (After deleting)

Note
This step could not been shown in demo. Because, the reason described previously, this operation could not be shown. If we have time so much and if we could try, party admin delete operation could be shown successfully. Because we do not have a constraint about election id.



	username [PK] character vary	partytitle character va	electionid integer
1	Fikret	Party 3	4
2	PARTYADMINC	PARTYD	4
3	PARTYADMIND	PARTYD	4
4	PARTYADMINE	PARTYE	4
5	PARTYADMINF	PARTYF	4
6	PARTYADMING	PARTYG	4
7	PARTYADMINH	PARTYH	4
8	PARTYADMINI	PARTYI	4
9	PARTYADMINK	PARTYK	4

Figure 2.48: Party Administrator Database (After deleting)

This picture added for showing that party administrator delete operation is successful. "PARTYADMINB" is deleted successfully.

2.6.3 Party Administrator Update Operation

Admin can update a party administrator with "UPDATE PARTY ADMIN" button.

Figure 2.49: Party Administrator Update Page

"Username" field is filled and then information of this party administrator is taken from database with "GET" button. After this step, administrator can update party title of party administrator. Admin can update only information of party administrator of next election. On the other hand, if there is less than two weeks to start date of election, party administrator can be updated.

Figure 2.50: Party Administrator Update Page (After updating)

After updating party administrator information, "Specified party admin was updated successfully" information is shown.

Note

This step could not been shown in demo because of the reason described previously. If we did not have a some problems, namely if election could be taken election four, this operation could been worked successfully.

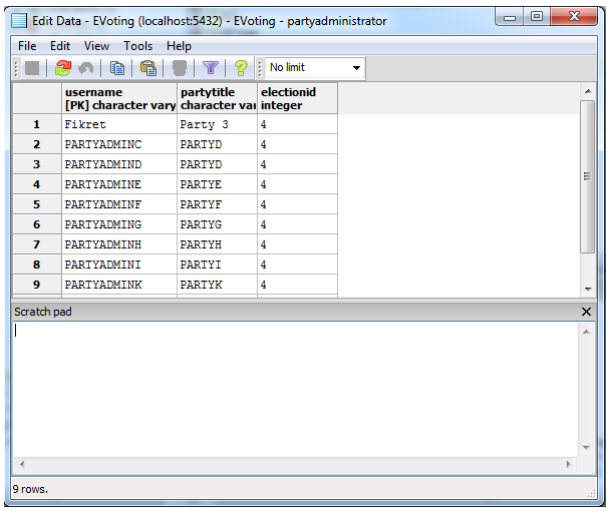


Figure 2.51: Party Administrator Database.

This picture added for showing that party administrator update operation is successful. "PARTYADMINC" is updated successfully. First case, party title of "PARTYADMINC" was "PARTYC".

2.7 Candidate Operation

Party administrator can update candidates of a party. For updating candidates, party administrator should be login the system as party admin. Party administrator can update candidates with "UPDATE CANDIDATE" button.

Figure 2.52: Party Administrator Database.

"TC Number" field is filled and then information of candidate is taken from database with "GET" button. After this step, party administrator can update candidate name or province id. Party administrator can update only information of candidate of next election. On the other hand, if there is less than two weeks to start date of election, candidates can not be updated.

Figure 2.53: Party Administrator Database.

When candidate information was updated successfully, "Specified candidate was updated successfully" information was shown.

2.8 Election Management

Admin has various jobs and one of them managing the election.

2.8.1 Adding Election

To add election admin must click the "ADD/EDIT ELECTION" link that is in the admin panel.

Figure 2.54: Add edit election form with tabbed panel.

There are some assumptions about election such as:

- There is at least one day difference between start date and finish date of election.
- Candidate adding task must be done at least 2 weeks earlier than the election start date.

According the these assumptions admin has to give proper date to text fields. Text fields are readonly and the way of giving input to textfields is using the datetimepicker. In datetimepicker, Past dates are not selectable and this property prevents the adding a election to past.

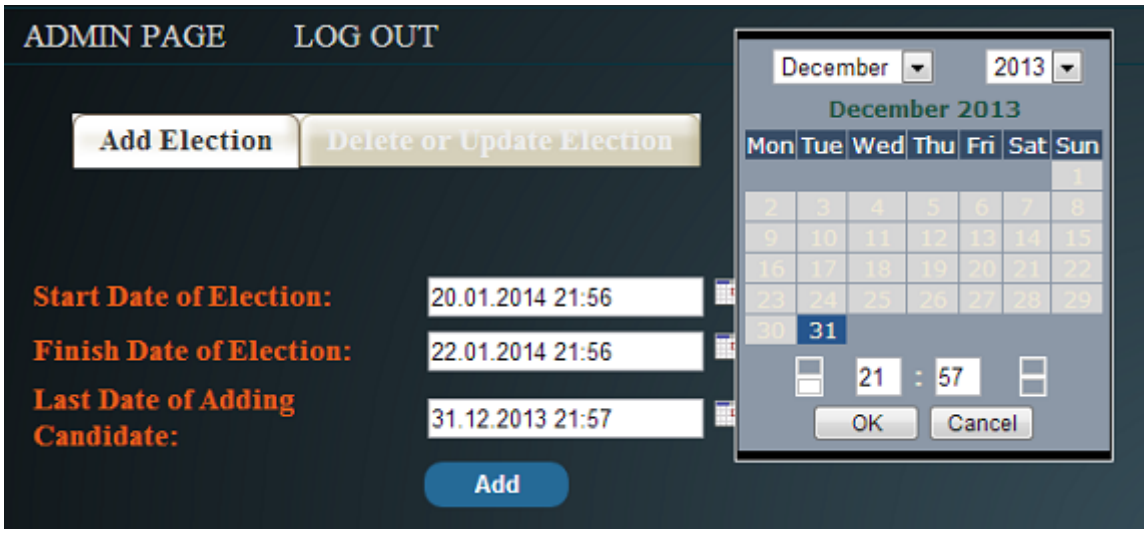


Figure 2.55: View of form when filling the textfields.

The process of adding election is completed by clicking the ADD button.

	electionid [PK] integer	startdateofel timestamp w	finishdateofe timestamp w	lastdateofad timestamp w
1	3	1998-04-03	1998-04-05	1998-02-21
2	4	2012-12-18	2013-01-01	2012-11-10
3	5	2014-02-10	2014-02-15	2014-01-10
4	6	2013-01-17	2013-01-19	2012-12-20
*				

Figure 2.56: Tuples in the ELECTION table before the add operation.

If the all textfields are filled with valid values a message that show the operation is finished successfully.

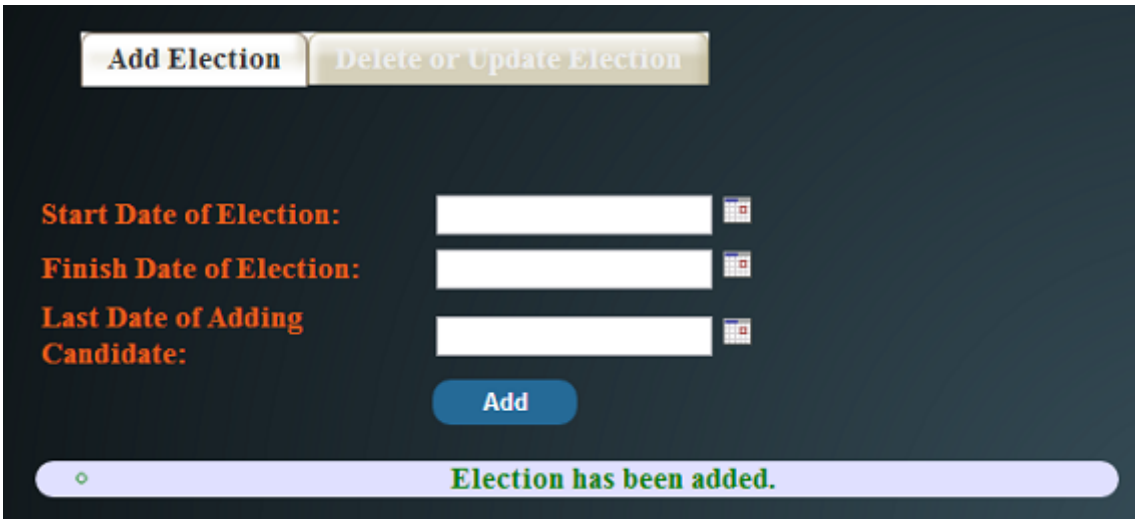


Figure 2.57: Message that show the operation is finished successfully.

It can be seen in the ELECTION table that the new election was added successfully.

	electionid [PK] integer	startdateofelection timestamp without time z	finishdateofelection timestamp without time	lastdateofaddingcandid timestamp without time
1	3	1998-04-03 19:00:00	1998-04-05 09:00:00	1998-02-21 18:00:00
2	4	2012-12-18 19:00:00	2013-01-01 19:00:00	2012-11-10 18:00:00
3	5	2014-02-10 14:00:00	2014-02-15 19:00:00	2014-01-10 08:00:00
4	6	2013-01-17 09:41:00	2013-01-19 09:42:00	2012-12-20 09:41:00
5	7	2014-01-20 21:56:00	2014-01-22 21:56:00	2013-12-31 21:57:00
*				

Figure 2.58: Tuples in the ELECTION table after the add operation.

If admin do not give the proper value for any textfields or leaves any of them blank, page is rendered again and an error message is shown with feedback panel.

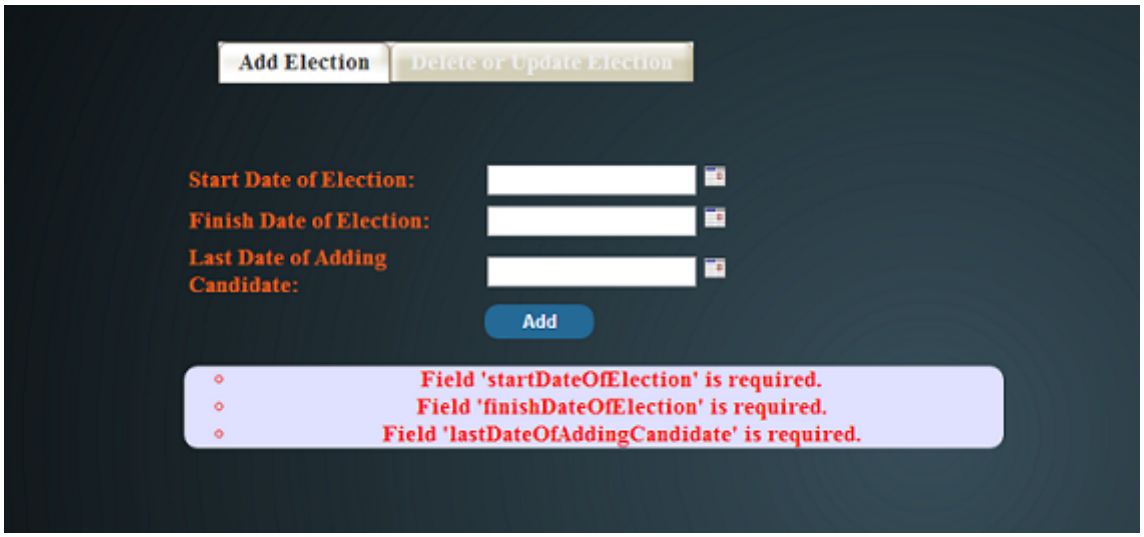


Figure 2.59: Error message for empty textfields.

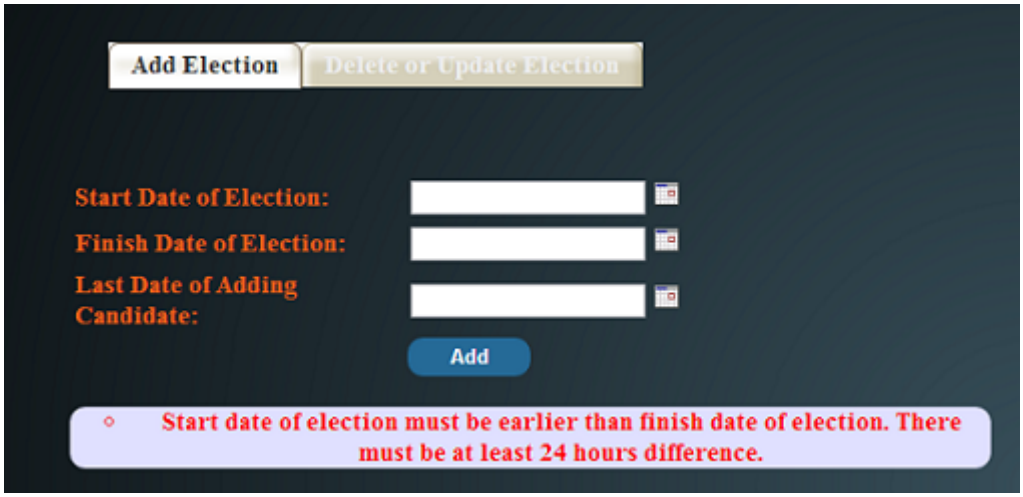


Figure 2.60: Error message for invalid value of finish date field.

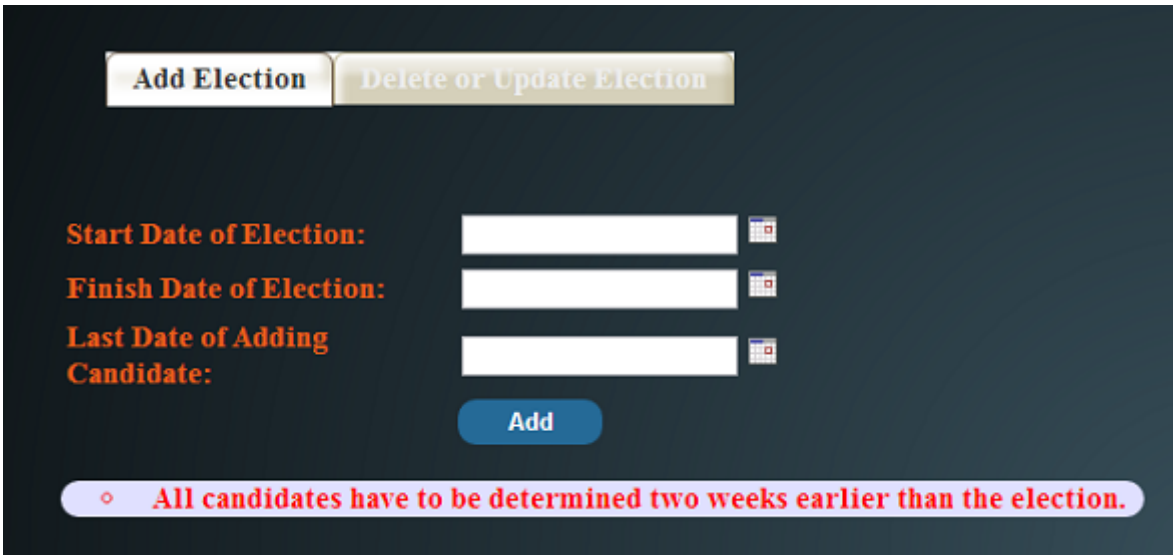


Figure 2.61: Error message for invalid value of last date of adding candidate field.

