

Detailed Design Report For Online National Election Voting

Group: iTeam4

- Emilbek Joldoshev 1592476
- Hassan Salahe Matar 1591114
- Mehmet Barış Özkan 1560747
- Hüseyin Lutin 1560408

27/02/2011

1. INTRODUCTION	5
1.1. Problem Definition	5
1.2. Purpose	5
1.3. Scope	6
1.4. Overview	6
1.5. Definitions and Abbreviations	6
1.6. References	7
2. SYSTEM OVERVIEW	8
3. DESIGN CONSIDERATIONS	12
3.1. Design Assumptions, Dependencies and Constraints	12
3.2. Design Goals and Guidelines	12
4. DATA DESIGN	14
4.1. ER Design	14
4.2. Data Schemas	15
4.2.1. City Table	15
4.2.2. District Table	15
4.2.3. Town Table	16
4.2.4. Village Table	16
4.2.5. Voter Table	17
4.2.6. Election Table	17
4.2.7. ElectionType Table	18
4.2.8. BallotBox Table	18
4.2.9. User Table	19
4.2.10. UserType Table	19
4.2.11. PoliticalParty Table	19
4.2.12. Candidate Table	20
4.2.13. CandidateType Table	20
4.2.14. CollectedVote Table	20
4.2.15. Question Table	21
4.2.16. Answer Table	21
4.3. Data Dictionary	22

4.3.1.	Data Classes	22
4.3.2.	User Controller Classes	25
4.3.3.	Data Controller Classes	27
5.	SYSTEM ARCHITECTURE	32
5.1.	Architectural Design	32
5.2.	Description of Components ^[7]	34
5.2.1.	Graphical User Interface	35
5.2.1.1.	Processing Narrative for GUI	35
5.2.1.2.	GUI Interface Description	35
5.2.1.3.	GUI Processing Detail	35
5.2.1.4.	Dynamic Behavior of GUI	36
5.2.2.	Data Storage	36
5.2.2.1.	Processing Narrative for Data Storage	36
5.2.2.2.	Data Storage Interface Description	36
5.2.2.3.	Data Storage Processing Detail	37
5.2.2.4.	Dynamic Behavior of Data Storage	37
5.2.3.	ServerAdministrator(ServerAdmin)	38
5.2.3.1.	Processing Narrative for ServerAdministrator	38
5.2.3.2.	ServerAdministrator Interface Description	38
5.2.3.3.	ServerAdministrator Processing Detail	38
5.2.3.4.	Dynamic Behavior of Server Admin	39
5.2.4.	Authentication	39
5.2.4.1.	Processing Narrative for Authentication	39
5.2.4.2.	Authentication Interface Description	39
5.2.4.3.	Authentication Processing Detail	40
5.2.4.4.	Dynamic Behavior Authentication	40
5.2.5.	Back End Applications	40
5.2.5.1.	Processing Narrative for Back End Applications	40
5.2.5.2.	Back End Applications Interface Description	41
5.2.5.3.	Back End Applications Processing Detail	41
5.2.5.4.	Dynamic Behavior of Back End Application	41
5.2.6.	Data Retrieval	41
5.2.6.1.	Processing Narrative for Data Retrieval	41
5.2.6.2.	Data Retrieval Interface Description	42
5.2.6.3.	Data Retrieval Processing Detail	42
5.2.6.4.	Dynamic Behavior of Data Retrieval	42
5.3.	Design Rationale	43
6.	USER INTERFACE DESIGN	44

6.1.	Overview of User Interface	44
6.2.	Screen Images	44
6.2.1.	Login	44
6.2.2.	User Registration	45
6.2.3.	Login For Vote	46
6.2.4.	Voting	46
6.3.	Screen Objects and Actions	47
7.	DETAILED DESIGN	48
7.1.	Model Package	49
7.2.	Controller Package	51
7.2.1.	User Sub-Package	51
7.2.2.	Database Sub-Package	52
8.	LIBRARIES AND TOOLS	54
9.	TIME PLANNING (GANTT CHART)	55
9.1.	Term1 Gantt Chart	55
9.2.	Term2 Gantt Chart	56
10.	CONCLUSION	57

1. Introduction

This document describes the initial design strategies and structural properties of the Online National Election Voting System which will be developed by iTeam4. It explains the data and interface designs of the project with system architecture in order to help the developers for better design.

1.1. Problem Definition

We are living in a democratic country and voting is one of the fundamental duties of the public. In our country, manual voting system has been deployed for many years. However, manual voting process has caused some difficulties for voting process and also it has some disadvantages for the public. We can list some of these problems as follows. [1]

- Especially there have been cases of threatening in Eastern part of Turkey at polling stations and people are faced with problems during voting.
- Sometimes people may not be in village/county registration and because of that reason they don't fulfill their voting duties.
- Lots of time and problems are occurring on vote counting process since this activity is done manually.
- Due to manual voting process there is lots of paper waste during election times.
- Voter usually doesn't know too much detail about the candidates in their election region.

With the growth and expansion in technology new ways were sought to handle the electoral process such as electronic voting. Electronic voting is the process of use of computers or other electronic devices to cast votes in an election.

So in order to overcome those problems there is a need for a contemporary electronic voting system in addition to manual voting. By design of such a system people can use their votes in any selection field condition to be registered to the system before. Also by using the system voters can learn details about the candidates and they will be interacting with each other before the Election Day. This system will also facilitate the vote counting processes and produce more accurate results and within a short time thanks to the computer technology. Because of these reasons such an electronic voting system contributes to the development of the country's democracy too much.

1.2. Purpose

The purpose of the document is to make the data design and system architecture of the Online National Election Voting System easy to comprehend. It also serves the purpose of making the functionality clear to system designers.

1.3. Scope

This initial design document applies to the initial version (release 1.0) of the “Online National Election Voting System” software package. It describes the database tables, entity relations between objects and architectural structure of the system as noted in SRS document. The main aim of the system is to provide a set of protocols that allow voters to cast ballots while a group of authorities collect votes and output final results.

1.4. Overview

The remainder of this document identifies the system overview, design considerations, data design with class and table structures, system architecture with components and user interface designs. Apart from these main parts, it also states the planning strategies of the project with Gantt diagrams and describes the tools that will be used during implementation.

1.5. Definitions and Abbreviations

The following table(**Table 1**) is a list of terms, acronyms and abbreviations used by the Online National Election Voting System software package and related documentation.

<u>ABREVIATIONS</u>	<u>DEFINITIONS</u>
ONEV	Online National Election Voting
EC	Election Candidate
ECA	Election Commission Authority
ESS	Election Station Supervisor
VIN	Voter Identity Number
DB	Database
TCK	TC Kimlik No
VIC	Voter Identity Card
YSK	Yüksek Seçim Kurulu

Table 1: A table of abbreviations, terms and acronyms.

For the simplicity of documentation throughout the paper we have used masculinity for all genders.

1.6. References

- [1] <http://www.yazilimakademisi.org/2011/detailproject.php?id=25>
- [2] *SRS report for ONEVS*, iTeam4, 2010, www.ceng.metu.edu.tr/~e1591114/SRS
- [3] http://www.w3schools.com/html/html_forms.asp
- [4] <http://experiencezen.com/wp-content/uploads/2007/04/adaptive-path-ajax-a-new-approach-to-web-applications1.pdf>
- [5] <http://www.redbooks.ibm.com/redbooks/pdfs/sg246316.pdf>
- [6] Aneesha Bakharia, (2001), *Java Servlet Pages*, Prima Tech.
- [7] Simon Bernett, Steve McRobb, Ray Farmer, (1996), *Object Oriented System Analysis and Design Using UML*

2. System Overview

There are different types of electronic voting systems such as Punch Card Voting System, Telephone Voting and Online Voting which are being used globally at the current period. Due to the impact of the internet the system will be based on online voting type.

Online voting is a form of voting in which the individuals are able to cast their votes through a web interface. Through the use of online voting, the voter navigates to the designated election site using a web browser on an ordinary PC. The voter is then permitted to select their chosen candidate and then cast the votes which would then be sent to the election server for processing. There three main types of online voting as stated above:

Kiosk Internet Voting: Voting from computers in kiosks set up by voting authority in locations such as post offices and shopping malls.

Poll Site Internet Voting: Voting from designated polling sites to cast their votes by using web interface.

Remote Internet Voting: Voting from any from any location through the use of a computer connected to the internet. Remote voting is typically carried out at the voter's home or work place.

Due to political conditions of our country the ONEV system will be designed as two main parts namely Normal Interactive Mode and Election Mode and the voting process will be executed only at polling stations.

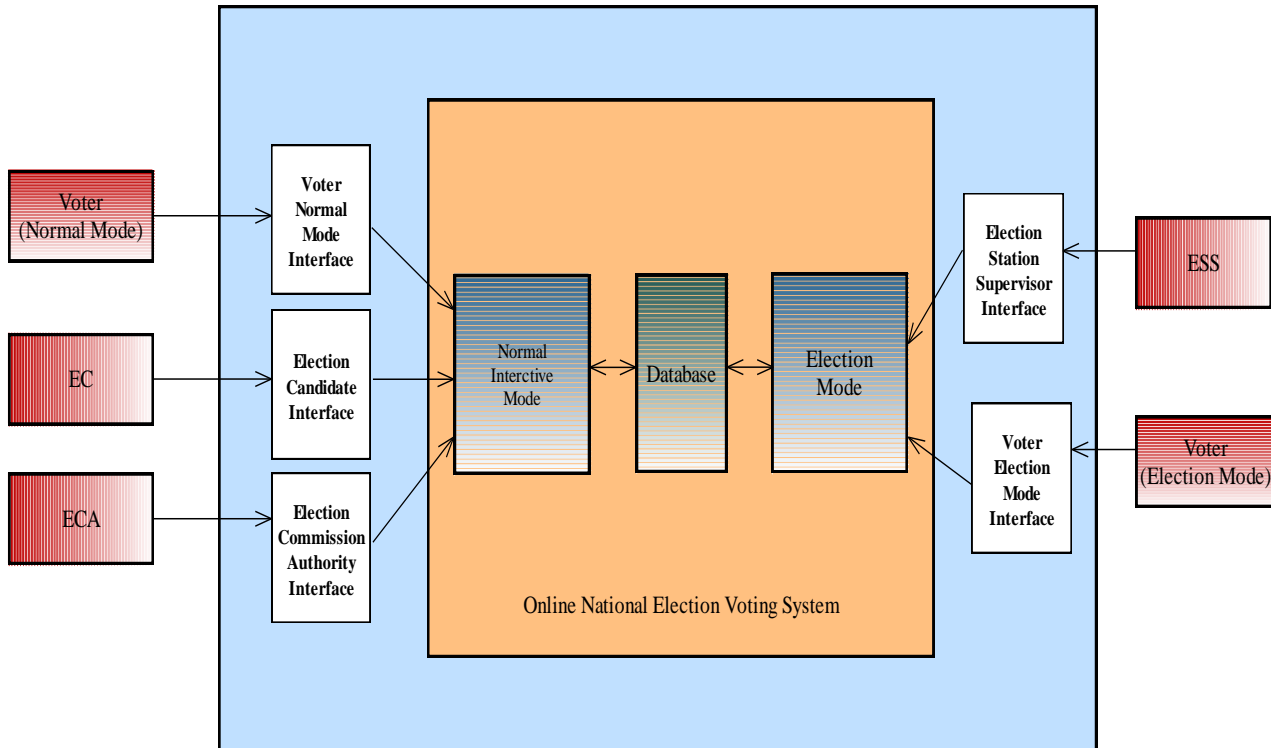


Figure 1: Block diagram showing interaction between users and the system

As shown in the **figure 1** above, Normal Interactive mode will be used by Voters, ECs, and ECAs for the pre-election and ordinary activities. For every stakeholder there will be a web interface that he can use the system functionalities that are described in the SRS report.