2.8.2 Deleting and Updating Election

Admin can delete or update the recorded elections. Admin can see the form for this job by clicking the "Delete or Update Election" tab. In that form recorded elections are listed in dropdown list.

Figure 2.62: List of recorded elections.

After an election is selected from dropdown list, textfields are filled automatically.

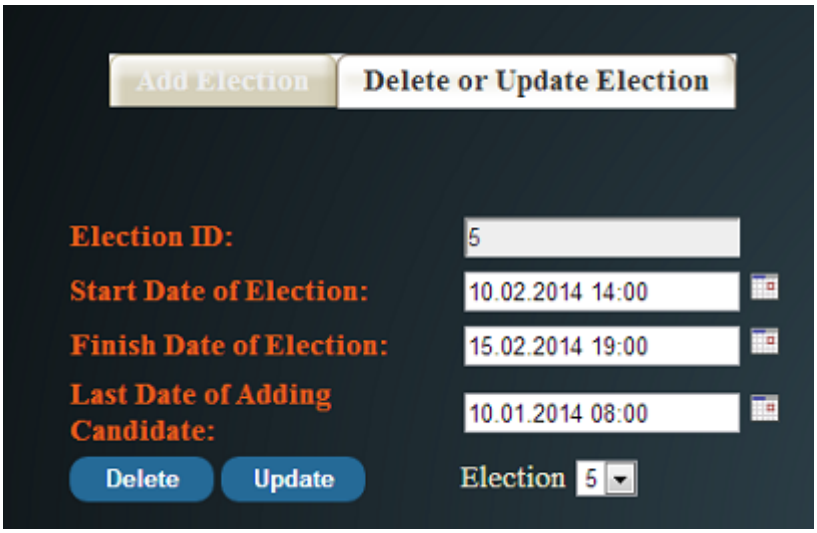


Figure 2.63: View of filled textfields.

Selected election can be deleted by clicking the delete operation. After that page is reloaded and id of election does not appear in the dropdown.

Figure 2.64: View of dropdown list after delete operation.

It can be seen in the ELECTION table that the selected election was deleted successfully.

	electionid [PK] integer	startdateofelection timestamp without time z	finishdateofelection timestamp without time	lastdateofaddingcandida timestamp without time
1	3	1998-04-03 19:00:00	1998-04-05 09:00:00	1998-02-21 18:00:00
2	4	2012-12-18 19:00:00	2013-01-01 19:00:00	2012-11-10 18:00:00
3	5	2014-02-10 14:00:00	2014-02-15 19:00:00	2014-01-10 08:00:00
4	6	2013-01-17 09:41:00	2013-01-19 09:42:00	2012-12-20 09:41:00
5	7	2014-01-20 21:56:00	2014-01-22 21:56:00	2013-12-31 21:57:00
*				

Figure 2.65: Tuples in the ELECTION table before the delete operation.

	electionid [PK] integer	startdateofel timestamp w	finishdateofe timestamp w	lastdateofad timestamp w
1	3	1998-04-03	1998-04-05	1998-02-21
2	4	2012-12-18	2013-01-01	2012-11-10
3	6	2013-01-17	2013-01-19	2012-12-20
4	7	2014-01-20	2014-01-22	2013-12-31
*				

Figure 2.66: Tuples in the ELECTION table after the delete operation.

Admin can update the selected election.If admin gives the proper value for field that admin want to change it, the operation is finished successfully; otherwise an error message is shown as add operation.

	electionid [PK] integer	startdateofelection timestamp without time	finishdateofelection timestamp without time	lastdateofaddingcandidate timestamp without time zo
1	3	1998-04-03 19:00:00	1998-04-05 09:00:00	1998-02-21 18:00:00
2	4	2012-12-18 19:00:00	2013-01-01 19:00:00	2012-11-10 18:00:00
3	6	2013-01-17 09:41:00	2013-01-19 09:42:00	2012-12-20 09:41:00
4	7	2014-01-20 21:56:00	2014-01-22 21:56:00	2013-12-31 21:57:00
*				

Figure 2.67: Tuples in the ELECTION table before the update operation.

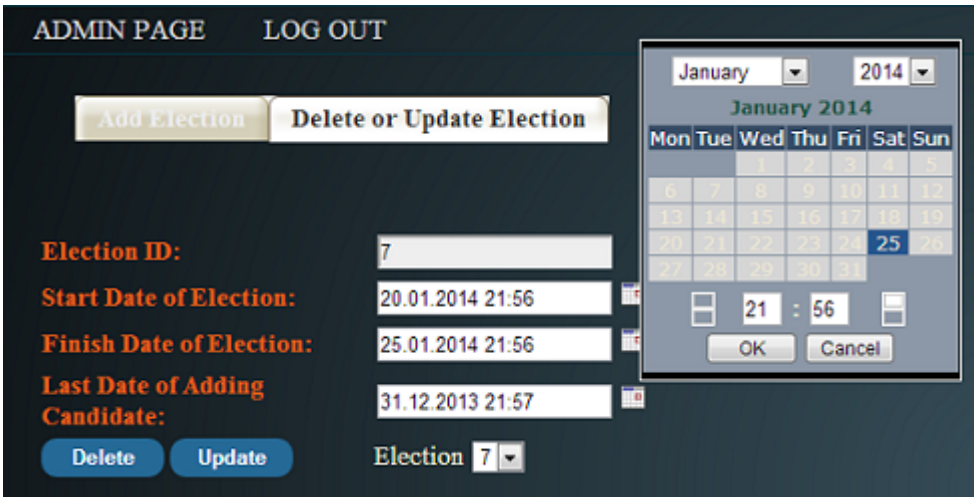


Figure 2.68: View of form that is for update operation.

Figure 2.69: Message that show the operation is finished successfully.

	electionid [PK] integer	startdateofelection timestamp without time	finishdateofelection timestamp without time	lastdateofaddingcandidate timestamp without time zo
1	3	1998-04-03 19:00:00	1998-04-05 09:00:00	1998-02-21 18:00:00
2	4	2012-12-18 19:00:00	2013-01-01 19:00:00	2012-11-10 18:00:00
3	6	2013-01-17 09:41:00	2013-01-19 09:42:00	2012-12-20 09:41:00
4	7	2014-01-20 21:56:00	2014-01-25 21:56:00	2013-12-31 21:57:00
*				

Figure 2.70: Tuples in the ELECTION table after the update operation.

2.9 Voting

Voting is the one of the important part of the election. If an elector logins and the election has started, "VOTE" link that is on the navigation bar is activated. When elector click the that link, if elector has voted before a message is shown to elector.

Figure 2.71: Error message for an elector who has already voted.

If election has not voted yet, the province id of obtain from session info of elector and parties and independent candidates who join the election from province of elector are list in a radio group. Elector can choose only one of them. By clicking the "Vote" button, the voting operation is completed. If elector tries to vote again, an error message is shown to elector.

Figure 2.72: View of voting operation.

Figure 2.73: Error message for repeated voting.

The changes on the "VOTE" table can be seen following figures.

	voteid [PK] serial	electionid integer	partytitle character var	provinceid integer	votetime timestamp without time zone
1	6	4	PARTYD	21	2012-12-27 10:26:41.16
2	7	4	CANDIDATEQ	6	2012-12-27 10:27:25.448
3	8	4	PARTYD	6	2012-12-27 10:28:47.132
4	9	4	PARTYH	6	2012-12-27 10:29:01.744
5	10	4	PARTYD	6	2012-12-27 10:29:15.335
6	11	4	PARTYA	6	2012-12-27 10:29:25.4
7	12	4	Party 3	6	2012-12-27 10:29:41.882
*					

Figure 2.74: Tuples in the VOTE table before the vote operation.

	voteid [PK] serial	electionid integer	partytitle character var	provinceid integer	votetime timestamp without time zone
1	6	4	PARTYD	21	2012-12-27 10:26:41.16
2	7	4	CANDIDATEQ	6	2012-12-27 10:27:25.448
3	8	4	PARTYD	6	2012-12-27 10:28:47.132
4	9	4	PARTYH	6	2012-12-27 10:29:01.744
5	10	4	PARTYD	6	2012-12-27 10:29:15.335
6	11	4	PARTYA	6	2012-12-27 10:29:25.4
7	12	4	Party 3	6	2012-12-27 10:29:41.882
8	13	4	CANDIDATEQ	6	2012-12-27 10:30:57.79
*					

Figure 2.75: Tuples in the VOTE table after the vote operation.

2.10 Results of Election

Results page is deactivated before the election and it is activated when election starts. There is a sliding bar which shows the general voting ratio on top of the results page. There is a message bar and that bar shows the congressman distribution in parliament. Final component is a Turkey map which has clickable area. When user clicks a province, a new page is opened for that province.

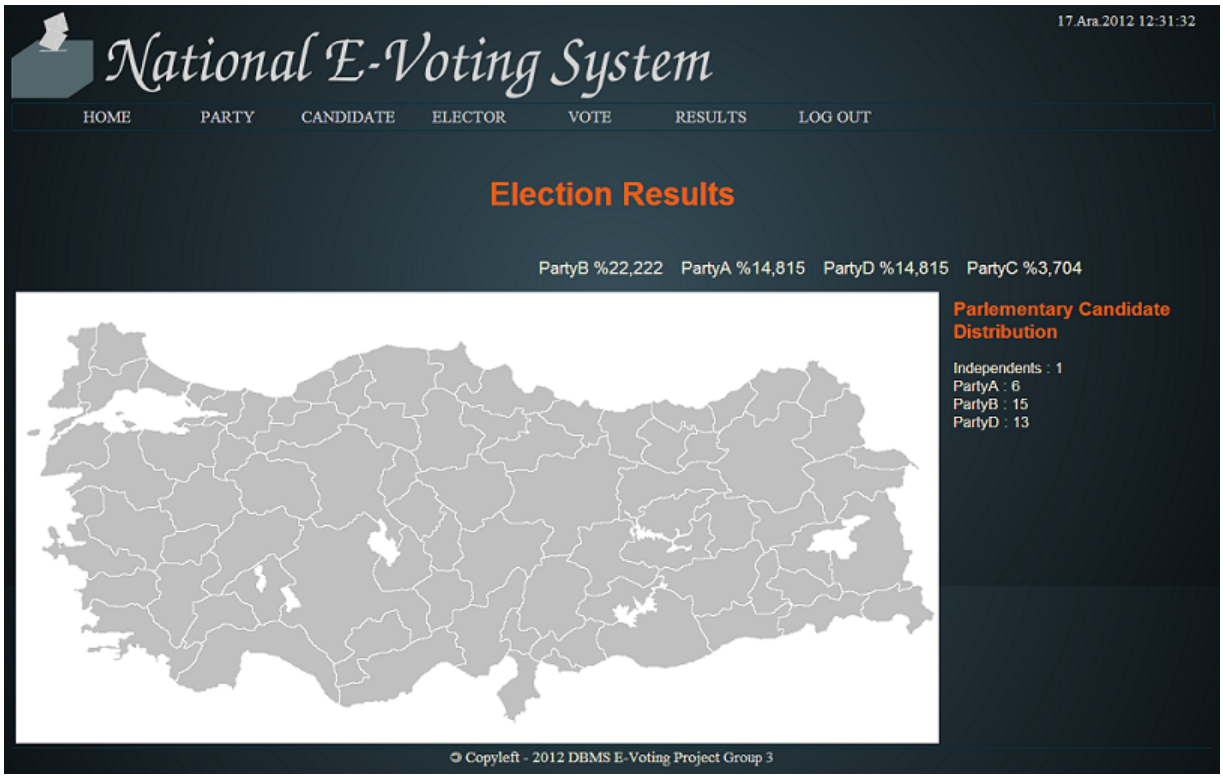


Figure 2.76: General view of the Results page.

The page for showing results for a province has two sliding message bar. One of them shows the vote ratio for parties and independent candidates who join the election from that province. Other message bar shows the congressman counts for that province. There is also a 3d chart which shows the vote ratio.

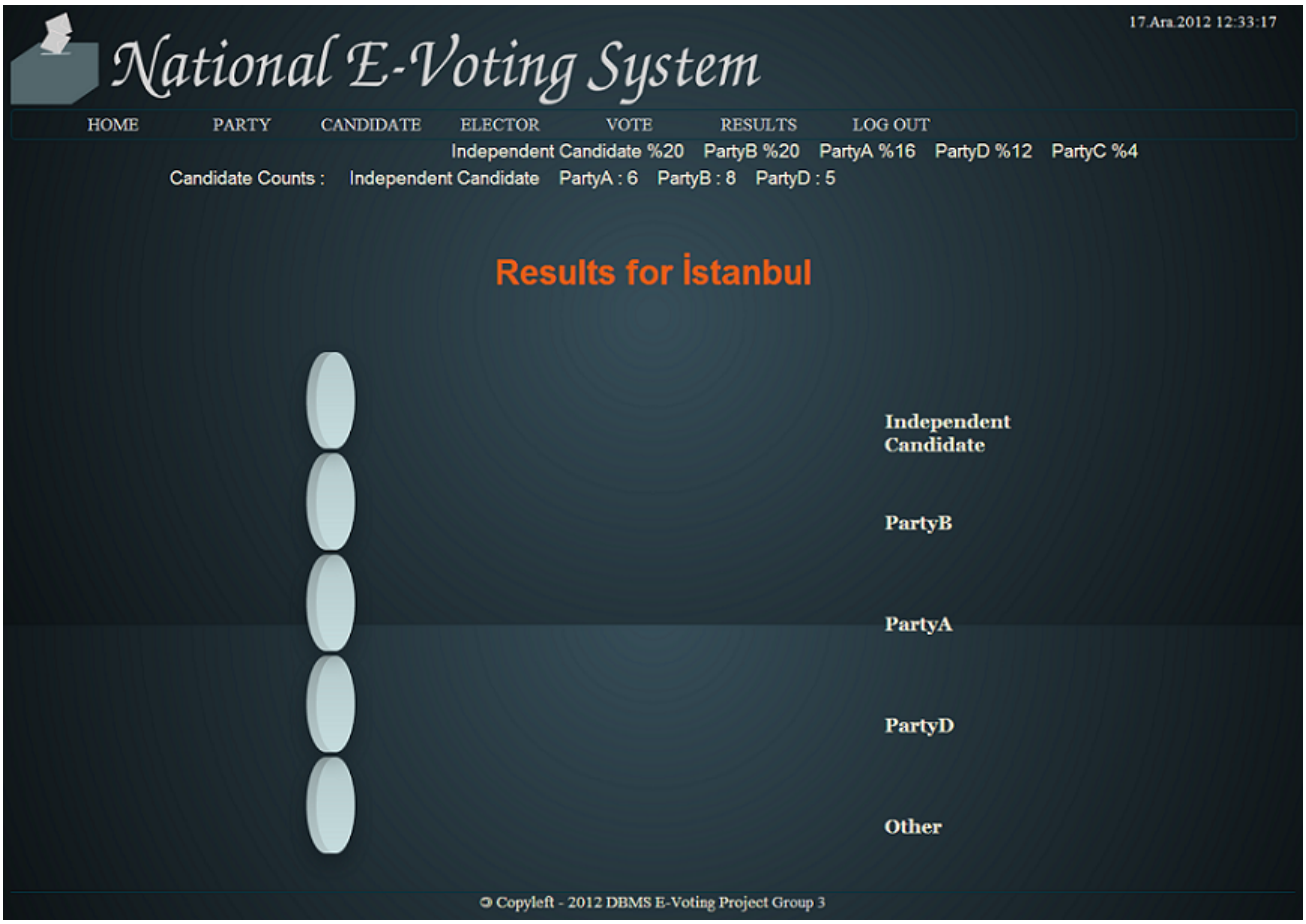


Figure 2.77: General view of the Results page for a province.

To see the values on the chart, mouse must be taken on bars.

Figure 2.78: 3d vote ratio chart.

Chapter 3

Technical Manual

3.1 Database Design

3.1.1 Elector and Province Tables

I have 2 tables in my part which are ELECTOR and PROVINCE. EER diagram of this project is given below, my part is marked with red pen.

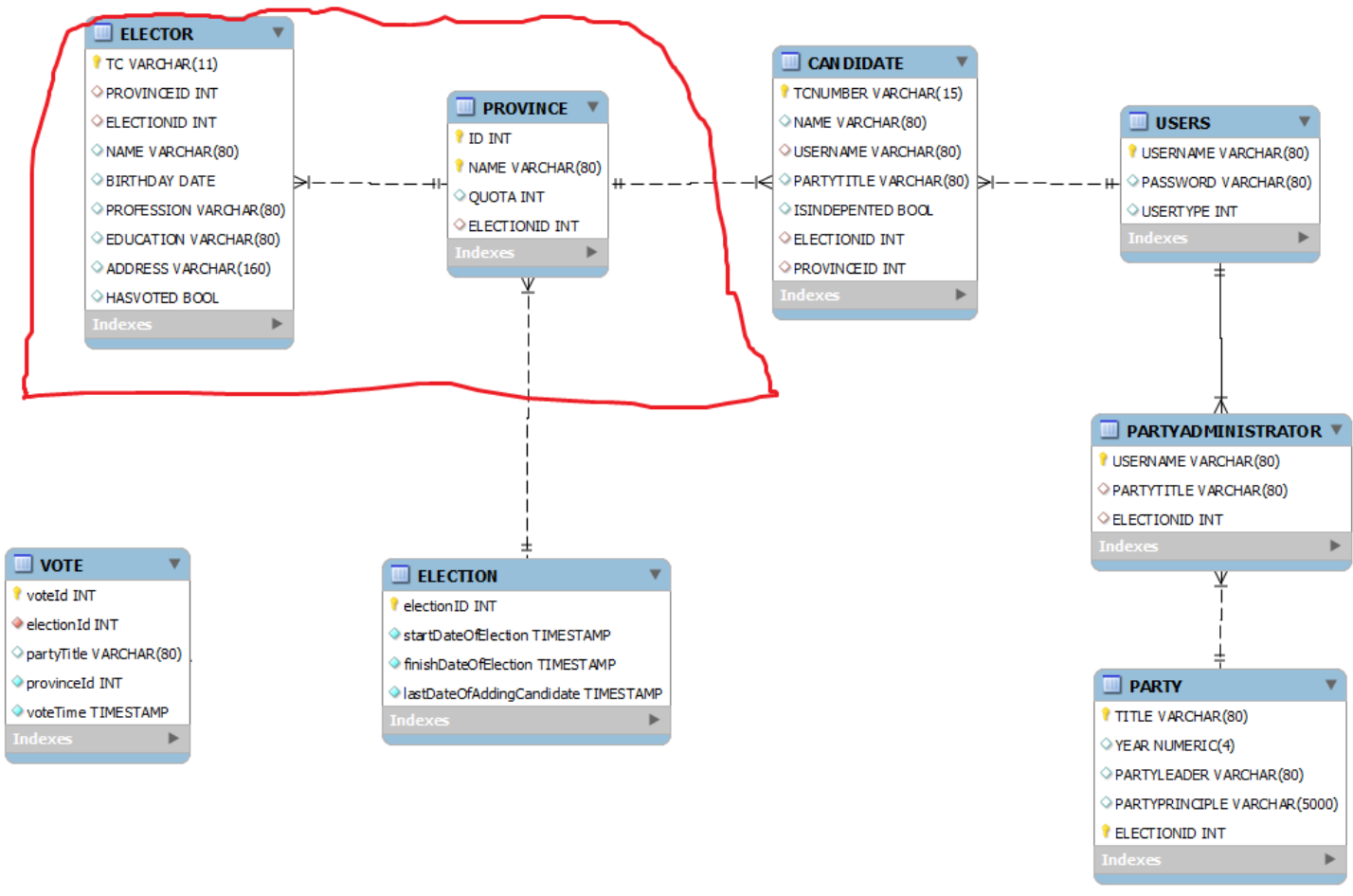


Figure 3.1: EER DIAGRAM

3.1.1.1 "Elector Table"

The table that keeps records of all registered electors. ProvinceID and ElectionID are foreign key to Province and Election Tables. On delete operation it will cascade other tables. All the attributes, keys information is below:

Name	Type	Not Null	Primary Key
TCnumber	VARCHAR(11)	1	1
ProvinceID	INTEGER	1	0
ElectionID	INTEGER	1	0
Name	VARCHAR(80)	1	0
Birthday	DATE	1	0
Profession	VARCHAR(80)	0	0
Education	VARCHAR(80)	0	0
Address	VARCHAR(160)	0	0
HASVOTED	BOOLEAN	0	0

Table 3.1: Elector

- TCnumber: Keeps TCnumber of Electors.It is an important attribute since this is the username of electors and it is the primary key of this table.
- ProvinceID: Keeps the province information of the Elector.This is a foreign key to Province table.
- ElectionID: Keeps the election information of the Elector. this is a foreign key to Election table.
- Name: Keeps the name of the Electors.
- Profession: Keeps the professions of the Electors.
- Education: Keeps the Education information of the Electors.
- Address: Keeps the address of Electors.
- Hasvoted: This shows the elector whether voted or not.

3.1.1.2 "Province Table"

This table is necessary because election results are dependent to province's quota. ElectionID is foreign key to ELECTION table. On delete operation, Election table will be cascaded. All the attributes, keys information is below:

- ID: Keeps Ids of Provinces.It is the primary key of this table.
- Name: Keeps the name of the Provinces.
- Quota: Keeps the number of parlimentarian for Provinces.
- ElectionID: This shows the election which this particular province will join. This is a foreign key to Election table.

Name	Type	Not Null	Primary Key
ID	INTEGER	1	1
Name	VARCHAR(80)	0	0
Quota	INTEGER	0	0
ElectionID	INTEGER	1	1

Table 3.2: Province

3.1.2 Login Operations and User

All kinds of users are stored in USERS table. There are 3 columns in this table:

- **username:** unique username for every user
- **password:** passwords of users
- **usertype:** 0(admin), 1(party admin), 2(elector) or 3(candidate)

Name	Data Type	Primary Key
USERNAME	VARCHAR(80)	1
PASSWORD	VARCHAR(80)	0
USERTYPE	INTEGER	0

Table 3.3: USERS

Username is primary key in USERS table, so it can not be null. It is also foreign key in candidate and party administrator tables. Before adding these, a user is added to USERS table.

Passwords are generated automatically using PasswordGenerator class.

When a user tries to login, information is taken from login form and compared with database. If there is a matching record, user is redirected to related page according to user type.

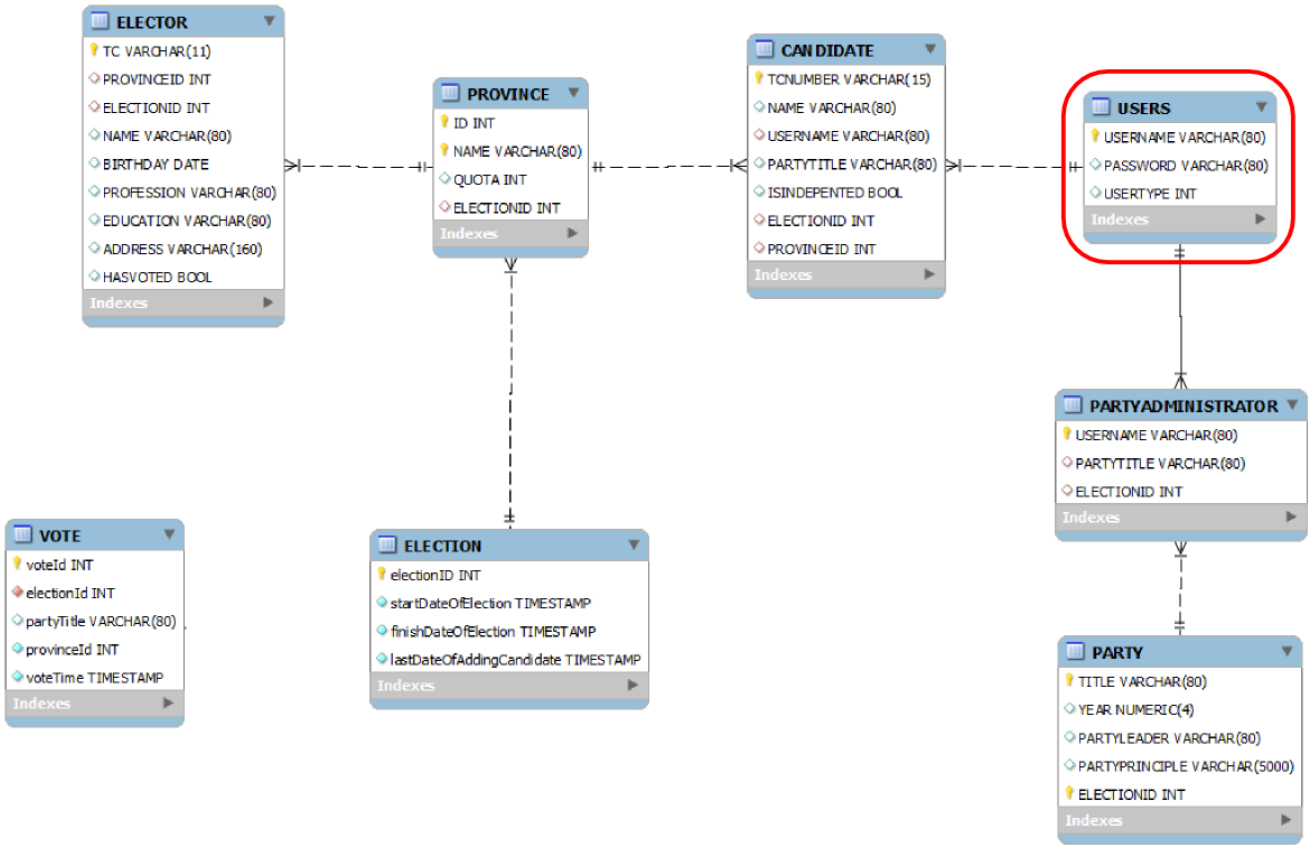


Figure 3.2: ER Diagram for USERS Table

3.1.3 Party Table

I create three database objects which are "PARTY", "PARTYADMINISTRATOR and "CANDIDATE".

Name	Type	Not Null	Primary Key	Foreign Key
TITLE	VARCHAR(80)	1	1	0
YEAR	NUMERIC(4)	0	0	0
PARTYLEADER	VARCHAR(80)	1	0	0
PARTYPRINCIPLE	VARCHAR(5000)	0	0	0
ELECTIONID	INTEGER	0	1	0

Table 3.4: PARTY

Information of party in the system is hold in the table above.

- TITLE: Used to hold the name of the party.
- YEAR: Used to hold founded year of the party.
- PARTYLEADER: Used to hold name of party leader.
- PARTYPRINCIPLE: Used to hold principles of party.
- ELECTIONID: Used to hold information of election of part.

3.1.4 Party Administrator Table

For PARTYADMINISTRATOR table:

Name	Type	Not Null	Primary Key	Foreign Key
USERNAME	VARCHAR(80)	1	1	1
PARTYTITLE	VARCHAR(80)	0	0	1
ELECTIONID	INTEGER	0	0	1

Table 3.5: PARTYADMINISTRATOR

Information of party administrator in the system is hold in the table above.

- USERNAME: Used to hold the user name of the party administrator. Foreign key for "USERNAME" in "USERS" table.
- PARTYTITLE: Used to hold the name of party of the party administrator. Foreign key for "TITLE" in "PARTY" table.
- ELECTIONID: Used to hold information of election of part administrator. Foreign key for "ELECTIONID" in "PARTY" table.

3.1.5 Candidate Table

For CANDIDATE table:

Informations of candidates in the system is hold in the table above.

- TCNUMBER: Used to hold the TC number of candidates.
- NAME: Used to hold name of candidates.
- USERNAME: Used to hold user name of candidates. Foreign key for "USERNAME" in "USERS" table.
- PARTYTITLE: Used to hold name of party of candidates.

Name	Type	Not Null	Primary Key	Foreign Key
TCNUMBER	VARCHAR(15)	1	1	0
NAME	VARCHAR(80)	0	0	0
USERNAME	VARCHAR(80)	0	0	1
PARTYTITLE	VARCHAR(80)	0	0	0
ISINDEPENDENT	BOOLEAN	0	0	0
ELECTIONID	INTEGER	0	0	1
PROVINCEID	INTEGER	0	0	1

Table 3.6: CANDIDATE

- ISINDEPENDENT: Used to hold that candidates is independent or not.
- ELECTIONID: Used to hold information of number of election. Foreign key for "ELECTIONID" in "PROVINCE" table.
- PROVINCEID: Used to hold province id of candidates. Foreign key for "ID" in "PROVINCE" table.