In Normal Interactive Mode,

Voters will be able to register to system, see the details of the ECs, ask questions to ECs about their election campaigns and view the past years' election results, as shown in figure 2 below.

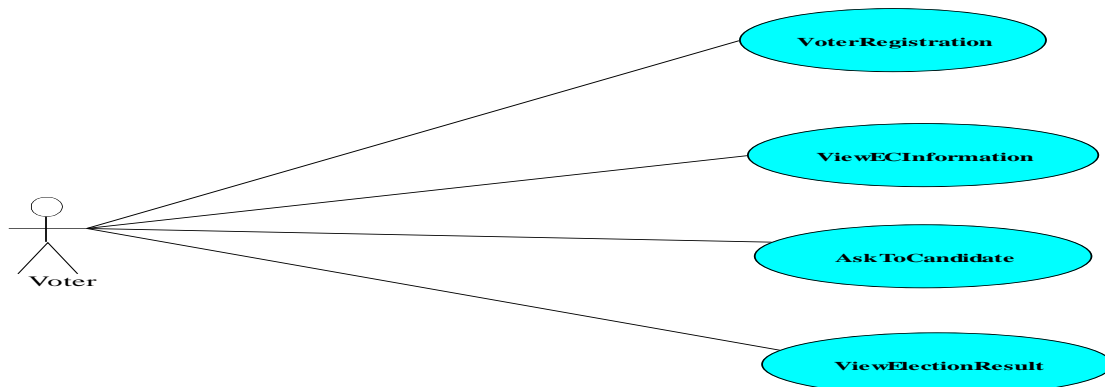


Figure 2: Use Case Diagram for Voter interactions with the system functions

ECs will be able to update their accounts, edit their CVs, add promises about their election campaigns and answer the questions from the voters. The summary of the their interaction with the system can be shown in the figure 3 below.

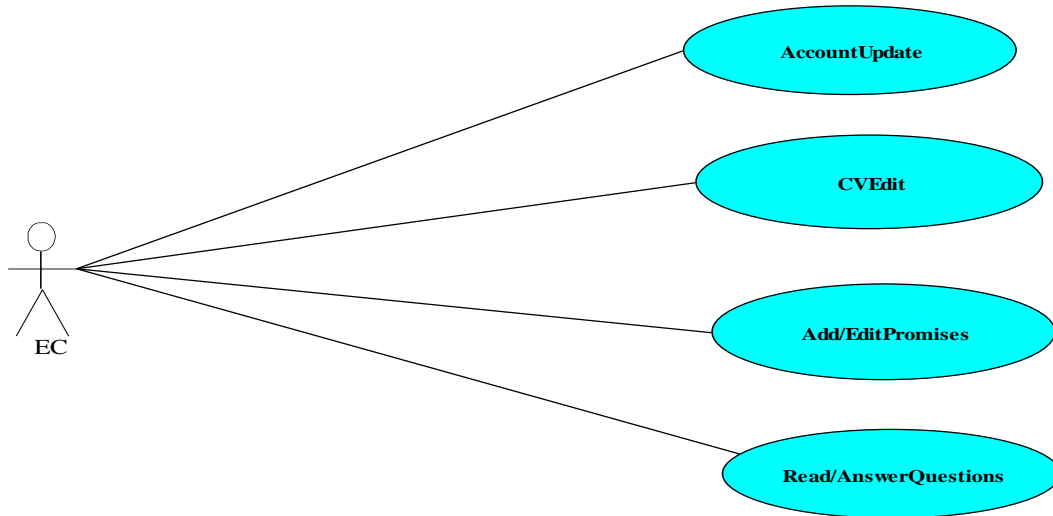


Figure 3: Use Case Diagram for EC interactions with the system functions

ECA s will be able to approve the applications from the voters, update current voters and open candidate account as summarized in the figure 4 below.

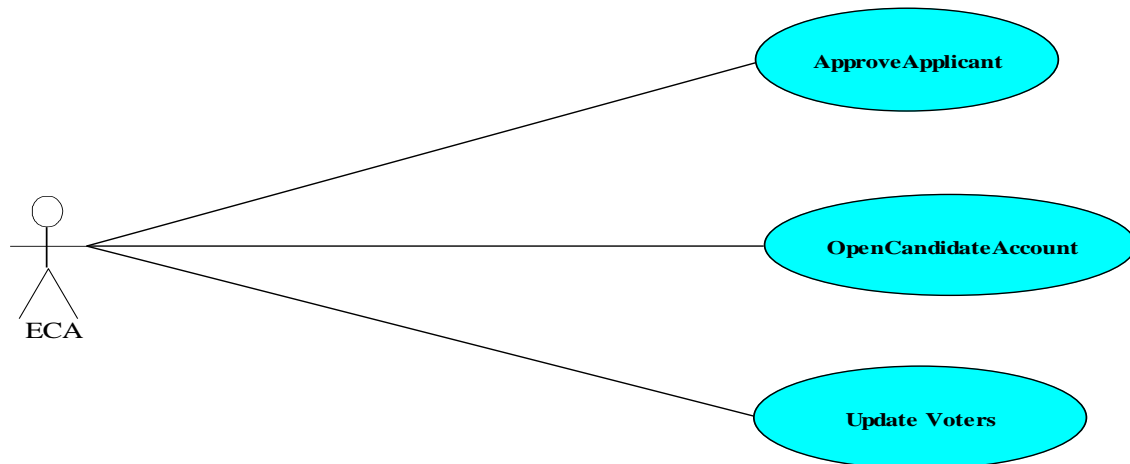


Figure 4: Use Case Diagram for ECA interactions with the system functions

In Election Mode,

The main users of the system are ESSs and Voters. Voters will cast their votes at polling stations with their user id's and passwords. By using the Election Mode, the ESSs will be able to open the system, enter the offline votes to the system and generate hash password-as shown in figure 5- that will be used by voters during the voting process [2].

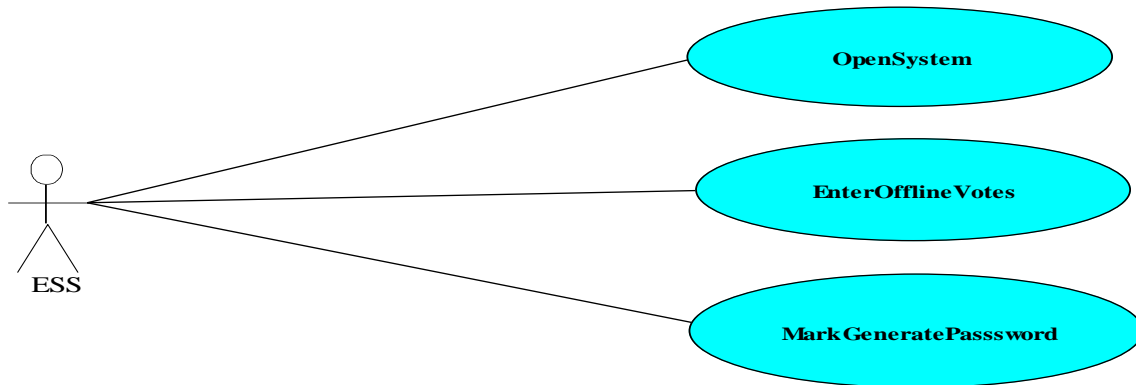


Figure 5: Use Case Diagram for ESS interacting with the system functions

Such a system will provide more contemporary election activities not only for voters but also for election candidates. It will provide the voters to cast their vote from any polling station in case he is not in his election region. Also it provides candidates to conduct their election campaigns through web environment and describe themselves to the voters more clearly.

3. Design Considerations

3.1. Design Assumptions, Dependencies and Constraints

In Turkey, people cast their votes nearly in 170,000 ballots from 81 different cities. Due to this fact the system must work on those ballots at the same time. Since the system divided into two parts, time constraints are different for these parts. In Normal Interactive Mode, the system is expected to serve up to 50000 voters instantly and each voter may be active for a long time. Similarly in Election Mode, the system is expected to serve a maximum of up to 50000 voters however each voter may be active for at 5 minutes for voting operation.

Since the ONEV is a safety critical system, security and safety constraints are the main issues of the system. The system should provide means for protecting and securing recounts of ballots cast in election. By using SSL technologies the data transaction between client and server will be encrypted and all the passwords will be stored in database in an encrypted form. A random word will be generated by the system to prevent attacks and the system will ask the user to enter it correctly for multiple trying.

For performance constraints the system will response in a reasonable short time. The voter should be able to login and should be able to get response in 2-3 seconds. In Election Mode, the system will handle about 2000 transactions each second and it will be working at 100% peak efficiency during voting process.

Apart from these constraints the system should satisfy the some assumptions and dependencies such as a working internet connection, a web server Java installed on the machine with Java's cryptographic packages. Also the election server will run on a http server that JSP is enabled.

3.2. Design Goals and Guidelines

Since our system is a safety critical system, in design of system architecture and database we have to take security principles into account. Since the system will work on web services, it must prevent all attacks from the outside and only authorized people must access the database. It must prevent the manipulation of the votes from unauthorized people.

Also another major principle that the system must provide is reliability. People must rely on the system and they must use the system in confidence. The system must not keep information about which voter cast to which party during execution. The main function of the system must be correct and fast calculation of the votes and results.

For interface designs we have to follow KISS principle. Because for voting operation, every voter has different technological and educational background so the interfaces must be clear to every user. For voting task the voter will only use a radio button to selection operation and a submit button to casting operation. The other interfaces will also designed clearly and simple to all stakeholders.

4. Data Design

4.1. ER Design

The poll server runs on http server that is enabled to handle server pages. It uses a relational database to keep track of the polls, which it connects through standard database connectivity interfaces (figure 6). In order to run the setup software, the environment needs to have a Java Virtual Machine running on it.