3.1.6 Entity - Relation Diagram

Entity - Relation Diagram:

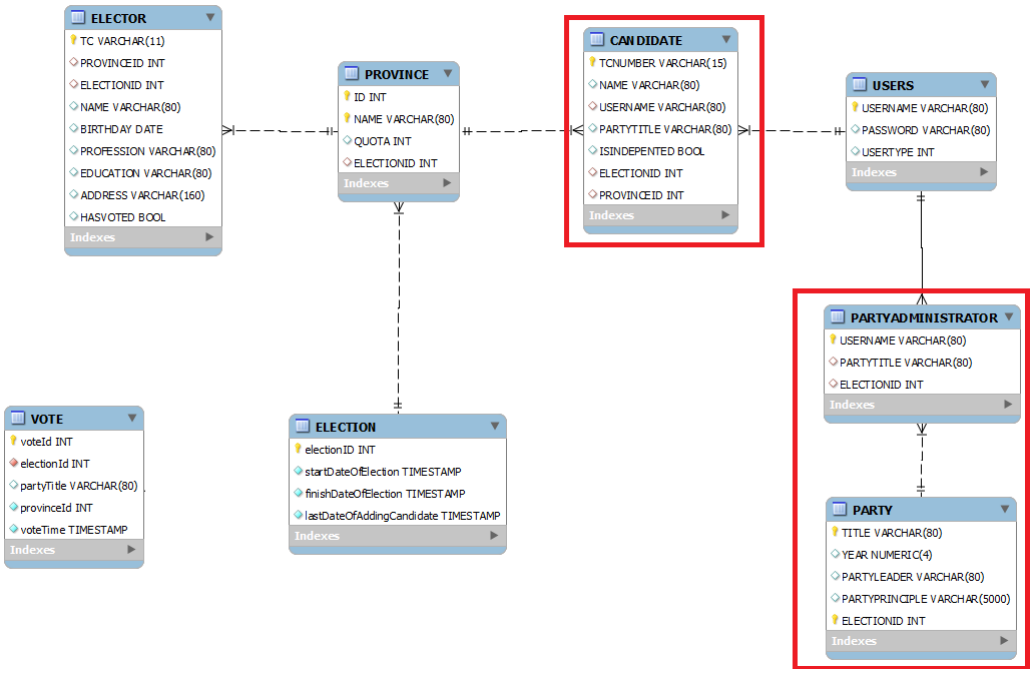


Figure 3.3: ER Diagram

3.1.7 ELECTION TABLE

ELECTION object has three dates which are start date of election, finish date of election and last date of adding candidate, and an id attribute to distinguish elections from each other.

Name	Type	Not Null	Primary Key
electionId	Integer	Not Null	1
startDateOfElection	Timestamp	Not Null	0
finishDateOfElection	Timestamp	Not Null	0
lastDateOfAddingCandidate	Timestamp	Not Null	0

Figure 3.4: ELECTION table

electionId attribute is foreign key in various table. According to this situation an election tuple must be added to database firstly. electionId is auto increment but the auto increment property is provided in the business layer not database layer.

3.1.8 VOTE table

Votes stored in "VOTE" table and there are 5 columns:

- voteId : it is used to distinguish the votes
- electionId : it is used to determine the owner of vote
- partyTile : shows that vote for which party or independent candidate
- provinceID : shows vote is valid for which province
- voteTime : voting time of that vote

voteId attribute is primary key for providing uniqueness and auto incremental. The auto increment property is provided in database layer. electionId must be foreign key, but in project it is certain that the given electionId for that vote is in database, so no need the determining it as foreign key. partyTitle attribute can be null because it is possible that elector wants to vote empty vote. provinceId must also be foreign key, but again it is certain that the given province id is in the database. Because they are given by system to insert operation.

Name	Type	Not Null	Primary Key
voteId	Serial	Not Null	1
electionId	Integer	Not Null	0
partyTitle	VARCHAR(80)	Null	0
provinceId	Integer	Not Null	0
voteTime	Timestamp	Not Null	0

Figure 3.5: VOTE table

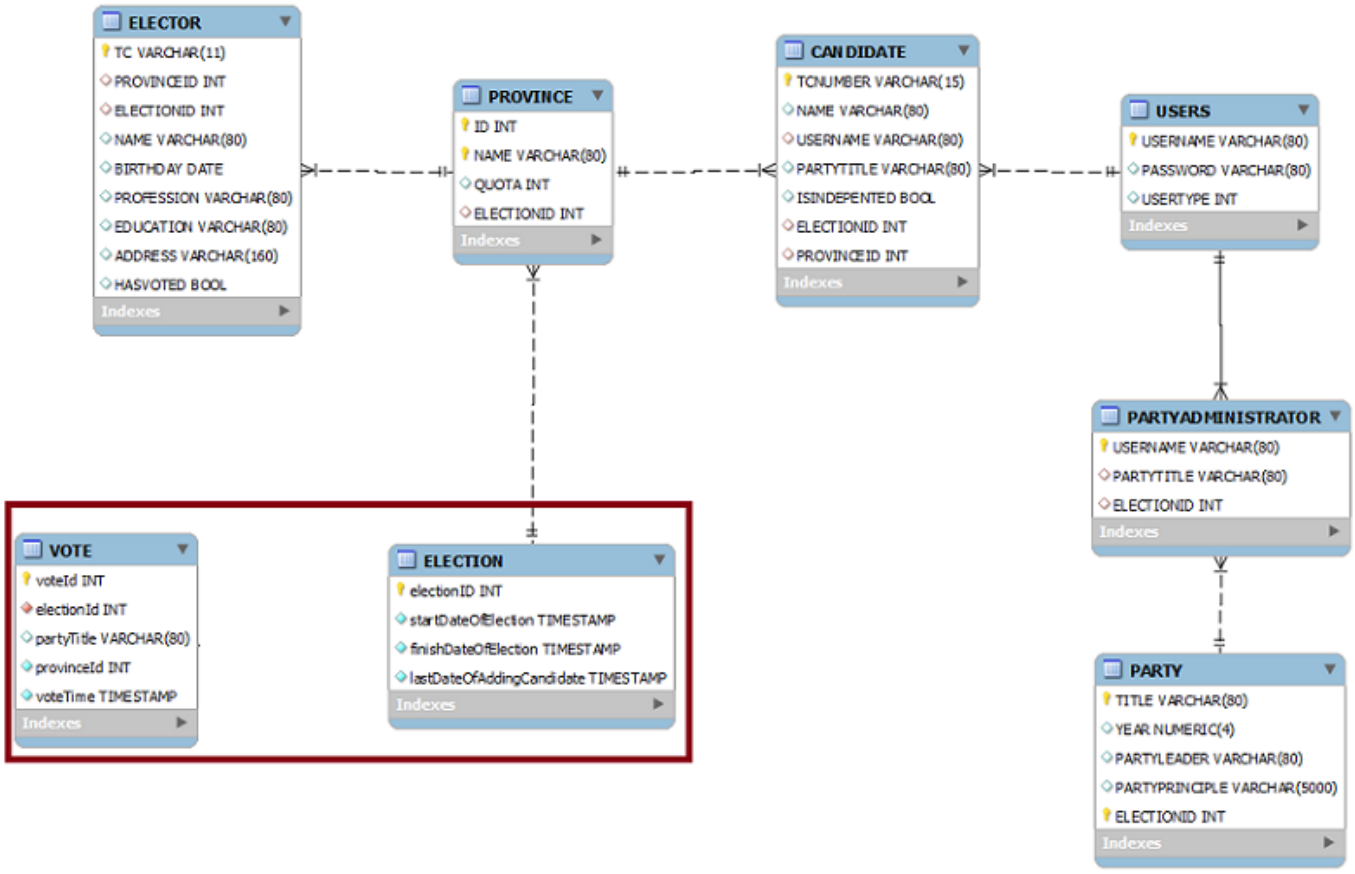


Figure 3.6: General design of Database as er diagram

3.2 Software Design

3.2.1 Visual Classes

3.2.1.1 Elector Page

Elector page is the page that elector sees after login to system. There is 6 label in the page to print the elector information which comes from database such as TCnumber, province, address etc. Province name doesn't kept in elector table. That's why i convert province id information to province name using ElectionOrganizer class functions.

3.2.1.2 Elector Update Page

Elector update page is the page that elector sees after clicking "UPDATE MY INFORMATION" button. This page extended from Base Page because elector should see this page. Form class has been used to implement update form. TCnumber of elector is pulled from Mysession class with MySession.get().getUser().getUsername() call. Since TCNumber is username of elector.

There is 6 field in the form which defined as not required field. RequiredTextField and TextField classes of wicket has been used. Also FeedBackPanel class of wicket is added for error handling.

GET button overrided and all information is pulled from database and inserted into fields for easyness. Update Button overrided and call updateElector function from ElectorDM.

3.2.1.3 Elector Add Page

Elector add page is the page extended from AdminBasePage since it should be only visible to admin. All form features are the same with update page.

3.2.1.4 Elector Delete Page

Elector delete page is the page extended from AdminBasePage since it should be only visible to admin. All form features are the same with update page. Admin can delete an elector with entering elector TC number.

3.2.1.5 Province Update Page

There is 3 field in the form which defined as not required field. RequiredTextField and TextField classes of wicket has been used. Also FeedBackPanel class of wicket is added for error handling.

GET button overrided and all information is pulled from database and inserted into fields for easyness. Update Button overrided and call updateElector function from ElectorDM.

3.2.1.6 Province Add Page

Province add page is the page extended from AdminBasePage since it should be only visible to admin. All form features are the same with update page.

3.2.1.7 Province Delete Page

Province delete page is the page extended from AdminBasePage since it should be only visible to admin. All form features are the same with update page. Admin can delete an province with entering province id.

3.2.2 Other Classes

3.2.2.1 Elector

Elector has several attributes. All getter setter methods have been implemented and a constructor which takes all attributes is implemented

```
            /*
                     * To change this template, choose Tools | Templates
                     * and open the template in the editor.
                     */
                    package itucs.blg361e.EVoting.Elector;

                    import java.io.Serializable;
                    import java.sql.Date;

                    /**
                     *
                     * @author Bahti
                     */

                    /*
                     * To change this template, choose Tools | Templates
                     * and open the template in the editor.
                     */
                    public final class Elector implements Serializable {

                        private String TCnumber = null;
```

```
    private Integer electionID = null;
    private Integer provinceID = null;
    private String name = null;
    private Date birthday = null;
    private String profession = null;
    private String education = null;
    private String address = null;
    private Boolean hasVoted = false;

    public Elector() {
    }

    public Elector(String TCnumber, Integer provinceid, String name,  ↩
        Date bday, String prof, String Edu, String ads) {

        this.setTC(TCnumber);
        this.setProvinceID(provinceid);
        this.setName(name);
        this.setBirthday(bday);
        this.setAddress(ads);
        this.setEducation(Edu);
        this.setProfession(prof);



    }

    public Integer getElectionID() {
        return electionID;
    }

    public void setElectionID(Integer electionID) {
        this.electionID = electionID;
    }

    public Boolean getHasVoted() {
        return hasVoted;
    }

    public void setHasVoted(Boolean hasVoted) {
        this.hasVoted = hasVoted;
    }

    public void setName(String nm) {

        this.name = nm;
    }

    public void setBirthday(Date bd) {

        this.birthday = bd;
    }

    public void setProfession(String p) {

        this.profession = p;
    }

    public void setTC(String tc) {
        this.TCnumber = tc;

    }
```

```
    public void setEducation(String e) {

        this.education = e;
    }

    public void setAddress(String a) {

        this.address = a;
    }

    public void setProvinceID(Integer p) {

        this.provinceID = p;
    }

    public String getName() {

        return this.name;
    }

    public String getAddress() {

        return this.address;
    }

    public Date getBirthday() {

        return this.birthday;
    }

    public String getProfession() {

        return this.profession;
    }

    public String getEducation() {

        return this.education;
    }

    public String getTCNumber() {
        return this.TCnumber;
    }

    public Integer getProvinceID() {
        return this.provinceID;
    }
}
```

3.2.2.2 Elector DM

Elector has several database operations such as delete-update-search-add. All these operations are handled by ElectorDM class and its methods.

ElectorDM have methods like deleteElector, getOneElector(String TC) etc. In each method, connect() function is called to make connection with database and at the end close() function is called to close connection. All necessary sql queries is implemented and runned over database using PreparedStatement Class.

```
/*
 * To change this template, choose Tools | Templates
```

```
 * and open the template in the editor.
 */
package itucs.blg361e.EVoting.Elector;

/**
 *
 * @author Bahti
 */
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
import itucs.blg361e.EVoting.PostgreSqlManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.ArrayList;

public class ElectorDM {

    private static transient Connection db;
    private static transient PostgreSqlManager psm;

    public ElectorDM() {
        psm = new PostgreSqlManager();
    }

    public void connect() {
        ElectorDM.psm = new PostgreSqlManager();
        ElectorDM.db = psm.getConnection("jdbc:postgresql://localhost ↩
            :5432/EVoting", "postgres", "123456");
    }

    public static void createTable() {
        try {
            ElectorDM.psm = new PostgreSqlManager();
            ElectorDM.db = psm.getConnection("jdbc:postgresql:// ↩
                localhost:5432/EVoting", "postgres", "123456");

            if (!psm.doesTableExists("ELECTOR")) {
                Statement state = db.createStatement();
                String createTable = "CREATE TABLE ELECTOR(TC VARCHAR ↩
                    (11) PRIMARY KEY NOT NULL,PROVINCEID INTEGER, ↩
                    ELECTIONID INTEGER, NAME VARCHAR(80),BIRTHDAY DATE, ↩
                    PROFESSION VARCHAR(80),EDUCATION VARCHAR(80),ADDRESS ↩
                     VARCHAR(160),HASVOTED BOOLEAN, FOREIGN KEY( ↩
                    PROVINCEID,ELECTIONID) REFERENCES PROVINCE(ID, ↩
                    ELECTIONID) ON UPDATE NO ACTION ON DELETE CASCADE )" ↩
                    ;
                state.executeQuery(createTable);
                state.close();
                ElectorDM.db.close();
            }
        } catch (SQLException ex) {
            Logger.getLogger(ElectorDM.class.getName()).log(Level. ↩
                SEVERE, null, ex);
        } catch (Exception ex) {
            Logger.getLogger(ElectorDM.class.getName()).log(Level. ↩
```

```
                    SEVERE, null, ex);
            }
        }

        public static void addElector(Elector e) {
            try {
                ElectorDM.db = psm.getConnection("jdbc:postgresql:// ↩
                    localhost:5432/EVoting", "postgres", "123456");

                String query = "INSERT INTO ELECTOR (TC, PROVINCEID,  ↩
                    ELECTIONID, NAME, BIRTHDAY, PROFESSION, EDUCATION,  ↩
                    ADDRESS,HASVOTED) VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?)";
                PreparedStatement state = ElectorDM.db.prepareStatement( ↩
                    query);
                state.setString(1, e.getTCNumber());
                state.setInt(2, e.getProvinceID());
                state.setInt(3, e.getElectionID());
                state.setString(4, e.getName());
                state.setDate(5, e.getBirthday());
                state.setString(6, e.getProfession());
                state.setString(7, e.getEducation());
                state.setString(8, e.getAddress());
                state.setBoolean(9, e.getHasVoted());
                state.executeUpdate();
                state.close();
                ElectorDM.db.close();
            } catch (SQLException ex) {
                Logger.getLogger(ElectorDM.class.getName()).log(Level. ↩
                    SEVERE, null, ex);
            }

        }

        public static void deleteElector(Elector e) {
            try {
                ElectorDM.db = psm.getConnection("jdbc:postgresql:// ↩
                    localhost:5432/EVoting", "postgres", "123456");
                String query = "DELETE FROM ELECTOR WHERE(TC = ?)";
                PreparedStatement state = ElectorDM.db.prepareStatement( ↩
                    query);
                state.setString(1, e.getTCNumber());
                state.executeUpdate();
                state.close();
                ElectorDM.db.close();
            } catch (SQLException ex) {
                Logger.getLogger(ElectorDM.class.getName()).log(Level. ↩
                    SEVERE, null, ex);
            }
        }

        public static void updateElector(Elector e) {
            try {
                ElectorDM.db = psm.getConnection("jdbc:postgresql:// ↩
                    localhost:5432/EVoting", "postgres", "123456");
                String query = "UPDATE ELECTOR SET PROVINCEID=? , NAME = ?  ↩
                    , BIRTHDAY=?, PROFESSION=?, EDUCATION=?, ADDRESS=? ,  ↩
                    HASVOTED=? WHERE(TC=?)";
                PreparedStatement state = ElectorDM.db.prepareStatement( ↩
                    query);
                state.setInt(1, e.getProvinceID());
                state.setString(2, e.getName());
                state.setDate(3, e.getBirthday());
```

```
                state.setString(4, e.getProfession());
                state.setString(5, e.getEducation());
                state.setString(6, e.getAddress());
                state.setBoolean(7, e.getHasVoted());
                state.setString(8, e.getTCNumber());
                state.executeUpdate();
                state.close();
                ElectorDM.db.close();
            } catch (SQLException ex) {
                Logger.getLogger(ElectorDM.class.getName()).log(Level. ↩
                    SEVERE, null, ex);
            }
        }

        public static ArrayList Elector getElector() {
            try {
                ElectorDM.psm = new PostgreSqlManager();
                ArrayList Elector electors = new ArrayList Elector();
                ElectorDM.db = psm.getConnection("jdbc:postgresql:// ↩
                    localhost:5432/EVoting", "postgres", "123456");
                String query = "SELECT * FROM ELECTOR";
                Statement statement = ElectorDM.db.createStatement();
                ResultSet rs = statement.executeQuery(query);
                while (rs.next()) {
                    Elector e = new Elector();
                    e.setName(rs.getString("NAME"));
                    e.setTC(rs.getString("TC"));
                    e.setProvinceID(rs.getInt("PROVINCEID"));
                    e.setBirthday(rs.getDate("BIRTHDAY"));
                    e.setProfession(rs.getString("PROFESSION"));
                    e.setEducation(rs.getString("EDUCATION"));
                    e.setAddress(rs.getString("ADDRESS"));
                    e.setHasVoted(rs.getBoolean("HASVOTED"));
                }
                rs.close();
                statement.close();
                ElectorDM.db.close();

                return electors;
            } catch (SQLException ex) {
                Logger.getLogger(ElectorDM.class.getName()).log(Level. ↩
                    SEVERE, null, ex);
                return null;
            }
        }

        public static Elector getOneElector(String TC) {
            Elector elector = null;

            try {
                ElectorDM.psm = new PostgreSqlManager();
                ElectorDM.db = psm.getConnection("jdbc:postgresql:// ↩
                    localhost:5432/EVoting", "postgres", "123456");

                String query = "SELECT * FROM ELECTOR WHERE(TC = ?)";
                PreparedStatement statement = db.prepareStatement(query);
                statement.setString(1, TC);
                ResultSet rs = statement.executeQuery();
                if (rs.next()) {
                    elector = new Elector();
                    elector.setTC(rs.getString("TC"));
                    elector.setProvinceID(rs.getInt("PROVINCEID"));
```

```
                elector.setName(rs.getString("NAME"));
                elector.setBirthday(rs.getDate("BIRTHDAY"));
                elector.setProfession(rs.getString("PROFESSION"));
                elector.setEducation(rs.getString("EDUCATION"));
                elector.setAddress(rs.getString("ADDRESS"));
                elector.setHasVoted(rs.getBoolean("HASVOTED"));
            }

            rs.close();
            statement.close();
            db.close();
        } catch (SQLException e) {
            throw new UnsupportedOperationException(e.getMessage());
        } catch (Exception ex) {
            Logger.getLogger(ElectorDM.class.getName()).log(Level. ↩
                SEVERE, null, ex);
        } finally {
            return elector;
        }
    }

    public static ResultSet searchElector() {
        try {
            ElectorDM.psm = new PostgreSqlManager();
            ElectorDM.db = psm.getConnection("jdbc:postgresql:// ↩
                localhost:5432/EVoting", "postgres", "123456");

            String query = "SELECT * FROM ELECTOR";
            Statement state;
            state = ElectorDM.db.createStatement();
            state.executeQuery(query);
            ResultSet r = state.getResultSet();

            return r;
        } catch (SQLException ex) {
            Logger.getLogger(ElectorDM.class.getName()).log(Level. ↩
                SEVERE, null, ex);
            return null;
        }
    }
}
```

3.2.2.3 ElectorCollection

This class keeps a list of electors. It has a constructor, getElectors, addElector and deleteElector methods as in the figure.

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package itucs.blg361e.EVoting.Elector;

/**
 *
 * @author bahti
 */
import java.util.LinkedList;
import java.util.List;

public class ElectorCollection {
```

```
        private List Elector electors;

        public ElectorCollection() {
            this.electors = new LinkedList Elector();
        }

        public List Elector getElectors() {
            return this.electors;
        }

        public void addElector(Elector e) {
            this.electors.add(e);
        }

        public void deleteElector(Elector e) {
            this.electors.remove(e);
        }
    }
```

3.2.2.4 Elector Display Page Link

This class makes a link to Elector Page.

```
import org.apache.wicket.markup.html.link.Link;

        public class ElectorDisplayPageLink extends Link {

            private Elector elector;

            public ElectorDisplayPageLink(String id, Elector e) {
                super(id);
                this.elector = e;
            }

            @Override
            public void onClick() {
                this.setResponsePage(new ElectorPage());
            }
        }
```

3.2.2.5 Province

Province has several attributes. All getter setter methods have been implemented and a constructor which takes all attributes is implemented

```
package itucs.blg361e.EVoting.Province;

        import java.io.Serializable;

        /**
         *
         * @author bahti
         */
        public class Province implements Serializable {

            private Integer id = null;
            private String name = null;
```

```
                private Integer quota = null;
                private Integer electionId = null;

                public Province() {
                }

                public Province(Integer id, String Name, Integer limit, Integer ↩
                     electionId) {
                    this.setName(Name);
                    this.setQuota(limit);
                    this.setId(id);
                    this.setElectionId(electionId);
                }

                public void setId(Integer a) {
                    this.id = a;
                }

                public Integer getId() {
                    return id;
                }

                public void setElectionId(Integer i) {
                    this.electionId = i;
                }

                public Integer getElectionId() {
                    return electionId;
                }

                public void setName(String Name) {
                    this.name = Name;
                }

                public String getName() {
                    return this.name;
                }

                public void setQuota(Integer number) {
                    this.quota = number;
                }

                public Integer getQuota() {
                    return this.quota;
                }
            }
```

3.2.2.6 Province DM

Province has several database operations such as delete-update-search-add. All these operations are handled by ProvinceDM class and its methods.

ProvinceDM have methods like deleteProvince, getOneProvince(Integer ID, Integer ElectionID) etc. In each method, connect() function is called to make connection with database and at the end close() function is called to close connection. All necessary sql queries is implemented and runned over database using PreparedStatement Class.

```
            /*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
```

```
 */
package itucs.blg361e.EVoting.Province;

import itucs.blg361e.EVoting.PostgreSqlManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.logging.Level;

import java.util.logging.Logger;

/**
 *
 * @author bahti
 */
public class ProvinceDM {

    private static transient Connection db;
    private static transient PostgreSqlManager psm;

    public ProvinceDM() {
        psm = new PostgreSqlManager();
    }

    public void connect() {
        ProvinceDM.psm = new PostgreSqlManager();
        ProvinceDM.db = psm.getConnection("jdbc:postgresql://localhost:5432/EVoting", " ↩
            postgres", "123456");
    }

    public static void createTable() {
        try {
            ProvinceDM.psm = new PostgreSqlManager();
            ProvinceDM.db = psm.getConnection("jdbc:postgresql://localhost:5432/EVoting", " ↩
                postgres", "123456");
            if (!psm.doesTableExists("PROVINCE")) {
                Statement state = db.createStatement();
                String createTable = "CREATE TABLE PROVINCE(ID INTEGER NOT NULL,NAME  ↩
                    VARCHAR(80),QUOTA INTEGER,ELECTIONID INTEGER REFERENCES ELECTION ON  ↩
                    DELETE CASCADE,PRIMARY KEY(ID,ELECTIONID))";
                state.executeQuery(createTable);
                state.close();
                ProvinceDM.db.close();
            }
        } catch (SQLException ex) {
            Logger.getLogger(ProvinceDM.class.getName()).log(Level.SEVERE, null, ex);
        } catch (Exception ex) {
            Logger.getLogger(ProvinceDM.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public static void addProvince(Province p) {
        try {
            ProvinceDM.db = psm.getConnection("jdbc:postgresql://localhost:5432/EVoting", " ↩
                postgres", "123456");
            String query = "INSERT INTO PROVINCE (ID,NAME, QUOTA,ELECTIONID) VALUES(?, ?,  ↩
                ?, ?)";
            PreparedStatement state = ProvinceDM.db.prepareStatement(query);
            state.setInt(1, p.getId());
```

```
            state.setString(2, p.getName());
            state.setInt(3, p.getQuota());
            state.setInt(4, p.getElectionId());
            state.executeUpdate();
            state.close();
            ProvinceDM.db.close();
        } catch (SQLException ex) {
            Logger.getLogger(ProvinceDM.class.getName()).log(Level.SEVERE, null, ex);
        }

    }

    public static void deleteProvince(Province p) {
        try {
            ProvinceDM.db = psm.getConnection("jdbc:postgresql://localhost:5432/EVoting", " ↩
                postgres", "123456");
            String query = "DELETE FROM PROVINCE WHERE(ID = ? AND ELECTIONID= ?)";
            PreparedStatement state = ProvinceDM.db.prepareStatement(query);
            state.setInt(1, p.getId());
            state.setInt(2, p.getElectionId());
            state.executeUpdate();
            state.close();
            ProvinceDM.db.close();
        } catch (SQLException ex) {
            Logger.getLogger(ProvinceDM.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public static void updateProvince(Province p) {
        try {
            ProvinceDM.db = psm.getConnection("jdbc:postgresql://localhost:5432/EVoting", " ↩
                postgres", "123456");
            String query = "UPDATE PROVINCE SET NAME = ? , QUOTA=? WHERE(ID=? AND  ↩
                ELECTIONID= ?)";
            PreparedStatement state = ProvinceDM.db.prepareStatement(query);
            state.setString(1, p.getName());
            state.setInt(2, p.getQuota());
            state.setInt(3, p.getId());
            state.setInt(4, p.getElectionId());
            state.executeUpdate();
            state.close();
            ProvinceDM.db.close();
        } catch (SQLException ex) {
            Logger.getLogger(ProvinceDM.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public static ArrayList Province getProvince() {
        try {
            ProvinceDM.psm = new PostgreSqlManager();
            ArrayList Province provinces = new ArrayList Province();
            ProvinceDM.db = psm.getConnection("jdbc:postgresql://localhost:5432/EVoting", " ↩
                postgres", "123456");
            String query = "SELECT * FROM PROVINCE";
            Statement statement = ProvinceDM.db.createStatement();
            ResultSet r = statement.executeQuery(query);
            while (r.next()) {
                Province p = new Province();
                p.setId(r.getInt("ID"));
                p.setName(r.getString("NAME"));
                p.setQuota(r.getInt("QUOTA"));
                p.setElectionId(r.getInt("ELECTIONID"));
```

```
                provinces.add(p);
            }
            r.close();
            statement.close();
            ProvinceDM.db.close();

            return provinces;
        } catch (SQLException ex) {
            Logger.getLogger(ProvinceDM.class.getName()).log(Level.SEVERE, null, ex);
            return null;
        }
    }

    public static Province getOneProvince(Integer ID, Integer ElectionID) {
        Province province = null;

        try {
            ProvinceDM.psm = new PostgreSqlManager();
            ProvinceDM.db = psm.getConnection("jdbc:postgresql://localhost:5432/EVoting", " ↩
                postgres", "123456");

            String query = "SELECT * FROM PROVINCE WHERE(ID = ? AND ELECTIONID=?)";
            PreparedStatement statement = db.prepareStatement(query);
            statement.setInt(1, ID);
            statement.setInt(2, ElectionID);
            ResultSet rs = statement.executeQuery();
            if (rs.next()) {
                province = new Province();
                province.setId(rs.getInt("ID"));
                province.setElectionId(rs.getInt("ELECTIONID"));
                province.setName(rs.getString("NAME"));
                province.setQuota(rs.getInt("QUOTA"));
            }

            rs.close();
            statement.close();
            db.close();
        } catch (SQLException e) {
            throw new UnsupportedOperationException(e.getMessage());
        } catch (Exception ex) {
            Logger.getLogger(ProvinceDM.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            return province;
        }
    }

    public static Province getOneProvince(String Name, Integer ElectionID) {
        Province province = null;

        try {
            ProvinceDM.psm = new PostgreSqlManager();
            ProvinceDM.db = psm.getConnection("jdbc:postgresql://localhost:5432/EVoting", " ↩
                postgres", "123456");

            String query = "SELECT * FROM PROVINCE WHERE(NAME = ? AND ELECTIONID=?)";
            PreparedStatement statement = db.prepareStatement(query);
            statement.setString(1, Name);
            statement.setInt(2, ElectionID);
            ResultSet rs = statement.executeQuery();
            if (rs.next()) {
                province = new Province();
                province.setId(rs.getInt("ID"));
```

```
                province.setElectionId(rs.getInt("ELECTIONID"));
                province.setName(rs.getString("NAME"));
                province.setQuota(rs.getInt("QUOTA"));
            }

            rs.close();
            statement.close();
            db.close();
        } catch (SQLException e) {
            throw new UnsupportedOperationException(e.getMessage());
        } catch (Exception ex) {
            Logger.getLogger(ProvinceDM.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            return province;
        }
    }

    /**
     *
     * @return
     */
    public static ResultSet searchProvince() {
        try {
            ProvinceDM.psm = new PostgreSqlManager();
            ProvinceDM.db = psm.getConnection("jdbc:postgresql://localhost:5432/EVoting", " ↩
                postgres", "123456");
            String query = "SELECT * FROM PROVINCE";
            Statement state;
            state = ProvinceDM.db.createStatement();
            state.executeQuery(query);
            ResultSet r = state.getResultSet();

            return r;
        } catch (SQLException ex) {
            Logger.getLogger(ProvinceDM.class.getName()).log(Level.SEVERE, null, ex);
            return null;
        }
    }
}
```

3.2.2.7 ProvinceCollection

This class keeps a list of provinces. It has a constructor, getProvinces, addProvince and deleteProvince methods as described in below.