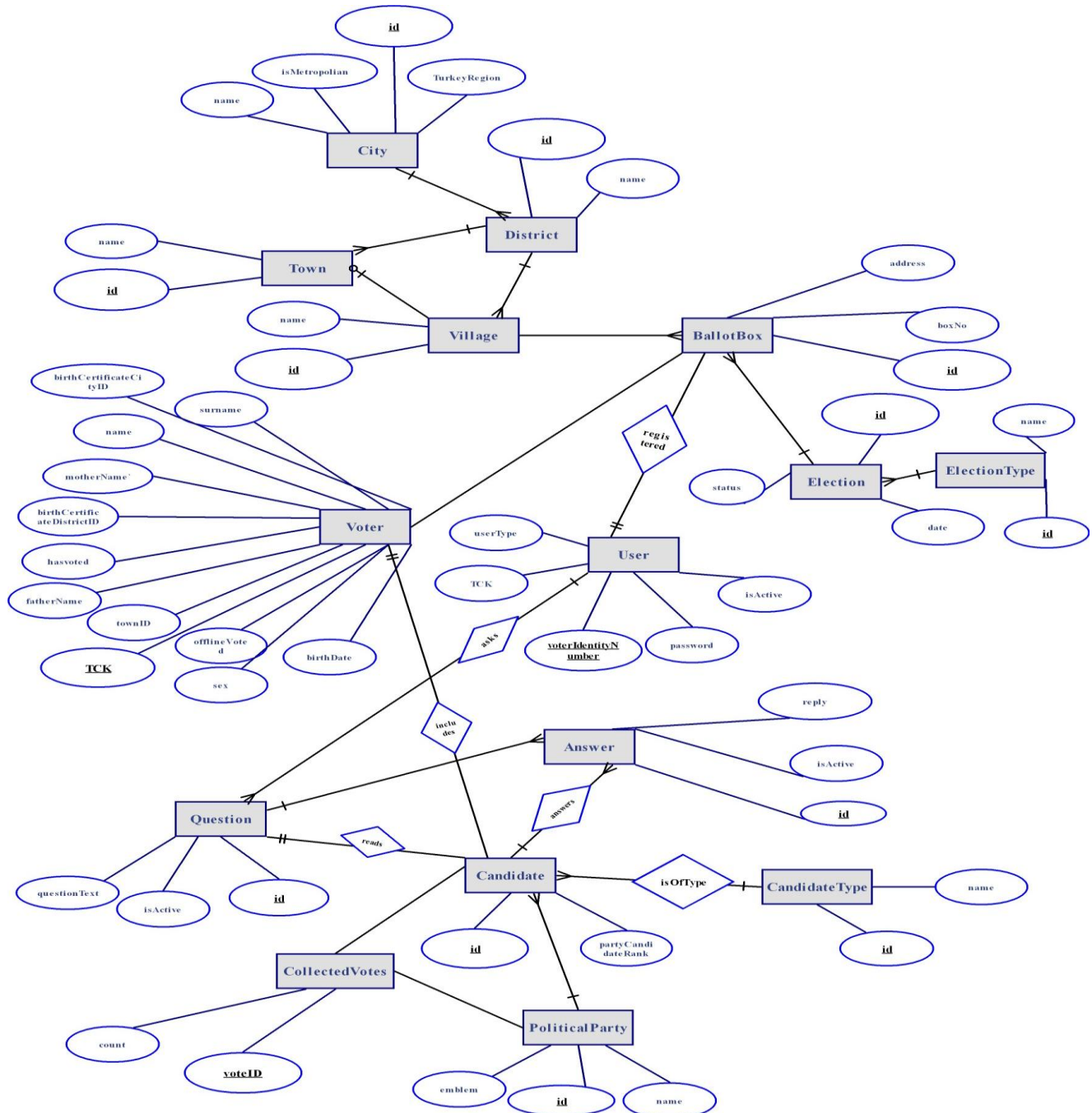


Figure 6: ER Diagram

4.2. Data Schemas

To keep information of some data's location information we designed following tables. Turkey is divided into Cities (il), Cities are composed of Districts (ilçe), and Districts are composed of both Towns (Belde) and Villages (Mahalle, Köy). Towns are the set of Villages. One exception is: villages do not need to be bound to towns. Some villages are directly bound to districts.

4.2.1. City Table

Field	Type	Null	Foreign Key	References
<u>id(P.K.)</u>	Integer	No	No	-
name	Nvarchar(50)	No	No	-
isMetropolitan	Boolean	No	No	-
TurkeyRegion	Integer	No	No	-

Table 2: A database table representing attributes of the City.

City table (Table 2 above) holds basic attributes of item city. Its primary key is id.

4.2.2. District Table

Field	Type	Null	Foreign Key	References
<u>id(P.K.)</u>	Integer	No	No	-
name	Nvarchar(50)	No	No	-
cityID	Integer	No	Yes	City

Table 3: A database table representing attributes of the District.

This table (table 3 above) holds attribute 'name' to keep the name of the district. Its primary key is id.

And it also includes cityID as a foreign key, so we can understand to which city it is bound.

4.2.3. Town Table

Field	Type	Null	Foreign Key	References
<u>id(P.K.)</u>	Integer	No	No	-
name	Nvarchar(50)	No	No	-
cityID	Integer	No	Yes	City
districtID	Integer	No	Yes	District

Table 4: A database table representing attributes of Town.

Town table holds information about towns. Its primary key is id. We could only give districtID as a foreign key and avoid giving cityID as a foreign key. The main reason is, most often we want to know information of towns or villages of some specific city. To, avoid additional query execution we designed as shown above.

4.2.4. Village Table

Field	Type	Null	Foreign Key	References
<u>id(P.K.)</u>	Integer	No	No	-
name	Nvarchar(50)	No	No	-
cityID	Integer	No	Yes	City
districtID	Integer	No	Yes	District
townID	Integer	Yes	Yes	Town

Table 5: A database table representing attributes of Village.

Village table also holds informations such as its name, city, district and town. Its primary key is id.

4.2.5. Voter Table

Field	Type	Null	Foreign Key	References
<u>TCK(P.K.)</u>	Nchar(11)	No	No	-
Name	Nvarchar(50)	No	No	-
Surname	Nvarchar(50)	No	No	-
motherName	Nvarchar(50)	Yes	No	-
fatherName	Nvarchar(50)	Yes	No	-
Sex	Integer	No	No	-
Birthday	Date	No	No	-
cityID	Integer	No	Yes	City
districtID	Integer	No	Yes	District
townID	Integer	Yes	Yes	Town
villageID	Integer	No	Yes	Village
birthCertificateCityID	Integer	No	Yes	City
birthCertificateDistrictID	Integer	No	Yes	District
boxID	Integer	No	Yes	BallotBox
hasVoted	Boolean	No	No	-
hasOfflineVoted	Boolean	Yes	No	-

Table 6: A database table representing information attributes of a Voter.

Voter table holds information about official voters such as their registered address, where they born, name, surname, sex, birthday, sex, etc. Its primary key is TCK (TC Kimlik No). It also includes boxID as a foreign key to BallotBox to keep information in which station he uses his vote. 'hasVoted' attribute is used to know whether voter has voted or not. 'hasOfflineVoted' keeps information if voter has voted 'Offline' – with paper. If 'hasOfflineVoted' is false, it means that the voter used ONEV system and voted 'Online'.

Below, the tables related to Election are described.

4.2.6. Election Table

Field	Type	Null	Foreign Key	References
<u>id(P.K.)</u>	Integer	No	No	-
electionType	Integer	No	Yes	ElectionType
date	Date	No	No	-
isActive	Boolean	No	No	-

Table 7: A database table representing attributes of Election Table.

Since our system should hold past elections' results, we must have election table to hold results for every election. Users can see filtered results of any past election.

We keep electionType, date and isActive to determine type of the election, date it occurred and if it is active or not.

4.2.7. ElectionType Table

Field	Type	Null	Foreign Key	References
<u>id(P.K.)</u>	Integer	No	No	-
Name	Integer	No	No	-

Table 8: A database table representing attributes of Election classifications.

Our system can handle every kind of election. Now, there are four types of election in Turkey. These are: Genel Seçim, Yerel Seçim, Cumhurbaşkanlığı Seçimi and Referandum. The voting behavior is different for every type of election.

4.2.8. BallotBox Table

Field	Type	Null	Foreign Key	References
<u>id(P.K.)</u>	Integer	No	No	-
boxNo	Integer	No	No	-
electionID	Integer	No	Yes	Election
cityID	Integer	No	Yes	City
districtID	Integer	No	Yes	District
townID	Integer	Yes	Yes	Town
villageID	Integer	No	Yes	Village
Address	Text	No	No	-

Table 9: A database table representing attributes of Ballot Box.

This table is to hold information about Election Centers (Sandık).

'boxNo' is numbering of boxes. But this numbering is particular to every city. Because of this, we haven't marked it as a primary key. The box's place information is can be found by its city, district, town and village.

4.2.9. User Table

Field	Type	Null	Foreign Key	References
<u>voterIdentityNumber(P.K.)</u>	Nchar(15)	No	No	-
Password	Nvarchar(50)	No	No	-
isActive	Boolean	No	No	-
TCK	Nchar(11)	No	Yes	Voter
UserType	Integer	No	Yes	UserType

Table 10: A database table representing attributes of User Information.

User table holds information about the registered user of ONEV. It holds basic attributes of the user entity such as voterIdentityNumber, password, and userType. TCK is a foreign key to Voter table, so detailed information of the user is kept in Voter table.

Primary key of the user table is voterIdentityNumber.

4.2.10. UserType Table

Field	Type	Null	Foreign Key	References
<u>id(P.K.)</u>	Integer	No	No	-
Type	Nvarchar(50)	No	No	-

Table 11: A database table representing attributes of User Types.

In our system, there is more than one type of users. These are Voter, Candidate and ECA. This table is to hold types of users.

4.2.11. PoliticalParty Table

Field	Type	Null	Foreign Key	References
<u>id(P.K.)</u>	Integer	No	No	-
Name	Nvarchar(50)	No	No	-
Rank	Integer	No	No	-
emblem	Image	No	No	-

Table 12: A database table representing information regarding Political Parties.

This table holds the list of Political Parties.

The 'rank' attribute is used to keep the rank of the specific Party among Parties to be showed in 'Voting Card' or in our system while voting.

4.2.12. Candidate Table

Field	Type	Null	Foreign Key	References
<u>candidateID(P.K.)</u>	Integer	No	No	-
TCK	Nchar(11)	No	Yes	Voter
candidateType	Integer	No	No	-
partyID	Integer	Yes	Yes	Party
partyRank	Integer	Yes	No	-
electionID	Integer	No	Yes	Election
cityID	Integer	Yes	Yes	City
districtID	Integer	Yes	Yes	District
townID	Integer	Yes	Yes	Town
villageID	Integer	Yes	Yes	Village

Table 13: A database table representing attributes of Candidates.

This table holds basic information about Candidate.

It has foreign key TCK to Voter, so detailed information can be got from Voter table.

For candidates that are member of a party, its partyID is stored and is a foreign key to Party table.

partyRank is used to show the Candidate's rank among same party's candidates in his region.

candidateType is foreign key that is used to show the type of the candidate.

4.2.13. CandidateType Table

Field	Type	Null	Foreign Key	References
<u>id(P.K.)</u>	Integer	No	No	-
name	Integer	No	No	-

Table 14: A database table representing attributes of Candidate Types.

CandidateType table is used to hold types of candidates.

4.2.14. CollectedVote Table

Field	Type	Null	Foreign Key	References
<u>voteID(P.K.)</u>	Integer	No	No	-
boxID	Integer	No	Yes	BallotBox
partyID	Integer	Yes	Yes	PoliticalParty
candidateID	Integer	Yes	Yes	Candidate
voteCount	Integer	No	No	-

Table 15: A database table representing attributes of Votes already collected.

This table (Table 15) is used to hold information of collected votes of a party or an individual candidate.

boxID is a foreign key to BallotBox, to show from which box the result is.

partyID is to show which party's result this is.

candidateID is to show which candidate's result this is.