```
                        /*
                 * To change this template, choose Tools | Templates
                 * and open the template in the editor.
                 */
                package itucs.blg361e.EVoting.Province;

                /**
                 *
                 * @author bahti
                 */
                import java.util.LinkedList;
                import java.util.List;

                public class ProvinceCollection {
```

```
    private List Province  province;

    public ProvinceCollection() {
        this.province = new LinkedList Province();
    }

    public List Province getProvinces() {
        return this.province;
    }

    public void addProvince(Province p) {
        this.province.add(p);
    }

    public void deleteProvince(Province p) {
        this.province.remove(p);
    }
}
```

3.2.2.8 Province Display Page Link

This class makes a link to Province Page.

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package itucs.blg361e.EVoting.Province;

/**
 *
 * @author bahti
 */
import org.apache.wicket.markup.html.link.Link;

public class ProvinceDisplayPageLink extends Link {

    private Province province1;

    public ProvinceDisplayPageLink(String id, Province p) {
        super(id);
        this.province1 = p;
    }

    @Override
    public void onClick() {
        this.setResponsePage(new ProvincePage());
    }
}
```

3.2.3 Login Operations and User

3.2.3.1 User and UserDM Classes

User class contains three attributes as shown below.

```
public class User implements Serializable {
```

```
    private String username;
    private String password;
    private Integer userType;

    public User() {
    }

    public User(String username, String password,
    Integer userType) {
        this.username = username;
        this.password = password;
        this.userType = userType;
    }

}
```

This class has also getter and setter methods, but they were not shown here.

Add, delete, update, and search operations related with user has been implemented in UserDM class. Since its methods are defined static, they can be called like UserDM.methodName() from any part of the project.

Methods in UserDM

- **createTable:** creates the USERS table in database
- **addUser:** adds a user to USERS table
- **deleteUser:** deletes a user from USERS table
- **updateUser:** updates user information
- **getUser:** returns a user from database according to username

Following queries were used in UserDM methods.

- **createTable:** CREATE TABLE USERS (USERNAME VARCHAR(80) PRIMARY KEY, PASSWORD VARCHAR(80), USERTYPE INTEGER)
- **addUser:** INSERT INTO USERS (USERNAME, PASSWORD, USERTYPE) VALUES(?, ?, ?)
- **deleteUser:** DELETE FROM USERS WHERE (USERNAME = ?)
- **updateUser:** UPDATE USERS SET PASSWORD = ?, USERTYPE = ? WHERE(USERNAME = ?)
- **getUser:** SELECT * FROM USERS WHERE(USERNAME = ?)

3.2.3.2 PasswordGenerator Class

User passwords are generated automatically, while adding a user to database. UUID(Universal Unique IDs) class from java language is used in PasswordGenerator class.

```
public class PasswordGenerator {

    private static UUID password;

    public PasswordGenerator() {
        // generate random unique ids
        password = UUID.randomUUID();
    }

    public static String getPassword() {
```

```
                    new PasswordGenerator();

                    // generates unique ids like
                    // 10d1b913-8985-421c-8c20-00baaa1166c6
                    return String.valueOf(password).split("-")[4];
                }
            }
```

We are getting the last part only, using split function.

3.2.3.3 MySession Class

MySession class is used for login operation. It stores a user in it. It extends Wicket WebSession

```java
public class MySession extends WebSession {

    private User user;

    public MySession(Request request) {
        super(request);
    }

    public static MySession get() {
        return (MySession) Session.get();
    }

    public synchronized void setUser(User user) {
        this.user = user;
    }

    public synchronized User getUser() {
        return user;
    }

    public synchronized boolean isLoggedIn() {
        return (user != null);
    }

    public synchronized boolean isAuthenticated(User user) {
        User result_user = UserDM.getUser(user.getUsername());
        boolean result = false;

        if (result_user != null) {
            result = result_user.getPassword().equals(user.getPassword());
            setUser(result_user);
        }

        return result;
    }
}
```

Figure 3.7: MySession Class

Methods in MySession

- **get:** returns the current session
- **setUser:** set a user for the session
- **getUser:** returns current user of current session
- **isLoggedIn:** returns if there is a logged in user in session
- **isAuthenticated:** returns if requested user's username and password match with values in database

MySession class is used in 'LoginPage.java' as shown below.

```
Button login = new Button("login") {
    @Override
    public void onSubmit() {

        MySession session = MySession.get();
        if (session.isAuthenticated(user)) {
            Integer user_kind = UserDM.getUser(user.getUsername()).getUserType();

            if (user_kind == 0) {
                this.setResponsePage(new AdminPage());
            } else if (user_kind == 1) {
                this.setResponsePage(new PartyAdministratorPage());
            } else if (user_kind == 2) {
                this.setResponsePage(new ElectorPage());
            } else if (user_kind == 3) {
                this.setResponsePage(new CandidatePage());
            } else {
                error("There is no such user type");
            }
        } else {
            error("Invalid username or password");
        }
    }
};
```

Figure 3.8: MySession usage in LoginPage.java

If user is authenticated, type of user is determined, and redirected to related page. If not, an error message appears.

MySession is also used by other classes to get the current user's information in the session.

3.2.4 View Classes

I made general page designs in project. While designing pages, I tried to make usage as simple and easy as possible. Following sections describe these essential pages.

3.2.4.1 Base Page

Base page is the core page of project. Almost all pages inherits from base page. It consists of header, navigation, content, and footer parts.

Header field consists of a title, and dynamic date and clock.

Figure 3.9: Header

Navigation field consists of page links. Some of the links become visible or invisible according to user type and election state.

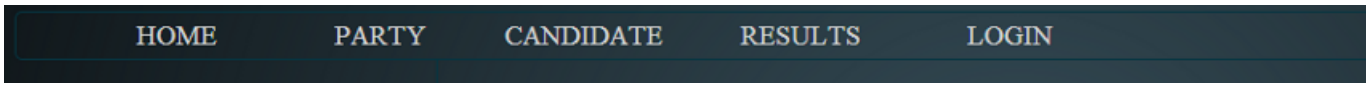


Figure 3.10: Navigation

Content field(between navigation and footer) is filled by other pages which inherits from base page according to needs. It can be used entirely or divided to left and right contents.

Footer field contains a copyleft symbol and annotation along with a link to project repository.

Figure 3.11: Footer

3.2.4.2 Home Page

Homepage is the first place that users will encounter. User can reach other pages or login screen from this page using links in navigation bar.

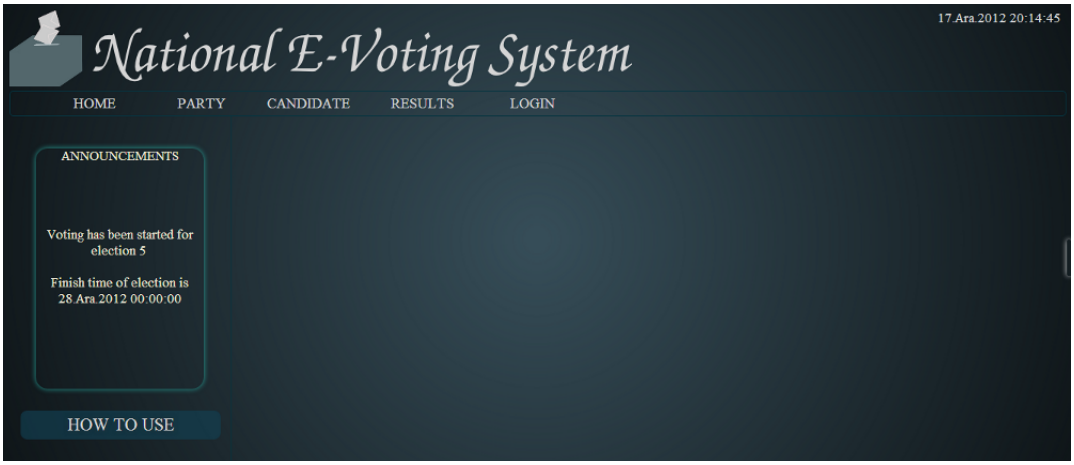


Figure 3.12: Home Page

It inherits from base page, so it contains header, navigation, and footer. It uses content as left and right. In left content, there is an announcements field and a how to use link.

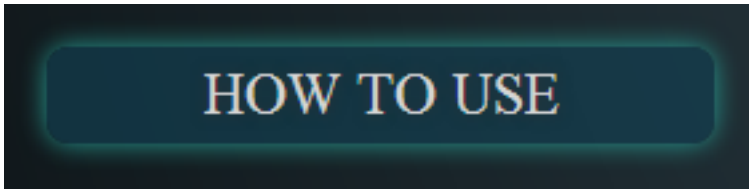


Figure 3.13: How To Use Link

If user clicks on this link, a simple guide for electors appears in right content.

Figure 3.14: How To Use Guide

Announcements field contains sliding information about election such as finish time of active election or start time of next election.

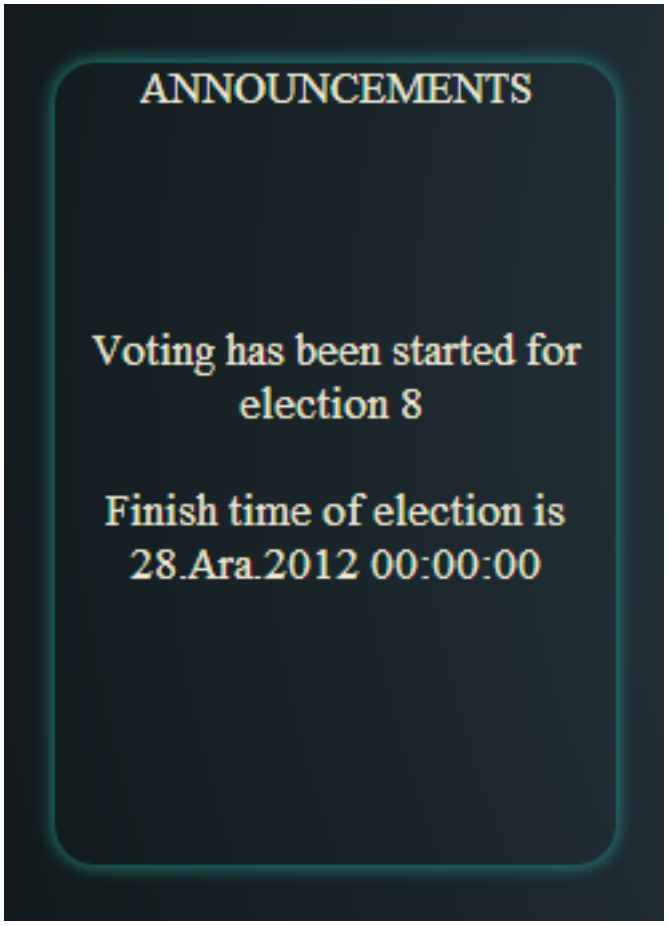


Figure 3.15: Announcements Field

There is a hidden link on right most of homepage to initialize the project with some data. If mouse pointer comes over it, it expands to left.

Figure 3.16: Hidden initialize project link

Figure 3.17: Initialize project link when mouse over position

3.2.4.3 Admin Panel

Admin panel contains add, delete and update links. It inherits from Wicket Panel component.

3.2.4.4 Admin Base Page

Admin base page inherits from base page and contains admin panel in left content.

Add, delete, and update pages inherits from admin base page to look same. Contents of these pages are placed in right content.

Figure 3.18: Admin Base Page

3.2.4.5 Login Page

This page was written for enabling users to login the system. It inherits from base page. It contains a Wicket Form component.

There are username and password fields in login form. Since username is compulsory, the text field related with it is in type RequiredTextField. For taking passwords, PasswordTextField was used. Form and user objects were bound each other using Wicket's PropertyModel model class.

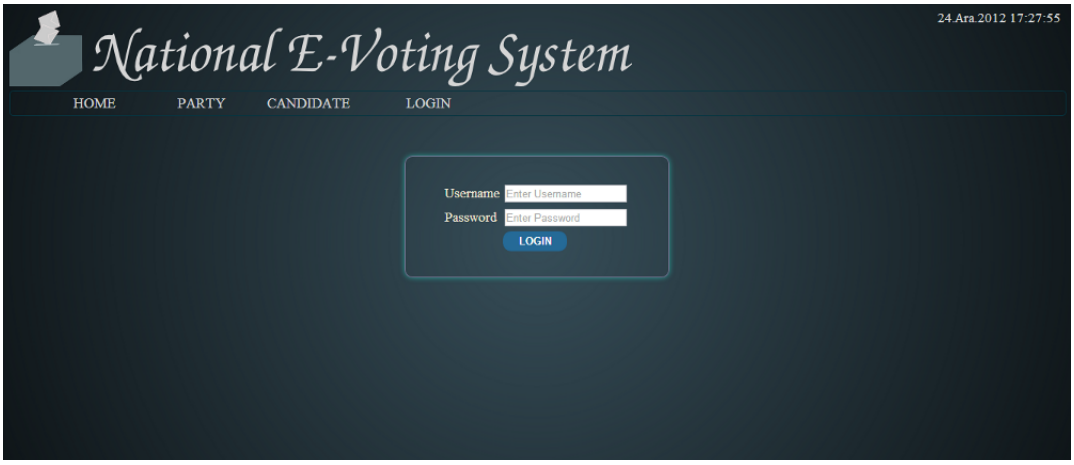


Figure 3.19: Login Page

3.2.5 Party Class

This class which represents the political parties, contains constructor of party class. Besides, this class include getters and setters for attributes of this class. Attributes of this class are:

- private String title,
- private Integer year,
- private String partyleader,
- private String partyPrinciples,
- private Integer electionId.

3.2.6 PartyDM Class

This class includes queries about the database operation of parties.

- With "createTable()", party table is created.
- Parties are added to database with "addParty" method.
- Deleted from database with "deleteParty" method and updated with "updateParty" method.
- Besides, all parties are gotten from database with "getParty()" method.
- By "title" and "electionId", one party is gotten from database with "getOneParty()" method.
- For selecting parties of an election, "getPartiesWithElectionId" is used.

3.2.7 Party Edit Page

In this wicket page, "title", "year", "partyleader" and "partyPrinciples" attributes are gotten from user (admin).

- When clicked on "ADD" button, election id is assigned automatically (*party.setElectionId(ElectionOrganizer.currentOrNextElection.g*
- Party can be added until two weeks before election (*if (!ElectionOrganizer.canAddCandidate())*).
- If there is a party with the same name, this party is not added (*if (PartyDM.getOneParty(party.getTitle(), party.getElectionId()) != null)*) .

3.2.8 Party Delete Page

For deleting a party, ""title" attributes is gotten from user.

- When clicked on "DELETE" button election id is assigned automatically (*party.setElectionId(ElectionOrganizer.currentOrNextElectio*
- End time of election is controlled (*if (ElectionOrganizer.state == State.NOTSTARTED)*).
- The presence of the party is controlled (if (*PartyDM.getOneParty(party.getTitle(), party.getElectionId()) == null)*).
  If the conditions are met, this party is deleted.

3.2.9 Party Update Page

In this wicket page, party is gotten from database by "title", but admin can get a party only next election's party.

- When clicked on "GET" button, the presence of the party is controlled. Besides party can be gotten from database until two weeks before election.
- When clicked on "UPDATE" button, after the necessary controls are made, updating operation is completed.