4.2.15. Question Table

Field	Type	Null	Foreign Key	References
<u>id(P.K.)</u>	Integer	No	No	-
userID	Integer	No	Yes	User
candidateID	Integer	No	Yes	Candidate
questionText	Text	No	No	-
isActive	Text	No	No	-

Table 16: A database table representing attributes of Questions addressed to candidates.

In our system, users can ask questions to candidate. This table is used for that aim.

4.2.16. Answer Table

Field	Type	Null	Foreign Key	References
<u>id(P.K.)</u>	Integer	No	No	-
questionID	Integer	No	Yes	Question
candidateID	Integer	No	Yes	Candidate
reply	Text	No	No	-
isActive	Boolean	No	No	-

Table 17: A database table representing attributes of Answers to questions addressed to candidates.

This table holds answers to question. There can be more than one answer for a question. So, id is a primary key, not questionID.

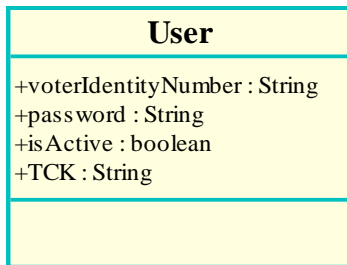
4.3. Data Dictionary

In this section the class diagrams that will be used during implementation are described with their attributes and methods.

4.3.1. Data Classes

In this class only attributes are encapsulated. These attributes will be controlled by controller classes.

- **User Class**



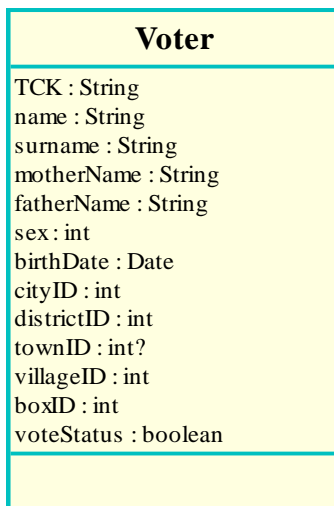
voterIdentityNumber: This will be used to keep the value of VIN that will be given to the voters before the election.

password: This attribute will keep the password information of the users in string format.

isActive: A Boolean variable to state whether the user is active or not.

TCK: The TCK number of the user in string format.

- **Voter Class**



TCK: The TCK value of the voter in string format.

Name: To keep the name of the voter.

Surname: To keep the surname of the voter.

MotherName: To keep the mother name of the voter.

FatherName: To keep the father name of the voter.

Sex: It will be used for identifying the sex of the voter as integer variable.

birthDate: To keep the birth date of the voter in data variable format.

cityID: To keep the city of the voter that he will cast the vote.

districtID: To keep the district of the voter that he will cast the vote.

townID: To keep the town of the voter that he will cast the vote.

villageID: To keep the village of the voter that he will cast the vote.

boxID: To keep the box of the voter that he will cast the vote.

voteStatus: A Boolean variable that state whether the casting has done or not.

- **Political Party Class**

PoliticalParty
id : int name : String rowNo : int emblem : Image

Id: To distinguish the political party with an integer variable.

Name: To keep the name of the political party in string format.

rowNo: to state the political party position on election day.

Emblem: An image to keep the political party emblem.

- **Candidate Class**

Candidate
id : int TCK : String candidateType : String partyID : int partyCandidateRowNo : int? electionID : int cityID : int? districtID : int? townID : int? mahalleID : int?

Id: To keep the candidate id value in integer format.

TCK: To keep the TCK value of the candidate as string.

candidateType: To keep the candidate type in string format.

partyID: States the political party of the candidate.

PartyCandidateRow: To state the position of the candidate in the election list of his political party.

electionID: To keep the which election the candidate joins.

cityID: States the city of the candidate that he joins the election.

districtID: States the city of the candidate that he joins the election.

townID: States the town of the candidate that he joins the election.

mahalleID: States the “mahalle” of the candidate that he joins the election.

- **Election Class**

Election
id : int electionType : int date : Date status : boolean

Id: To distinguish the elections based on their id numbers.

ElectionType: States the type of the election in integer format.

Date: To keep the date of the election in Date variable.

Status: To state whether the election has been done or not in Boolean variable.

- **Ballot Box Class**

BallotBox
id : int boxNo : int electionID: int cityID : int districtID : int townID : int? villageID : int address : String

Id: To keep the id number of the ballot.

Boxno: States the ballot number in integer variable.

electionID: To keep the election id number for correct election.

cityId: To keep the city of the ballot that it presents in integer format.

districtID: To keep the district of the ballot that it presents in integer format.

townID: To keep the town of the ballot that it presents in integer format.

villageID: To keep the village of the ballot that it presents in integer format.

address: To keep the address of the ballot that it presents in string variable.

- **Collected Vote Class**

CollectedVotes
votedID : int boxID : int partyID : int? candidateID : int?

votedID: To keep the voted id number in integer format.

boxID: States the ballot id that votes collected.

partId: States the party to keep the party's votes according to their id values.

candidateID: States the candidate id to keep the candidate's votes according to their id values.

4.3.2. User Controller Classes

These classes have no attributes and they have methods that control the user actions.

- **VoterUserManager Class**

VoterUserManager
addVoterUser(VoterUser) : bool updateVoterUser(VoterUser) : bool deleteVoterUser(VoterUser) : bool getVoterUserByID(string) : VoterUser setPass word(String) : bool

addVoterUser(VoterUser): Takes a voter user and add to the system. It returns a Boolean variable whether the addition has done successfully or not.

updateVoterUser(Voteruser): Takes a voter user and update the information about this object. It returns a Boolean variable whether the configuration has done successfully or not.

deleteVoterUser(Voteruser): Takes a voter user and delete it from the voter list. It returns a Boolean variable whether the deletion has done successfully or not.

getVoterUserByID(string): Takes a id value and returns the correct voter based on his id.

setPassword(string): To set the password of the voter. It takes password as string value and returns a Boolean variable whether the setting has done correctly or not.

- **CandidateUserManager Class**

CandidateUserManager
addCandidateUser(CandidateUser) : bool updateCandidateUser(CandidateUser) : bool deleteCandidateUser(CandidateUser) : bool getCandidateUserByID(string) : CandidateUser setPassword(String) : bool

addCandidateUser(CandidateUser): Takes a candidate user and add to the system. It returns a Boolean variable whether the addition has done successfully or not.

updateCandidateUser(Candidateuser): Takes a candidate user and update the information about this object. It returns a Boolean variable whether the configuration has done successfully or not.

deleteCandidateUser(Candidateuser): Takes a candidate user and delete it from the voter list. It returns a Boolean variable whether the deletion has done successfully or not.

getCandidateUserByID(string): Takes a id value and returns the correct candidate based on his id.

setPassword(string): To set the password of the candidate. It takes password as string value and returns a Boolean variable whether the setting has done correctly or not.

- **ECAUserManager Class**

ECAUserManager
addECAUser(ECA User) : bool updateECAUserECAUser) : bool deleteECAeUser(ECA User) : bool getECA UserByID(string) : ECA User setPassword(String) : bool

addECAUser(CandidateUser): Takes a ECA user and add to the system. It returns a Boolean variable whether the addition has done successfully or not.

updateECAUser(Candidateuser): Takes a ECA user and update the information about this object. It returns a Boolean variable whether the configuration has done successfully or not.

deleteECAUser(Candidateuser): Takes a ECA user and delete it from the voter list. It returns a Boolean variable whether the deletion has done successfully or not.

getECAUserByID(string): Takes a id value and returns the correct ECA based on his id.

setPassword(string): To set the password of the ECA. It takes password as string value and returns a Boolean variable whether the setting has done correctly or not.

- **AuthenticationManager Class**

AuthenticationManager
login() : bool authenticate(User) : bool logout(User) : bool isLoggedIn(User) : bool getCurrentUser() : User createSession() : bool

Login(User): This function returns a Boolean variable whether the login operation has done correctly or not.

Authenticate(User): This method takes an user variable and makes the authentication of the user.

Logout(User): This function returns a Boolean variable whether the logout operation has done correctly or not.

isLoggedIn(User): To state whether the user is logged out or not in Boolean variable.

getCurrentUser(): Returns the current user of the system as user object.

createSession(): To create the session for the current user. It returns a Boolean variable for the successful creation.

4.3.3. Data Controller Classes

These classes have no attributes and they have methods that control the data relations.

- **Candidate Manager Class**

CandidateManager
getCandidateByID(int) : Candidate getCandidateByTCK(String) : Candidate addCandidate(Candidate) : boolean updateCandidate(Candidate) : boolean deleteCandidate(Candidate) : boolean setPartyID(int) : bool getParty() : PoliticalParty setPartyRank(int) : boolean getQuestions(Candidate) : Question[] getCollectedVotes(Candidate) : CollectedVotes getAnswers(Candidate) : Answer []

getCandidateByID(int): Takes id number as parameter and returns the candidate object based on the his id number.

getCandidateByTCK(string): Takes TCK as parameter and returns the candidate object based on the his TCK value.

addCandidate(Candidate): Takes a candidate object and adds to the system. Returns a Boolean variable for correct addition operation.

updateCandidate(Candidate): Takes a candidate object and configure the information about the candidate. Returns a Boolean variable for correct update operation.

deleteCandidate(Candidate): Takes a candidate object and deletes the candidate from the system. Returns a Boolean variable for correct deletion operation.

setPartyID(int): Sets the party of the candidate.

getParty(): Returns the political party of the candidate that he is member of.

setPartyRank(int): Sets the position of the candidate in the party list.

getQuestions(Candidate): Returns the questions that asked to the candidate in array format.

getCollectedVotes(Candidate): Takes a candidate as a parameter and returns the collected votes.

getAnswers(Candidate): Returns the answers of the candidates to the questions in array format.

- **PoliticalPartyManager Class**

PoliticalPartyManager
<pre>getPoliticalPartyByID(int) : PoliticalParty getPoliticalPartyByName(String) : PoliticalParty addPoliticalParty(PoliticalParty) : boolean updatePoliticalParty(PoliticalParty) : boolean deletePoliticalParty(PoliticalParty) : boolean setRank(int) : boolean setEmblem(Image) : boolean</pre>

getPoliticalPartyByID(int): This method takes the political party id and returns the correct political part having this id number.

getPoliticalPartyByName(string): This method takes the political party name and returns the correct political part having this name.

addPoliticalParty(PoliticalParty): This method takes a political party object and add the political party list.

updatePoliticalParty(PoliticalParty): This method takes a political party object and update this party on political party list.

deletePoliticalParty(int): This method takes a political party object and delete this party from political party list.

setRank(int): To state the rank of the party in the election.

setEmblem(Image): Takes and image object and sets it as political party emblem.

- **ElectionManager Class**

ElectionManager
getElectionByID(int) : Election getActiveElection() : Election getElectionsByType(ElectionType) : Election [] addElection(Election) : boolean updateElection(Election) : boolean deleteElection(Election) : boolean setDate(Date) : boolean setElectionType(int) : boolean

getElectionByID(int): This method takes an integer value as election id number and returns the election.

getActiveElection(): This method takes no argument and returns the active election object.

getElectionsByType(ElectionType): This method returns the election lists based on the election type.

addElection(Election): Takes an election object and adds the election objects list.

updateElection(Election): Takes an election object and configures the object on the election object list.

deleteElection(Election): Takes the election object list and deletes from the list.

setDate(Date): To set the election date takes a date variable.

setElectionType(int): Takes an integer variable and sets the election type based on this integer value.

- **BallotBox Manager**

BallotBoxManager
getBallotBoxByID(int) : BallotBox getBBofCity(City) : BallotBox [] getBBofDistrict(District) : BallotBox [] getBBofTown(Town) : BallotBox [] getBBofVillage(Village) : BallotBox [] getBBofElection(Election) : BallotBox [] addBallotBox(BallotBox) : boolean updateBallotBox(BallotBox) : boolean deleteBallotBox(BallotBox) : boolean

getBallotBoxByID(int): Returns the ballot box object based on its id number.

getBBofCity(City): Takes the city as parameter and returns the ballots as array format on this city.

getBBofDistrict(District): Takes the district as parameter and returns the ballots as array format on this district.

getBofTown(Town): Takes the town as parameter and returns the ballots as array format on this town.

getBofVillage(Village): Takes the village as parameter and returns the ballots as array format on this village.

addBallotBox(BallotBox): Takes a ballot box object and adds ballots list.

updateBox(BallotBox): Configures the ballot box given as parameter on the ballot box list.

deleteBox(BallotBox): Takes a ballot box object and deletes it from the ballot box list.

- **QuestionManager Class**

QuestionManager
<pre>getQuestionByID(int) : Question getQuestionsFromUser(User) : Question [] getQuestionsToCandidate(Candidate) : Question [] getUnAnsweredQuestionsOfCandidate(Candidate) : Question [] addQuestion(Question) : boolean updateQuestion(Question) : boolean deleteQuestion(Question) : boolean</pre>

getQuestionByID(int): Takes an id number and returns the question object based on this id number.

getQuestionsFromUser(User): Takes an user parameter and returns the lists of the questions asked by this user as question array format.

getQuestionsToCandidate(Candidate): Takes a candidate parameter and returns the lists of the questions asked to this candidate as question array format.

getUnAnsweredQuestionsOfCandidate(Candidate): Takes an candidate parameter and returns the lists of the questions that is not answered by this candidate.

addQuestion(Question): Takes a question object and adds to the question list.

updateQuestion(Question): Takes a question object and updates this object in the question list.

deleteQuestion(Question): Takes a question object and deletes it from the question object list.

- **AnswerManager Class**

AnswerManager
<pre> getAnswerByID(int) : Answer getAnswersToUser(User) : Answer [] getAnswersOfQuestion(Question) : Answer [] getAnswersOfCandidate(Candidate) : Answer [] getQuestion(Answer) : Question addQuestion(Question) : boolean updateQuestion(Question) : boolean deleteQuestion(Question) : boolean </pre>

getAnswerByID(int): Takes an id number and returns the answer object based on this id number.

getAnswersToUser(User): Takes an user parameter and returns the lists of the answers related with this user as answer array format.

getAnswersOfQuestion(Question): Takes a question parameter and returns the lists of the answers to this question as answer array format.

getAnswersOfCandidate(Candidate): Takes an candidate parameter and returns the lists of the answers that this candidate response.

getQuestion(Answer): Takes an answer object and returns the corresponding question object.

addQuestion(Question): Takes a question object and adds to the question list.

updateQuestion(Question): Takes a question object and updates this object in the question list.

deleteQuestion(Question): Takes a question object and deletes it from the question object list.

5. System Architecture

5.1. Architectural Design

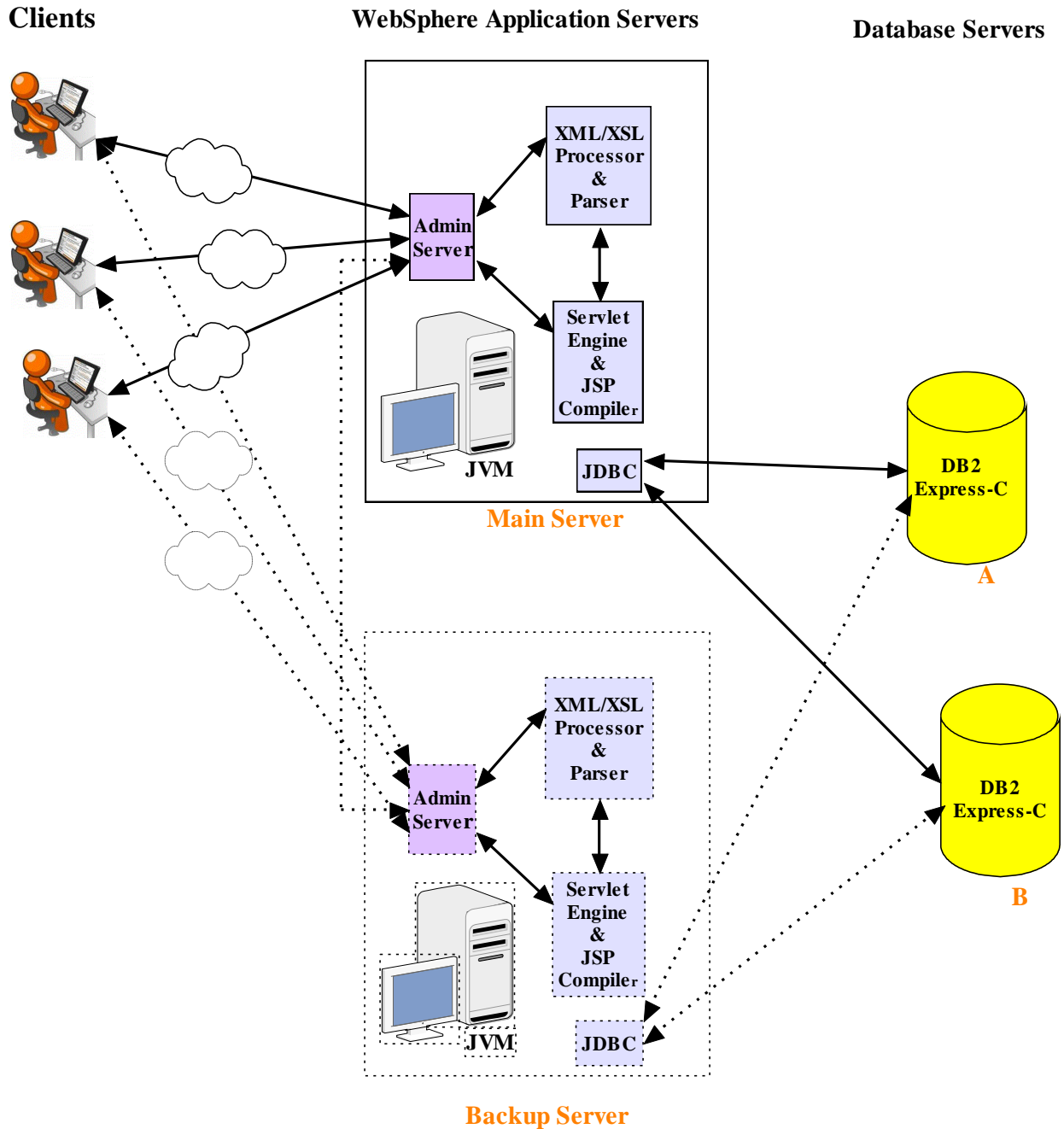


Figure 7: General View Of ONEV System

Basically, Our system is a 3-tier Client/Server architecture, as shown in figure 7, comprising of two databases, two Application servers and PC stations. The additional application server presented in dotted lines in the diagram above acts as a backup to the working main server. Therefore, during critical operations, in case of failure the reserve server comes into operation. The two databases work together during critical operations of polling votes. However, the backup server is responsible for storing critical information like votes and results of election. In the front phase of the system architecture lies the clients. The clients represent the PC centers formed throughout the country during election periods. It also represents any PC that can connect to our server during normal working days for regular applications like viewing election results, editing profiles and so on. The middle phase of the architecture comprises of Application servers we have discussed above. It should be noted that the servers consist of back-end applications to handle different tasks delegated by the administration server. The far end phase is comprised of storage subsystems, mainly the databases. These phases communicate in a formal protocol. That is, application server communicates directly with the clients and the storage devices. However, clients-databases communication is not directly. The application server – through a database connector- handles all database requests from the clients side to the database, as well as the responses are controlled by the server.

HIGH ABSTRACT MODULAR SYSTEM STRUCTURE

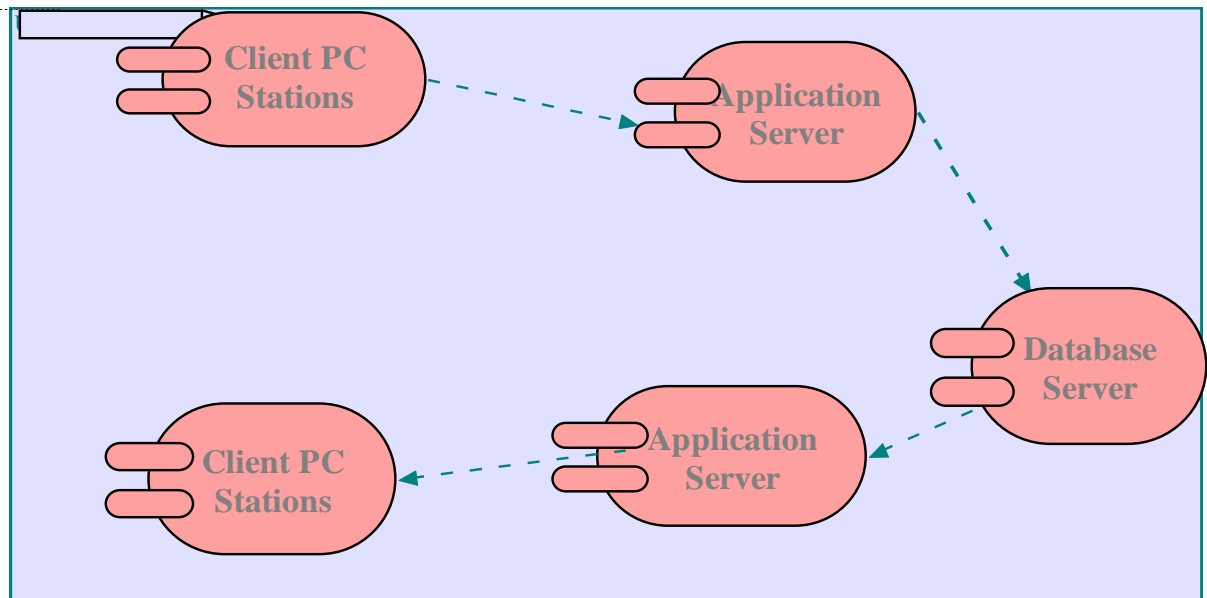


Figure 7: High Abstract Modular System Structure

The major components in the system can be represented in form of modules. Therefore, we have three unique major modules Clients, Application server, and Database server modules (figure 7). The diagram below shows the application sequence of the modules. The normal flow of actions in the system follows this order. A client issues a communication or data request with the server. The server (in many functions of the systems) checks the validity and eligibility of the client to the system by contacting the data storage server. Upon the response from the database server; the application server responds to the client request with positive or negative acknowledgement. Again, it should be noted that there is no direct communication between the clients and the database server.