3.2.10 Party Page Class

In this class, all parties are gotten to "ResultSet". Then parties are added "collection" from "ResultSet". Parties are added to "List<party>" from "collection" . Then all this parties are shown as a link. These links contain party title, party year and party leader.

3.2.11 Party Display Page

Information of party is shown with labels. Candidates of this parties are listed as a link by using "populateItem". If these links is clicked, candidate page is shown.

3.2.12 PartyAdministrator Class

This class which represents the political parties administrator, contains constructor of PartyAdministrator class. Besides, this class include getters and setters for attributes of this class. Attributes of this class are:

- private String userName,
- private String partyTitle,
- private Integer electionId.

This class is implemented serializable.

3.2.13 PartyAdministratorDM Class

This class includes queries about the database operations of party administrators.

- "connect()" method is used for establishing a connection with the database.
- With "createTable()", party administrator table is created.
- Party administrators are added to database with "addPartyAdmin" method, deleted from database with "deletePartyAdmin" method and updated with "updatePartyAdmin" method.
- Besides, all party administrators are gotten from database with "getPartyAdministrator()" method.
- With "userName", one party admin is gotten from database with "getOneAdmin()" method.
- For selecting party administrators of any party, "getPartyAdministratorWithParty()" is used.

3.2.14 Party Administrator Edit Page

In this wicket page, "userName", "partyTitle" attributes are gotten from admin.

- When clicked on "ADD" button, election id is assigned automatically (*party.setElectionId(ElectionOrganizer.currentOrNextElection.g*
- If no record can be found in the user table (*if (UserDM.getUser(user.getUsername()) == null)*)
- and "partyTitle" is in the party table(if (*PartyDM.getOneParty(partyAdministrator.getPartyTitle(), partyAdministrator.getElectionId())* *!= null)*),
- party administrator is added to part administrator and users table.

3.2.15 Party Administrator Delete Page

- For deleting a party administrator, "userName" attributes is gotten from user (*this.add(new RequiredTextField("userName"))*).
- When clicked on "DELETE" button, if party administrator database have an administrator with this "userName",
  - this admin is deleted party administrator table (*deneme.deletePartyAdmin(partyAdministrator)*)
  - and users table (*UserDM.deleteUser(UserDM.getUser(partyAdministrator.getUserName()))*).

3.2.16 Party Administrator Update Page

In this wicket page, party is gotten from database by "userName".

- When clicked on "GET" button, the presence of the party administrator is controlled (*if (PartyAdministratorDM.getOneAdmin(partyAd != null)*).
- When clicked on "UPDATE" button, after making the necessary controls, updating operation is completed.

3.2.17 Party Administrator Page

Party administrator page inherits from base page and contains party administrator panel in left content. Add, delete, and update pages of candidate inherits from admin party administrator page to look same. Contents of these pages are placed in right content.

3.2.18 Candidate Class

This class which represents the candidates of parties, contains constructor of candidate class. Besides, this class include getters and setters for attributes of this class. Attributes of this class are:

- private String TCnumber,
- private String name,
- private String partyTitle,
- private String userName,
- private Boolean isIndependent,
- private Integer electionId.

This class implements serializable.

3.2.19 CandidateDM Class

This class includes queries about the database operations of candidates.

- With "createCandidateTable()",candidate table is created.
- Candidates are added to database with "addCan()" method,
- Deleted from database with "deleteCan()" method and updated with "updateCan()" method.
- Besides, all candidates are gotten from database with "getCandidate()" method.
- Using "TCnumber", one candidate is gotten from database with "getOneCandidate()" method.
- Any province's candidates (independents or dependents) are gotten from database with "getCandidatesForProvince()".
- Also, candidates of any party are gotten from database with "getCandidateWithPartyTitle()" methods.
- Finally, a candidate is gotten by name with "getCandidateWithName()" method.

3.2.20 Candidate Update Page

In this wicket page, candidate is gotten from database with "TCnumber".

- Admin can get only next election's candidate (*candidate.setElectionId(ElectionOrganizer.currentOrNextElection.getElectionID())*).
- When clicked on GET button, the presence of the candidate is controlled (if (*CandidateDM.getOneCandidate(candidate.getTCnumber != null)*).
- Besides, candidate can be gotten from database until two weeks before election (*if (ElectionOrganizer.canAddCandidate())*).
- When clicked on UPDATE button, provinceid is controlled whether it is valid or not. After the necessary controls are made, updating operation is completed.

3.2.21 Database Connection

First of all i implemented *PostgreSqlManager* class to handle basic connection operations. Mentioned Database manager classes use that class to connect database.

```
public class PostgreSqlManager {
    private static Connection connection;

    public PostgreSqlManager()
    {
        connection = null;
    }
    public Connection getConnection()
    {
        try {
            Class.forName("org.postgresql.Driver");
            connection = DriverManager.getConnection("jdbc:postgresql://localhost:5432/ ↩
                testdb", "user","password");
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(PostgreSqlManager.class.getName()).log(Level.SEVERE, null, ex) ↩
                ;
        } catch (SQLException ex) {
            Logger.getLogger(PostgreSqlManager.class.getName()).log(Level.SEVERE, null, ex) ↩
                ;
        }
        return connection;
    }

    public Connection getConnection(String url, String user, String password)
    {
        try {
            Class.forName("org.postgresql.Driver");
            connection = DriverManager.getConnection(url, user, password);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(PostgreSqlManager.class.getName()).log(Level.SEVERE, null, ex) ↩
                ;
        } catch (SQLException ex) {
            Logger.getLogger(PostgreSqlManager.class.getName()).log(Level.SEVERE, null, ex) ↩
                ;
        }
        return connection;
    }

    public boolean doesTableExists(String table) throws Exception
    {
        if(connection == null)
```

```
            throw new Exception("There is no connection.");
        else
        {
            try {
                boolean result = false;
                Statement statement = connection.createStatement();
                String tables = "SELECT * FROM pg_tables WHERE schemaname='public'";
                ResultSet results = statement.executeQuery(tables);
                while(results.next())
                {
                    if(table.equals(results.getString(2)))
                    {
                        result = true;
                        break;
                    }
                }
                results.close();
                statement.close();
                return result;
            } catch (SQLException ex) {
                Logger.getLogger(PostgreSqlManager.class.getName()).log(Level.SEVERE, null, ↩
                     ex);
                throw new Exception(ex.getMessage());
            }
        }
    }
}
```

- *getConnection* function without parameter is used to connect to database using jdbc driver. The parameters that are necessary to connect database are hard codded in function.
- *getConnection* function with parameters is used to connect database by using given parameters.
- *doesTableExist* function checks the database and returns if there is any table named given parameter.

3.2.22 *Election* class

Election class is used to represent a tuple in the "ELECTION" table. There are getter and setter methods for members. *Election* class implements the *Serializable* class to handle serialization operation on Apache server.

3.2.23 *ElectionDM* class

ElectionDM class is used to manage the database operation of *Election* object. The methods of that class are listed below:

- *createTabel* : creates "ELECTION" table if it does not exist, as defined before.
- *addElection* : inserts new election object to database. The function sets the electionId by looking existing elections and the other attributes are taken from outside as parameters.
- *deleteElection* : deletes given election from "ELECTION" table
- *updateElection* : updates given election for given parameters.
- *getElections* : without parameters returns the all elections that are in "ELECTION" table.
- *getElections* : with parameters returns the elections that are proper given parameters.

```
/*
 * 0 for startDateOfElection
 * 1 for lastDateOfElection
 * 2 for lastDateOfAddingCandidate
 */
public static ArrayList<Election> getElections(Timestamp currentDate, int queryForWhat, int ordering)
{
    try {
        ElectionDM.psm = new PostgreSqlManager();
        ArrayList<Election> elections = new ArrayList<Election>();
        ElectionDM.dbConnection = psm.getConnection("jdbc:postgresql://localhost:5432/EVoting","postgres","123456");
        String whichDate = "";
        switch(queryForWhat)
        {
            case 0:
                whichDate = "startDateOfElection";
                break;
            case 1:
                whichDate = "finishDateOfElection";
                break;
            case 2:
                whichDate = "lastDateOfAddingCandidate";
                break;
            default:
                whichDate = "startDateOfElection";
        }

        String order = (ordering == 0) ? "ASC" : "DESC";
        String query = "SELECT * FROM election WHERE(" + whichDate + " > ?) ORDER BY "
                + whichDate + " " + order;
        PreparedStatement statement = ElectionDM.dbConnection.prepareStatement(query);
        statement.setTimestamp(1, currentDate);
        ResultSet rs = statement.executeQuery();
```

Figure 3.20: getElections method

currentDate is datetime of current day, queryForWhat choose the attribute that query is related, ordering determines the result order such as ascending or descending.

Query	Method
CREATE TABLE ELECTION(electionID INTEGER PRIMARY KEY, startDateOfElection TIMESTAMP NOT NULL, finishDateOfElection TIMESTAMP NOT NULL, lastDateOfAddingCandidate TIMESTAMP NOT NULL)	createTable
SELECT electionID from election ORDER BY electionID DESC INSERT INTO ELECTION (electionID, startDateOfElection, finishDateOfElection, lastDateOfAddingCandidate) VALUES(?, ?, ?, ?)	addElection
DELETE FROM ELECTION WHERE (electionID = ?)	deleteElection
UPDATE ELECTION SET startDateOfElection = ?, finishDateOfElection = ?, lastDateOfAddingCandidate = ? WHERE (electionID = ?)	updateElection
SELECT * FROM election	getElections
SELECT * FROM election WHERE(? > ?) ORDER BY ? ASC/DESC	getElections

Figure 3.21: Used query in *ElectionDM* class

3.2.24 Web Interface for Election

ElectionEditorPage.html has been implemented for database operations that are related with election. Wicket related jobs are handled in *ElectionEditorPage.java* . This page has especially two pages with AjaxTabbedPanel AddElection, EditElection that includes delete and update operations. These pages are referenced pages, so their name includes $ sign. *ElectionEditorPage$AddElection.html* includes *AddElectionForm* that handles add operation. *ElectionEditorPage$ElectionEdit.html* includes *DeleteAndUpdateElectionForm* that handles delete and update operations. Both form have datetimepicker and this is created with *datetimepicker_css.js.*

3.2.25 *ElectionOrganizer* class

This class organize the election, controls dates related elections and sets the state of election as NOTSTARTED, STARTED or FINISHED that are enum *State* class object. To handle dynamic time issues there is a label in *BasPage* which every page is inherited from it.

```
electionLabel = new Label("electionLabel", "");
electionLabel.setOutputMarkupId(true);
this.add(electionLabel);
electionLabel.add(new AbstractAjaxTimerBehavior(Duration.seconds(3)) {

    @Override
    protected void onTimer(AjaxRequestTarget target) {
        ElectionOrganizer.setCurrentOrNextElection();
    }
});
```

Figure 3.22: A label with AbstractAjaxTimerBehavior for dynamic time model.

The methods of that class are listed below:

- *setCurrentOrNextElection* : Determines the current election by searching the database according to current time.
- *setState* : Compares current time and date fields of current election and sets state variable as proper *State* value.
- *canVote* : Checks the state is STARTED.
- *canAddCandidate* : Checks the state is NOTSTARTED and election is going to start at least two weeks later.

```
public class ElectionOrganizer {
    public enum State
    {
        NOTSTARTED, STARTED, FINISHED
    }

    public static Election currentOrNextElection = null;
    public static State state = null;

    public static void setCurrentOrNextElection()
    {
        Timestamp current = new Timestamp(Calendar.getInstance().getTimeInMillis());
        if(currentOrNextElection == null)
        {
            ArrayList$lt;Election$gt; elections = ElectionDM.getElections(current, 1, 0);
            if(elections != null)
            {
                if(!elections.isEmpty())
                {
                    currentOrNextElection = elections.get(0);
                    setState();
                }
            }
        }
        else
        {
            //1 month
            if(current.getTime() $gt; currentOrNextElection.getFinishDateOfElection(). ↩
                getTime() + 2592e6)
            {
                currentOrNextElection = null;
                setCurrentOrNextElection();
                setState();
            }
        }
```

```
    }

    public static void setState()
    {
        Timestamp current = new Timestamp(Calendar.getInstance().getTimeInMillis());
        if(currentOrNextElection.getStartDateOfElection().getTime() $gt; current.getTime())
            state = State.NOTSTARTED;
        else
        {
            if(currentOrNextElection.getFinishDateOfElection().getTime() $lt; current. ↩
                getTime())
                state = State.FINISHED;
            else
                state = State.STARTED;
        }

    }

    public static boolean canVote()
    {
        boolean result = false;
        if(state == State.STARTED)
            result = true;
        return result;
    }

    public static boolean canAddCandidate()
    {
        boolean result = false;
        Timestamp current = new Timestamp(Calendar.getInstance().getTimeInMillis());
        if(currentOrNextElection != null)
            if(currentOrNextElection.getLastDateOfAddingCandidate().getTime() $gt; current. ↩
                getTime())
                result = true;
        return result;
    }
}
```

3.2.26 *Vote* class

Vote class is used to represent a tuple in the "VOTE" table. There are getter and setter methods for members. *Vote* class implements the *Serializable* class to handle serialization operation on Apache server.

3.2.27 *VoteDM* class

VoteDM class is used to manage the database operation of *Vote* object. The methods of that class are listed below:

- *createTabel* : creates "VOTE" table if it does not exist, as defined before.
- *addVote* : inserts new vote object to database.
- *deleteVote* : deletes given vote from "VOTE" table
- *updateVote* : updates given vote for given parameters.
- *getAllVotesForElection* : returns all votes for election or given province and election.
- *getVotesForAProvince* : returns all votes as party title and vote count for election or given province and election.
- *getAllVotesForElection* : returns all votes for election or given province and election.

- *getVotesOfOverTresholdParties* : returns all votes which count ratio is greater than given treshold as party title and vote count for given province and election.

```
public static Map$lt;String, Integer$gt; getVotesOfOverTresholdParties(int electionId,  ↩
    double treshold)
    {
        Map$lt;String, Integer$gt; votes = new HashMap$lt;String, Integer$gt;();
        HashMap$lt;String, Integer$gt; allVotes = VoteDM.getVotesForAProvince( ↩
            ElectionOrganizer.currentOrNextElection.getElectionID(),0);

        double total = 0;

        for(String title : allVotes.keySet())
            total += allVotes.get(title).doubleValue();
        for(String title : allVotes.keySet())
        {
            if(allVotes.get(title).doubleValue() / total $gt;= treshold)
                votes.put(title, allVotes.get(title));
        }

        return votes;
    }
```

- *getCandidateDistributionForAProvince* : returns congressman distribution by calculating the votes.

  Pseudocode of calculating congressman distribution for a province.