5.2. Description of Components^[7]

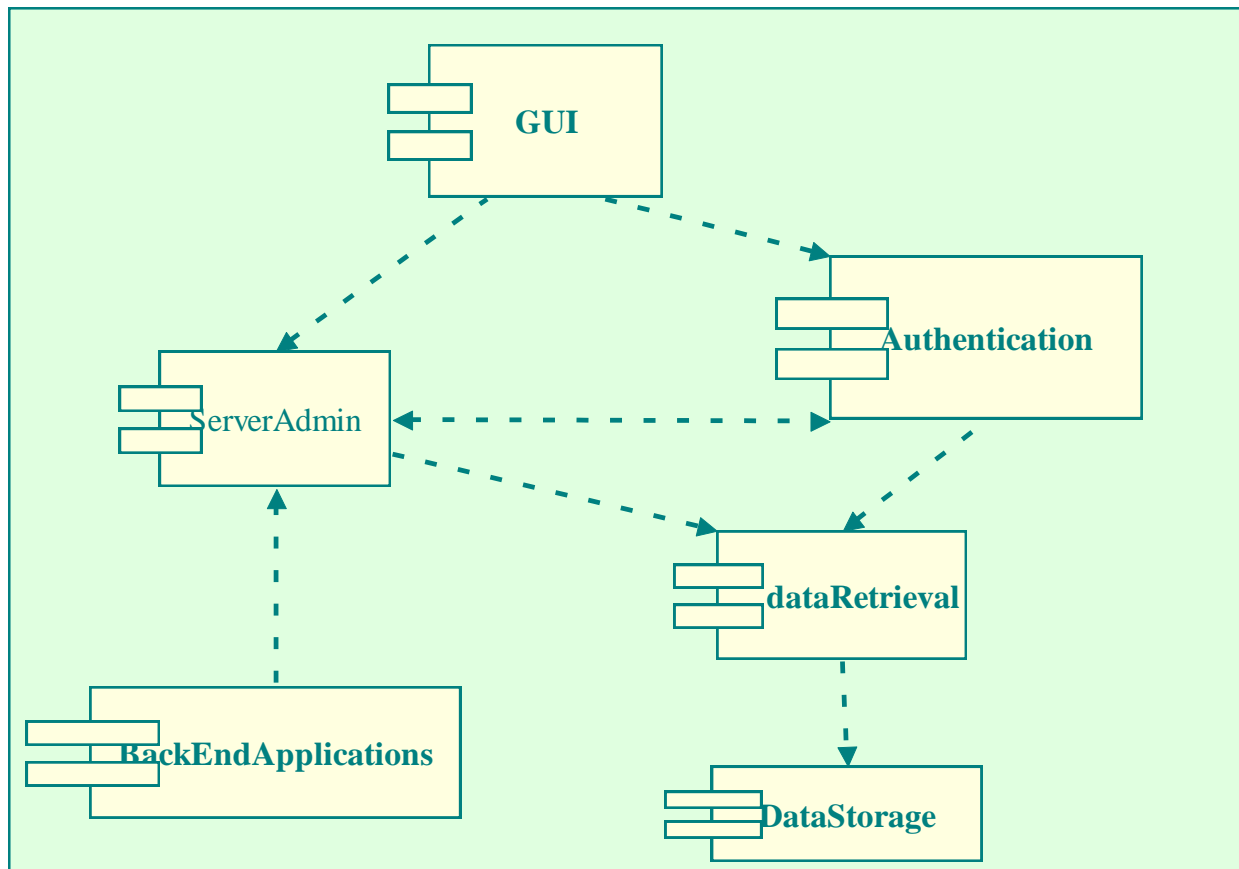


Figure 8: Components of the System

Our system can be subcategorized into six components according to major activities performed by the system (figure 8). The components are namely; Graphical User

Interface(GUI), Server Administrator(ServerAdmin), Authentication, Back End Applications(BackEndApplications), Data Retrieval(dataRetrieval) and data Storage(DataStorage).

5.2.1. Graphical User Interface

5.2.1.1. Processing Narrative for GUI

This component comprises all the objects that render the graphical User Interfaces with the appropriate contents. When a client issues an http request to the application server, a corresponding instance of class is issued by the Java Servlet^[6] to respond to and process the request. In addition to that, the component is responsible for creation of dynamic HTML webpage using JSP technology before sending them to the client side.

5.2.1.2. GUI Interface Description

The inputs to this component are the viewable webpage requests from the client side. On the other hand the outputs are the dynamically/statically created webpages to be displayed on the client side.

5.2.1.3. GUI Processing Detail

The complete step-by-step procedural activities related to this component are as follows;

1. User/client requests a page from the system through internet
2. Server Admin captures the request.
3. After processing administration tasks according to the type of request, Server Admin delegates the presentation of solution page(s) to the GUI component to create appropriate internet page.
4. The GUI presents the created page to the Server Admin to send it to the requester.

5.2.1.4. Dynamic Behavior of GUI

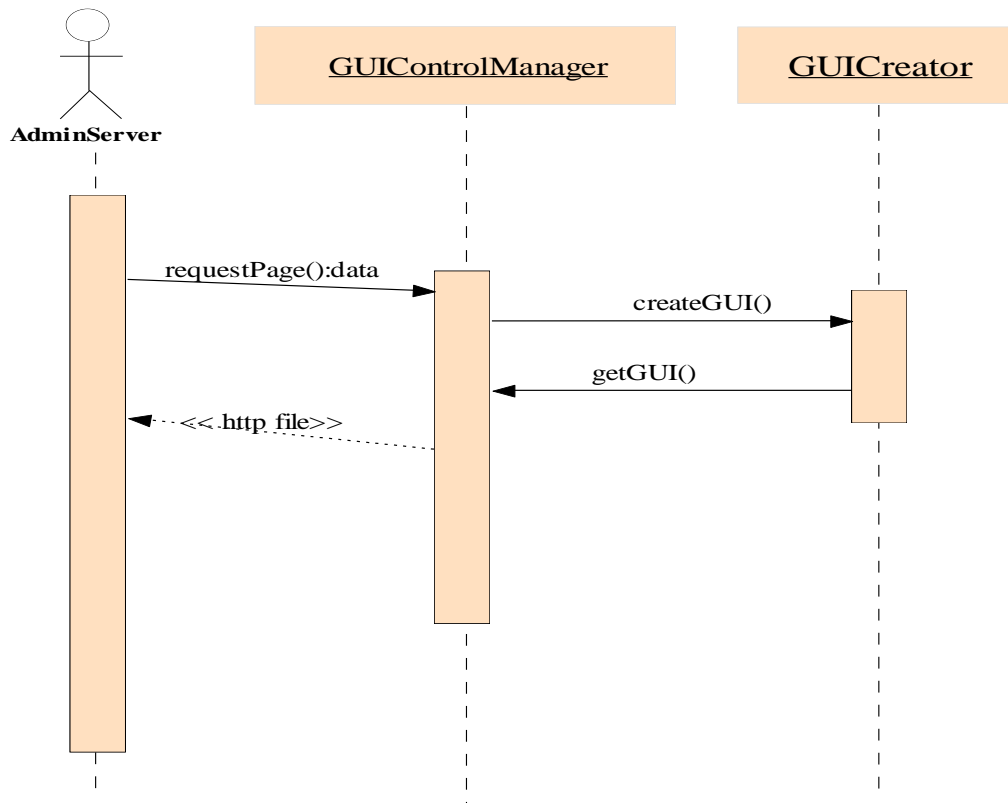


Figure 9: Sequence Diagram of GUI

5.2.2. Data Storage

5.2.2.1. Processing Narrative for Data Storage

This component is responsible for creating and storing data objects. Therefore it makes frequently requested data available instead of querying into the database frequently. It uses the JDBC connector to get data from the database and create corresponding objects with attributes and methods to access the data easily.

5.2.2.2. Data Storage Interface Description

It receives data requests from the dataRetrieval component as an input. Then it translates these into SQL commands and processes them using JDBC connector. The obtained result is put into an object. The object becomes available for future use.

5.2.2.3. Data Storage Processing Detail

It works as follows

1. It receives a request of data from dataRetrieval component
2. It issues the command through JDBC connector
3. The received response from the run queries and creates a corresponding object.

5.2.2.4. Dynamic Behavior of Data Storage

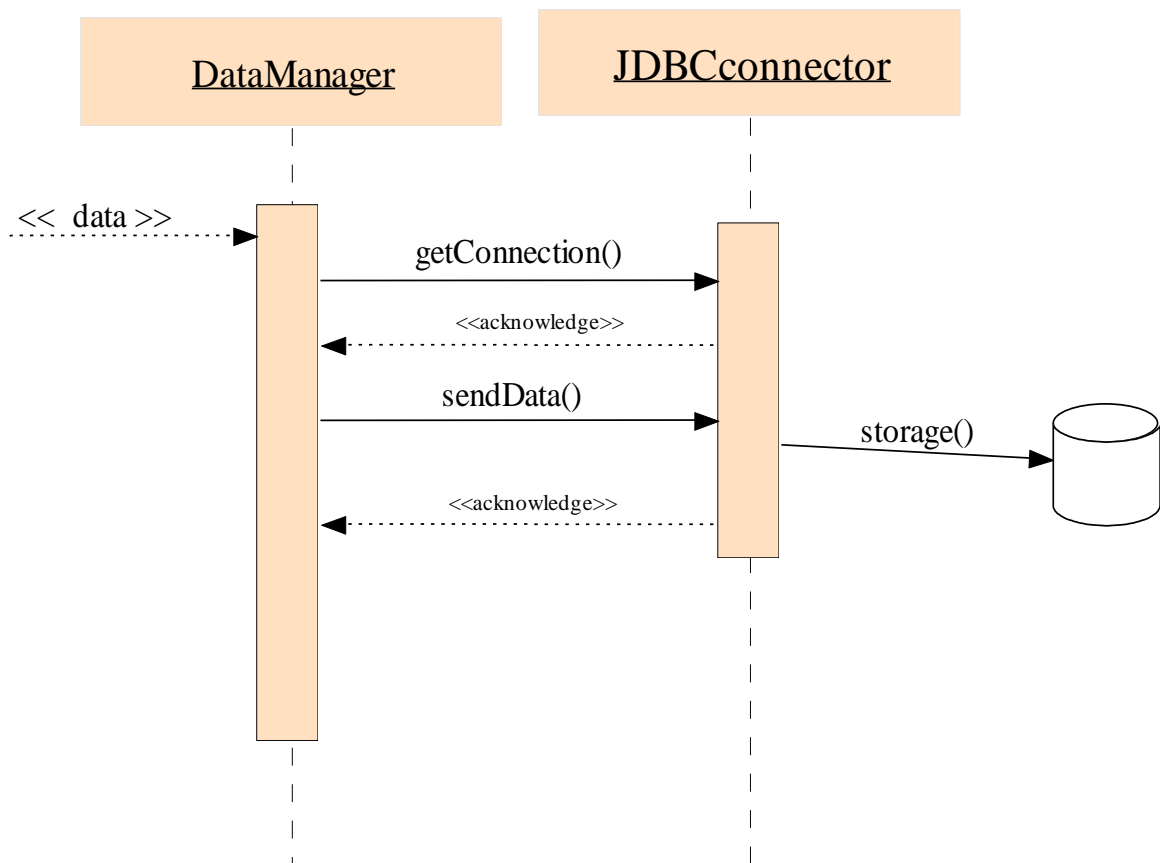


Figure 10: Sequence Diagram for Data Storage

5.2.3. ServerAdministrator(ServerAdmin)

5.2.3.1. Processing Narrative for ServerAdministrator

ServerAdmin is “a junction” between requests and responses. It receives HTTP requests from the client side and delegate the requests to respect servlets to process the requests. In addition to that, it collects the ready responses and sends them to the appropriate clients. It works closely with authentication component to authenticate the income requests before delegating them to the corresponding back end applications to process them.

5.2.3.2. ServerAdministrator Interface Description

It receives data packets online in form of HTTP protocols as an input. Using back end programs the packets are processed, the required information is extracted and the necessary steps taken into actions. It outputs HTTP responses and sends them to the clients via the internet.

5.2.3.3. ServerAdministrator Processing Detail

It works as follows

1. It receives a request from clients through HTTP.
2. It checks the validity of the request.
3. According to the type the request it assigns the request to a corresponding back end program.
4. When the request is processed it sends to the corresponding client

5.2.3.4. Dynamic Behavior of Server Admin

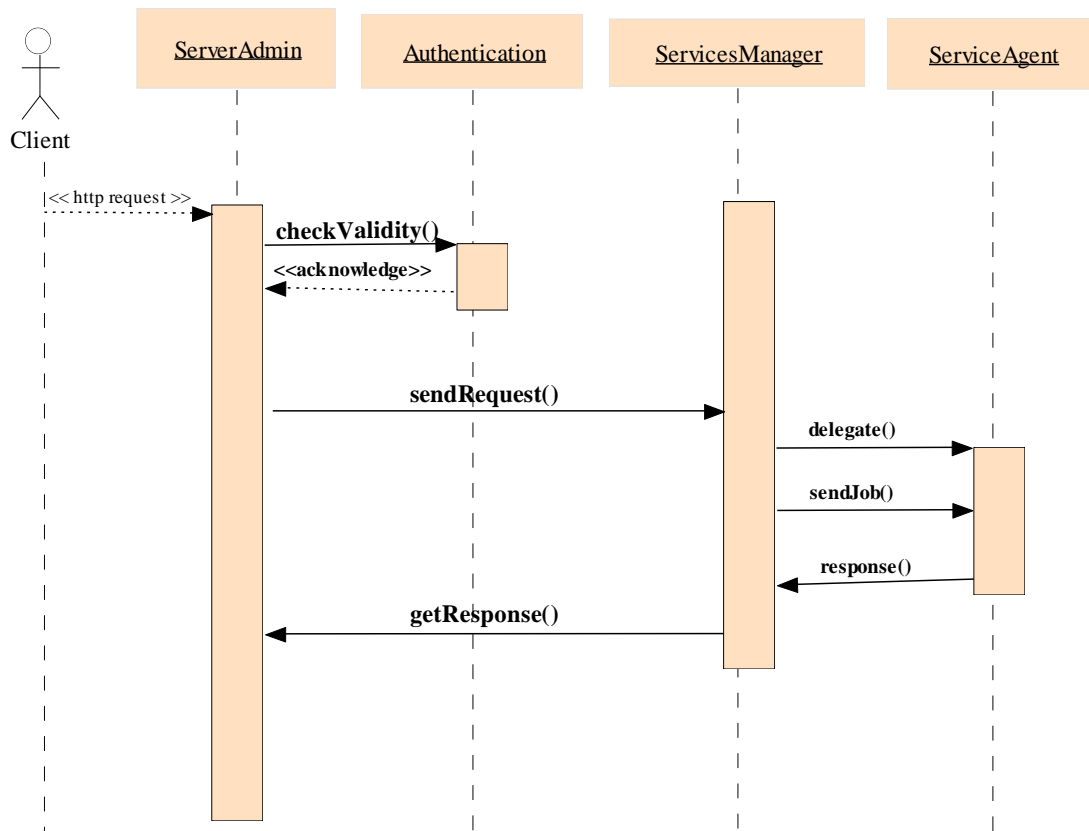


Figure 11: Sequence Diagram for Server Admin

5.2.4. Authentication

5.2.4.1. Processing Narrative for Authentication

This component is responsible for checking the critical requests with the permission of the clients. For example if a client tries to log on into the system Authentication checks if he is a registered user of the system according to the user identification and password. This is also the same when user wants to access some data. An election commission officer may be granted to view the voter profile while a voter cannot be granted the access the profile of other voters.

5.2.4.2. Authentication Interface Description

It receives commands as well as data from the ServerAdmin to help authenticate the process in question. The output is either request granted or denied. The output is directed to the ServerAdmin. It interacts with DataRetrieval in order to get data from the data storage component.

5.2.4.3. Authentication Processing Detail

It works as follows

1. It receives authentication request from ServerAdmin along with data.
2. It using the given data and that in the database it processes authentication.
3. It returns a grant or a denial response.

5.2.4.4. Dynamic Behavior Authentication

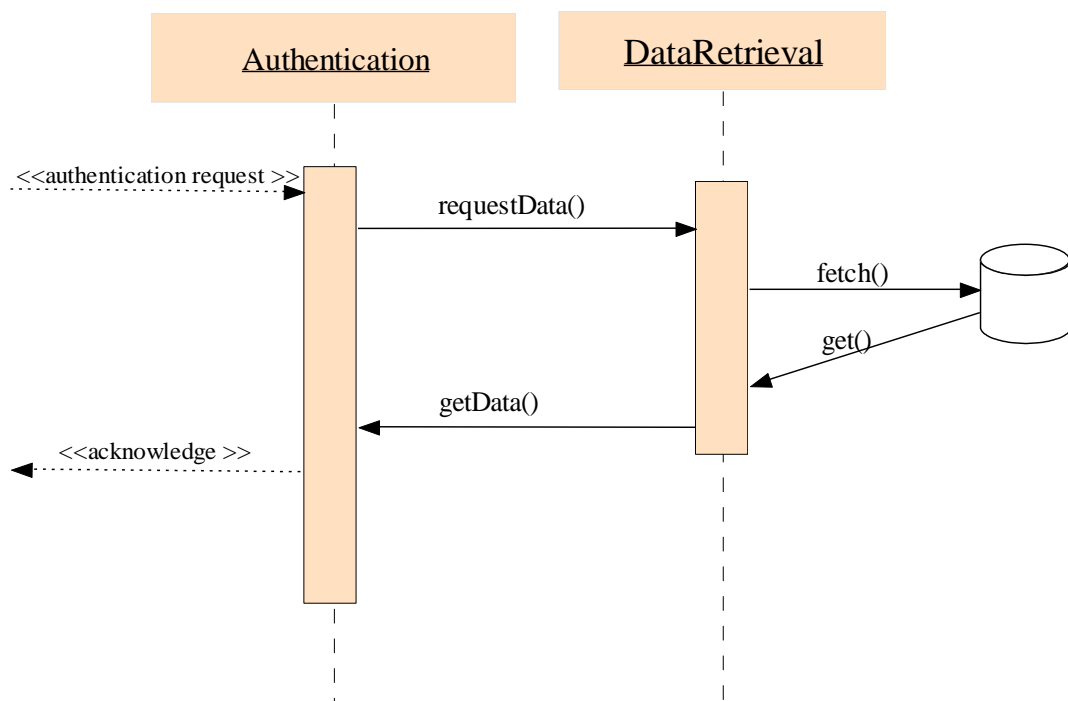


Figure 12: Sequence Diagram for Authentication

5.2.5. Back End Applications

5.2.5.1. Processing Narrative for Back End Applications

This includes technologies to handle different tasks and instantiate and serve different tasks delegated by ServerAdmin. The technologies involved include XML-parsers, JSP, Servlets, and the JVM.

5.2.5.2. Back End Applications Interface Description

In general the server task can be considered as an input to the back- end server. The output is the result of the back end server according to the requirements of the ServerAdmin.

5.2.5.3. Back End Applications Processing Detail

It works as follows

1. ServerAdmin triggers a job to the appropriate back- end application.
2. ServerAdmin provides appropriate input to the application.
3. The application processes the job
4. The application returns response to the ServerAdmin.

5.2.5.4. Dynamic Behavior of Back End Application

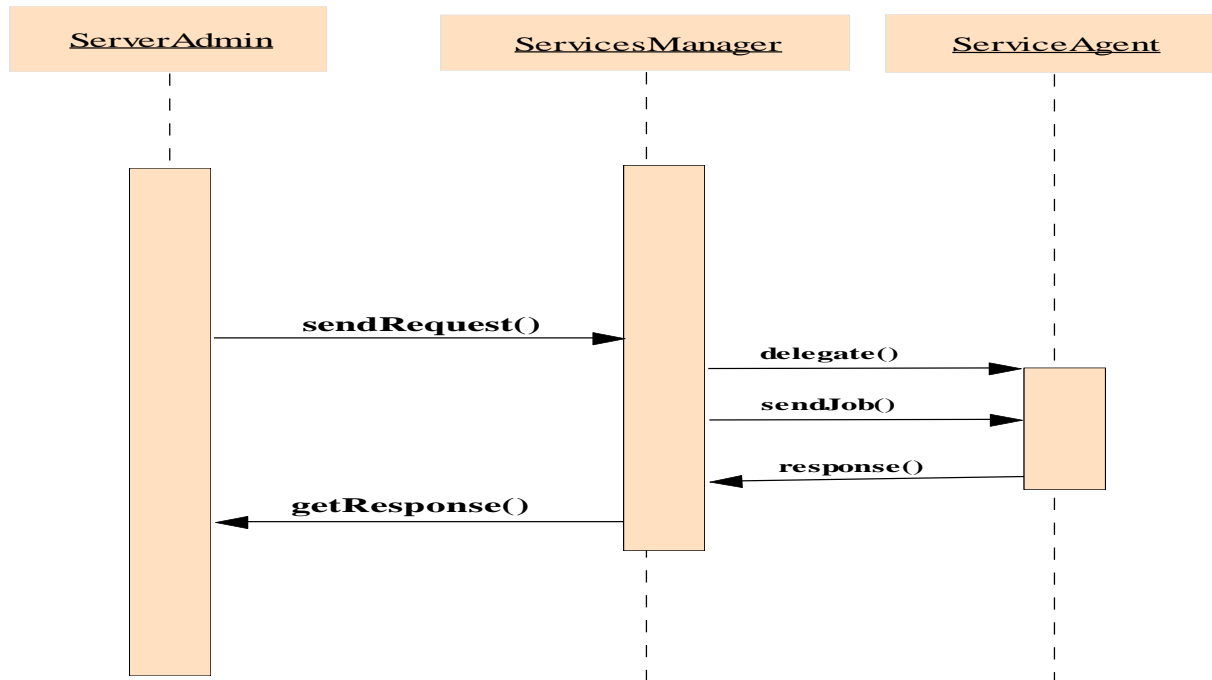


Figure 13: Sequence Diagram for Back End Applications

5.2.6. Data Retrieval

5.2.6.1. Processing Narrative for Data Retrieval

This component is responsible for accessing data from and storing data to the database. It acts as a bridge between the applications and the database objects.