  - Find the parties that exceeded the election treshold
  - Get votes of that parties and independent candidates who join the election from that province
  - Divide votes 1 to quota of province for each party and independent candidate
  - Sort votes in descending order
  - Assign a congressman for first number as quota from sorted list

```
public static Map$lt;String, Integer$gt; getCandidateDistributionForAProvince(int  ↩
    electionId, int provinceId)
    {
        Map$lt;String, Integer$gt; distribution = new HashMap$lt;String, Integer$gt;();
        Map$lt;String, Double$gt; mapForCalculation = new HashMap$lt;String, Double$gt;();
        Map$lt;String, Integer$gt; votesForProvince = VoteDM.getVotesForAProvince( ↩
            ElectionOrganizer.currentOrNextElection.getElectionID(),provinceId);
        Map$lt;String, Integer$gt; overTresholdParties = VoteDM. ↩
            getVotesOfOverTresholdParties(electionId, 0.1);
        int quota = ProvinceDM.getOneProvince(provinceId, electionId).getQuota();


        for(String title : votesForProvince.keySet())
        {
            if(!title.equals(""))
            {
                if(CandidateDM.getCandidateWithName(title, electionId))
                    mapForCalculation.put(title, votesForProvince.get(title).doubleValue() ↩
                        );
                else
                {
                    if(overTresholdParties.containsKey(title))
                    {
                        for(int i = 1; i $lt;= quota; i++)
                            mapForCalculation.put(title + "-" + Integer.toString(i),  ↩
                                votesForProvince.get(title).doubleValue() / i);//solves  ↩
                                duplicate keys problem
                    }
```

```
                }
            }
        }

        mapForCalculation = VoteDM.sortByComparator(mapForCalculation);

        int i = 1;

        for(String title : mapForCalculation.keySet())
        {
            if(i $gt; quota)
                break;
            else
            {
                String originalTitle = title.split("-")[0];
                if(!distribution.containsKey(originalTitle))
                    distribution.put(originalTitle, 1);
                else
                    distribution.put(originalTitle, distribution.get(originalTitle) + 1);
                i++;
            }
        }

        return distribution;
    }
```

- *sortByComparator* : sorts given map in descending order

```
public static Map sortByComparator(Map$lt;String, Double$gt; unsortMap) {

        List list = new LinkedList(unsortMap.entrySet());

        // sort list based on comparator
        Collections.sort(list, new Comparator() {

            public int compare(Object o1, Object o2) {
                return -(((Comparable) ((Map.Entry) (o1)).getValue()).compareTo(((Map. ↩
                    Entry) (o2)).getValue()));
            }
        });

        Map sortedMap = new LinkedHashMap();
        for (Iterator it = list.iterator(); it.hasNext();) {
            Map.Entry entry = (Map.Entry) it.next();
            sortedMap.put(entry.getKey(), entry.getValue());
        }
        return sortedMap;
    }
```

Query	Method
CREATE TABLE VOTE(voteId SERIAL PRIMARY KEY, electionId INTEGER NOT NULL, partyTitle VARCHAR(80), provinceId INTEGER NOT NULL, voteTime TIMESTAMP NOT NULL) startDateOfElection TIMESTAMP NOT NULL, finishDateOfElection TIMESTAMP NOT NULL, lastDateOfAddingCandidate TIMESTAMP NOT NULL)	createTable
INSERT INTO VOTE (partyTitle, electionId, provinceId, voteTime) VALUES(?, ?, ?, ?)	addVote
DELETE FROM VOTE WHERE (voteId = ?)	deleteVote
UPDATE VOTE SET partyTitle = ?, electionId = ?, provinceId = ?, voteTime = ? WHERE (electionID = ?)	updateVote
SELECT * FROM VOTE WHERE (electionId = ?) SELECT * FROM VOTE WHERE (electionId = ? AND provinceId = ?)	getAllVotesForElection
SELECT partytitle, COUNT(*) FROM VOTE WHERE (electionId = ? AND provinceId = ?) GROUP BY partytitle SELECT partytitle, COUNT(*) FROM VOTE WHERE (electionId = ?) GROUP BY partytitle	getVotesForAProvince

Figure 3.23: Used query in *VoteDM* class

3.2.28 Web Interface for Vote

VotePage.html has been implemented for voting and *VotePage.java* has been implemented to handle business logic for voting. Parties and independent candidates that who join the election from election area of elector are listed in a RadioChoice component. There is a button for voting. It is prevented that elector can vote repeatedly.

Obtaining elector from session.

```
user = MySession.get().getUser();
        elector = ElectorDM.getOneElector(user.getUsername());
```

Creating participants list

```
public ArrayList$lt;String$gt; createParticipantList()
    {
        ArrayList$lt;String$gt; participantList = new ArrayList$lt;String$gt;();
        for(Party party : PartyDM.getPartiesWithElectionId(ElectionOrganizer. ↩
            currentOrNextElection.getElectionID()))
            participantList.add(party.getTitle());
        for(Candidate candidate : CandidateDM.getCandidatesForProvince(elector. ↩
            getProvinceID(),
    ElectionOrganizer.currentOrNextElection.getElectionID(), 1))//independents
            participantList.add(candidate.getName());
        return participantList;
    }
```

Controlling repeated voting

```
if(elector.getHasVoted())
                    error("You have already voted!");
                else
                {
                    if(ElectionOrganizer.canVote())
                    {
                        elector.setHasVoted(true);
                        ElectorDM.updateElector(elector);
```

```
                    Vote vote = new Vote();
                    vote.setElectionId(ElectionOrganizer.currentOrNextElection. ↩
                        getElectionID());
                    vote.setVoteTime(new Timestamp(Calendar.getInstance(). ↩
                        getTimeInMillis()));
                    if(selected != null)
                        vote.setPartyTitle(selected);
                    else
                        vote.setPartyTitle("");
                    vote.setProvinceId(elector.getProvinceID());
                    VoteDM.addVote(vote);
                    info("You have voted successfully");
                }
                else
                {
                    error("Voting finished. You can not vote.");
                }
            }
```

3.2.29 Showing Results

In project, there are two different results page. One of them is for showing overall results, *ResultsPage.html*. In that page there is a sliding message bar which shows the general vote ratio with marquee tag. There is a label that shows the congressman distribution. Dynamic components are created in *ResultsPage.java*. Wicket has good facility to create dynamic html components as string with *setEscapeModelStrings*

```
    this.add(new Label("slidingResult", getResultMessage()).setEscapeModelStrings(false));
        this.add(new Label("candidateDistribution", getDistributionMessage()). ↩
            setEscapeModelStrings(false));
```

There is a Turkey map with clickable areas. To insert a map, map and area tags are used and coords attributes determines the area for that related province.

```
<div id="mapOfTurkey">
    <img wicket:id="TurkeyMap" src="images/Turkey.png" width="1024" height="500" usemap="#Map"/>
    <map name="Map" id="Map">
        <area wicket:id="Edirne" title="Edirne" shape="poly" coords="89,39,86,53,82,64,84,67,81,74,8
        <area wicket:id="Konya" title="Konya" shape="poly" coords="307,319,295,293,312,283,317,256,3
        <area wicket:id="Antalya" title="Antalya" shape="poly" coords="198,410,213,378,231,368,291,3
```

Figure 3.24: map, area and coords elements in html

The second results page is for showing results for a province, *ResultsForAProvince.html*. In that page there are two sliding bar. One of them shows the vote ratio for that province. The other one shows the congressman counts for that province. There is also a 3d chart for showing vote ratio. The chart is a html list and using pure css it looks like a chart. The list must be created before page loads. And the css file must be changed according the vote ratio. To handle this issues javascript is used in wicket. *ResultsForAProvince.java* is used to carry out to all mentioned jobs.

```
public String AddLiElement(int order, String partyName, double votePercentage) {
    String partyId;

    switch (order) {
        case 0:
            partyId = "party1";
            break;
        case 1:
            partyId = "party2";
            break;
        case 2:
            partyId = "party3";
            break;
        case 3:
            partyId = "party4";
            break;
        default:
            partyId = "other";
    }
    String structure = "";
    structure = "<li id='" + partyId + "'>\n"
            + "<div class='top'></div>\n"
            + "<div class='bottom'>\n"
            + "<div class='infobox'>\n"
            + "<h3>" + partyName + "</h3>\n"
            + "<p>%" + df.format(votePercentage * 100) + "</p>\n"
            + "</div>\n"
            + "</div>\n"
            + "</li>\n";
    return structure;
}
```

Figure 3.25: Creating list in html

"onLoad" AjaxEventBehaviour handles the adding html element before the page is loaded and AjaxCallListener carries out the js codes.

```
this.add(new AjaxEventBehavior("onLoad") {

            @Override
            protected void onEvent(AjaxRequestTarget target) {
            }

            @Override
            protected void updateAjaxAttributes(AjaxRequestAttributes attributes) {
                super.updateAjaxAttributes(attributes);
                AjaxCallListener myAjaxCallListener = new AjaxCallListener() {

                    @Override
                    public CharSequence getBeforeHandler(Component component) {
                        String js = "";
                        int i = 0;
                        for (String partyTitle : votes.keySet()) {
                            if (!partyTitle.equals("")) {
                                if (i $lt; 4) {
                                    switch (i) {
                                        case 0:
                                            js += "var rule = document.styleSheets[0]. ↩
                                                cssRules[13];"
                                                    + "rule.style.cssText = 'z-index:98;  ↩
                                                        height:100px;';"
```

```
                                + "rule = document.styleSheets[0]. ↩
                                    cssRules[14];"
                                + "rule.style.cssText = 'z-index:999;  ↩
                                    background-color:#1f81ac; margin- ↩
                                    left:"
                                + Double.toString(Math.floor((votes.get ↩
                                    (partyTitle).doubleValue() / total)  ↩
                                    * (460 - 10) + 10)) + "px;';"
                                + "rule = document.styleSheets[0]. ↩
                                    cssRules[15];"
                                + "rule.style.cssText = 'z-index:998;  ↩
                                    background-color:#1a6c90; width:"
                                + Double.toString(Math.floor((votes.get ↩
                                    (partyTitle).doubleValue() / total)  ↩
                                    * (460 - 10) + 50)) + "px;"
                                + "background:-moz-linear-gradient(-90 ↩
                                    deg, #1a6c90, #14506b);"
                                + "background:-webkit-gradient(linear,  ↩
                                    0 top, 0 bottom, from(#1a6c90), to ↩
                                    (#14506b));';";
                    i++;
                    break;
                case 1:
                    js += "rule = document.styleSheets[0].cssRules ↩
                        [17];"
                                + "rule.style.cssText = 'z-index:96;  ↩
                                    height:100px;';"
                                + "rule = document.styleSheets[0]. ↩
                                    cssRules[18];"
                                + "rule.style.cssText = 'z-index:997;  ↩
                                    background-color:#bc003c; margin- ↩
                                    left:"
                                + Double.toString(Math.floor((votes.get ↩
                                    (partyTitle).doubleValue() / total)  ↩
                                    * (460 - 10) + 10)) + "px;';"
                                + "rule = document.styleSheets[0]. ↩
                                    cssRules[19];"
                                + "rule.style.cssText = 'z-index:996;  ↩
                                    background-color:#9d0032; width:"
                                + Double.toString(Math.floor((votes.get ↩
                                    (partyTitle).doubleValue() / total)  ↩
                                    * (460 - 10) + 50)) + "px;"
                                + "background:-moz-linear-gradient(-90 ↩
                                    deg, #9d0032, #7a0027);"
                                + "background:-webkit-gradient(linear,  ↩
                                    0 top, 0 bottom, from(#9d0032), to ↩
                                    (#7a0027));';";
                    i++;
                    break;
                case 2:
                    js += "rule = document.styleSheets[0].cssRules ↩
                        [21];"
                                + "rule.style.cssText = 'z-index:94;  ↩
                                    height:100px;';"
                                + "rule = document.styleSheets[0]. ↩
                                    cssRules[22];"
                                + "rule.style.cssText = 'z-index:995;  ↩
                                    background-color:#d98f23; margin- ↩
                                    left:"
                                + Double.toString(Math.floor((votes.get ↩
                                    (partyTitle).doubleValue() / total)  ↩
                                    * (460 - 10) + 10)) + "px;';"
```

```
                                        + "rule = document.styleSheets[0]. ↩
                                            cssRules[23];"
                                        + "rule.style.cssText = 'z-index:994;  ↩
                                            background-color:#9d0032; width:"
                                        + Double.toString(Math.floor((votes.get ↩
                                            (partyTitle).doubleValue() / total)  ↩
                                            * (460 - 10) + 50)) + "px;"
                                        + "background:-moz-linear-gradient(-90 ↩
                                            deg, #b6781e, #916018);"
                                        + "background:-webkit-gradient(linear,  ↩
                                            0 top, 0 bottom, from(#b6781e), to ↩
                                            (#916018));';";
                                    i++;
                                    break;
                                case 3:
                                    js += "rule = document.styleSheets[0].cssRules ↩
                                        [25];"
                                            + "rule.style.cssText = 'z-index:92;  ↩
                                                height:100px;';"
                                            + "rule = document.styleSheets[0]. ↩
                                                cssRules[26];"
                                            + "rule.style.cssText = 'z-index:993;  ↩
                                                background-color:#7da864; margin- ↩
                                                left:"
                                            + Double.toString(Math.floor((votes.get ↩
                                                (partyTitle).doubleValue() / total)  ↩
                                                * (460 - 10) + 10)) + "px;';"
                                            + "rule = document.styleSheets[0]. ↩
                                                cssRules[27];"
                                            + "rule.style.cssText = 'z-index:992;  ↩
                                                background-color:#9d0032; width:"
                                            + Double.toString(Math.floor((votes.get ↩
                                                (partyTitle).doubleValue() / total)  ↩
                                                * (460 - 10) + 50)) + "px;"
                                            + "background:-moz-linear-gradient(-90 ↩
                                                deg, #698d54, #506b40);"
                                            + "background:-webkit-gradient(linear,  ↩
                                                0 top, 0 bottom, from(#698d54), to ↩
                                                (#506b40));';";
                                    i++;
                                    break;
                                default:
                                    break;
                            }
                        } else {
                            break;
                        }
                    }
                }

                js += "rule = document.styleSheets[0].cssRules[29];"
                        + "rule.style.cssText = 'z-index:90; height:100px;';"
                        + "rule = document.styleSheets[0].cssRules[30];"
                        + "rule.style.cssText = 'z-index:991; background-color:#3 ↩
                            f1150; margin-left:"
                        + df.format(Math.floor((totalOther / total) * (460 - 10) +  ↩
                            10)) + "px;';"
                        + "rule = document.styleSheets[0].cssRules[31];"
                        + "rule.style.cssText = 'z-index:990; background-color:#9 ↩
                            d0032; width:"
                        + df.format(Math.floor((totalOther / total) * (460 - 10) +  ↩
                            50)) + "px;"
```

```
                                + "background:-moz-linear-gradient(-90deg, #340e43, #1a0721 ↩
                                    );"
                                + "background:-webkit-gradient(linear, 0 top, 0 bottom,  ↩
                                    from(#340e43), to(#1a0721));';";

                        return js;
                    }
                };
                attributes.getAjaxCallListeners().add(myAjaxCallListener);
            }
        });
```