It uses the JDBC connector to process the data queries in form of SQL commands.

5.2.6.2. Data Retrieval Interface Description

It receives data requests from the Server admin, authentication and the back end applications. Then it translates these into SQL commands and processes them using JDBC connector. The obtained result is returned as an object. The returned object is extracted to get the required data and reported to the component requested it.

5.2.6.3. Data Retrieval Processing Detail

It works as follows

1. It receives a request of data from Application server components
2. It translates the request into SQL command
3. It issues the command through JDBC connector
4. The received response from the run query is extracted to get the required data
5. The data is sent to the component asked for it.

5.2.6.4. Dynamic Behavior of Data Retrieval

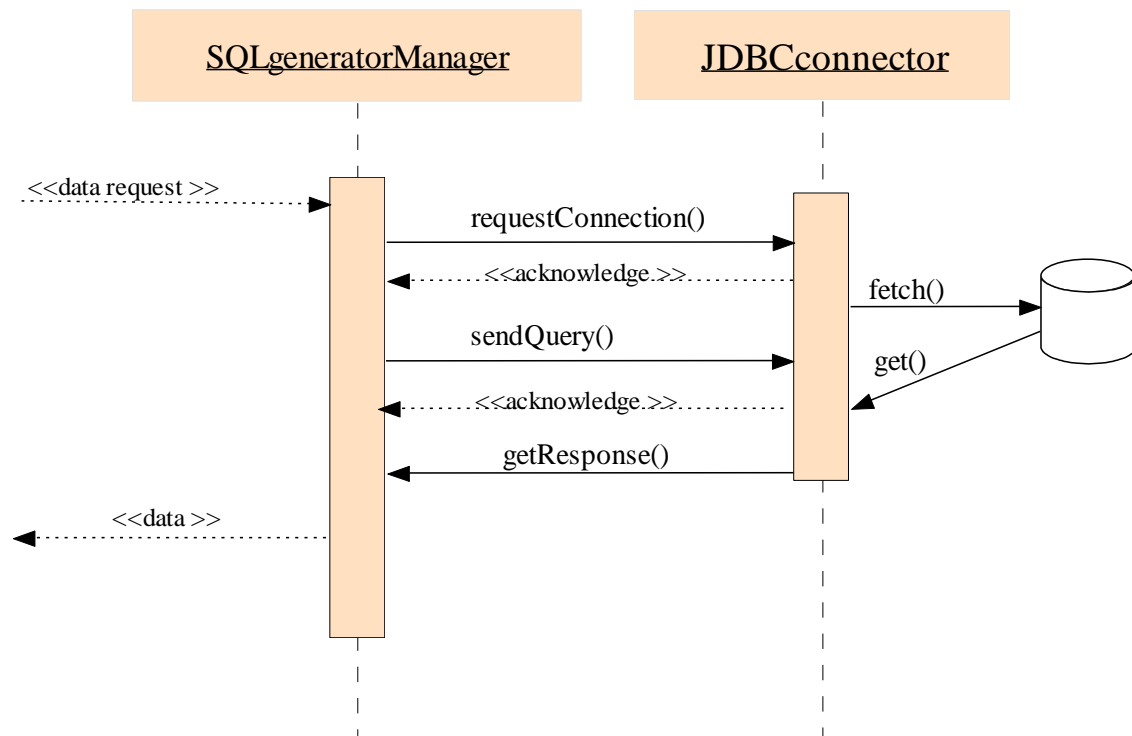


Figure 14: Sequence Diagram for Data Retrieval

5.3. Design Rationale

We separated the system into three major modules in order to keep the system simple, minimize cost and increase security level. As it can be seen from the system representation diagram there is much of computational activities rather than just presentation of windows as graphical user interfaces. The presence of one application server minimizes cost in terms of money and the cost of system distribution. All the necessary computations are carried out at the particular center. The presence of backup application server makes sure that the system is available most of the time even in the case the main application server encounters a problem that hinders its functioning.

Data storage is separate because we wanted to separate it completely from direct communication with the clients. Query issuing over the internet can be a threat and sometimes degrades performance. The communication between the application server and the database can be improved by storing the already queried data into the server machine, which we cannot do in the client machine to avoid insecurity.

Before concluding this architecture we had discussed architectures like Single Tier and Two Tier architectures. In Single tier architecture we decided to design an application that runs on a client machine (like a desktop application). However, due to criticality of the system, this cannot be possible because the system can be easily attacked by viruses in the client machine. The 2-tier architecture was totally inappropriate for our system because it requires storage of information in a formatted order for easier access. This is due to the fact that data storage and retrieval is more than 50% of all activities carried out by the system to meet the clients' needs.

6. User Interface Design

6.1. Overview of User Interface

Since the system consists of two parts user interfaces will be different in those two modes. In normal interactive mode there will be common home page interface for all system users and they will use this page for login operation.

In this mode voters interface will contain the links to view the candidates profiles and past years' election results. EC's interface will include his own profile and he will conduct the election tasks by using this interface. ECA interface will cover the functionalities related with registration of the voters and candidates.

In election mode there will be a major interface that the voting operation is executed. This interface will be used by the voters. And there will be another interface for the ESS. By using this interface the ESS's will generate a password for the voters used in casting operation and also he can enter the offline votes to the system.

6.2. Screen Images

In this part some of the screen images and their functionalities are described.

6.2.1. Login

A screenshot of a login interface. It features two input fields: the first is labeled 'kullanıcı adı' (username) and contains the text 'admin'; the second is labeled 'şifre' (password) and contains seven asterisks '*****'. Below these fields is a button labeled 'giris' (login).

Figure 15: User Interface of Login Page

This interface will be used by all of the system users and by entering the userid and password they will be able to use the system. For an incorrect password or userid the system will promote an error message to the users (figure 15).

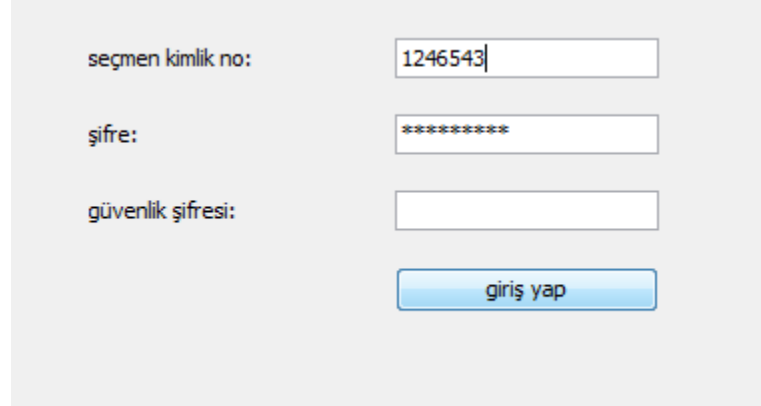
6.2.2. User Registration

The image shows a web form titled "Çevrimiçi Aday Kayıt Formu". It is divided into two main sections: "Kişisel Bilgiler:" (Personal Information) and "Adres Bilgisi:" (Address Information). In the "Kişisel Bilgiler:" section, there is a label "TC Kimlik No:" followed by a text input field containing the value "12345678901". In the "Adres Bilgisi:" section, there are five labels: "İl:", "İlçe:", "Belde:", "Mahalle:", and "Adres:". The "İl:" label is followed by a text input field containing the value "Ankara". The "İlçe:", "Belde:", and "Mahalle:" labels are each followed by an empty text input field. The "Adres:" label is followed by a larger, empty text input field. At the bottom left of the form, there is a button labeled "Bilgilerimi Gönder".

Figure 16: User Interface of Registration Page

The form in figure 16 will be used for the registration of the citizens to this system. We only require TCK of the citizen as personal information. We can get other required personal information such as birthday, sex, father's name, etc from governmental web service by providing only TCK. It will be easier for the user to register. Additionally, citizens must provide their address information. Then the official goes to that address and checks if the citizen is at that address or not. If the citizen is at that address and right to vote, then he will be approved.

6.2.3. Login For Vote



The login interface features three input fields and a button. The first field is labeled 'seçmen kimlik no:' and contains the text '1246543'. The second field is labeled 'şifre:' and contains '*****'. The third field is labeled 'güvenlik şifresi:' and is empty. Below these fields is a blue button labeled 'giriş yap'.

Figure 17: User Interface of Voting Stage Login Page

The interface in figure 17 will be used by the voters during the election mode in voting process. Before casting the vote, the voter must provide his Voter Identity Number, password and security password generated by the ESS. After entering the correct values the voter can reach the voting interface.

6.2.4. Voting



The voting interface displays a header with four checkboxes: 'İl Genel Meclis Üyesi' (checked), 'Büyükşehir Belediye Başkanlığı', 'İl Genel Meclis Üyesi', and 'Muhtarlık'. Below this, a bold text reads 'Büyükşehir Belediye Başkanlığına Oy Kullanıyorsunuz'. The main area is divided into two columns: 'Partiler Listesi' and 'Bağımsız Adaylar Listesi'. The 'Partiler Listesi' column contains three radio buttons labeled 'Parti 1', 'Parti 2', and 'Parti 3', each followed by 'Parti 1 Amblem', 'Parti 2 Amblem', and 'Parti 3 Amblem' respectively. The 'Bağımsız Adaylar Listesi' column contains three radio buttons labeled 'Aday 1', 'Aday 2', and 'Aday 3'. At the bottom, there are two buttons: 'Oyumu Kaydet ve Sonraki Aşamaya Geç' and 'Oy Kullanmadan Sonraki Aşamaya Geç'.

Figure 18: User Interface of Voting Page

After the voter logged in successfully, the interface in figure 19 is used for casting vote. In our system, the user interfaces will be simple and clear since stakeholders of the system have different educational, technological background. The Voter casts for only one candidate type and go to next page for the next type of candidate casting.

6.3. Screen Objects and Actions

Since the users interact with our system through web browser, our objects will be html elements. Some of the main objects and their functionalities are described below:

- **Label**
The <label> tag defines a label for an input element (Password Field, Text Field). In our application, we use labels for every important input element. If the user clicks on the text within the label element, it toggles the input element.
- **Text Field, Password Field**
When the user fills these fields and sends the form, the server gets filled values and do some transactions and returns results according to given values. We use these objects in order to get required information.
- **Check Box**
When we want to get only 'Yes – No' or 'True – False' information for the specific question we use check boxes.
- **Radio Button**
When the user is forced to choose only one option from the list, the radio button is used. The main usage of this object is at voting process. To illustrate, voter chooses only one political party or individual candidate from the list.
- **Submit Button**
A submit button is used to send form data to a server. The data is sent to the page specified in the form's action attribute.[3]
- **Hyperlink**
A hyperlink (or link) is a word, group of words, or image that you can click on to jump to a new document or a new section within the current document. Hyperlink's difference from Submit Button is, it does not send any field's values, it's aim is only to redirect to some other page.

7. Detailed Design

The ONEV is divided into two main packages namely Model Package and Controller Package as seen in figure 19 below. In model package contains the data description classes and manager package contains the functionalities that control the data objects. Also Controller Package has two sub-packages that control the user and data objects separately.

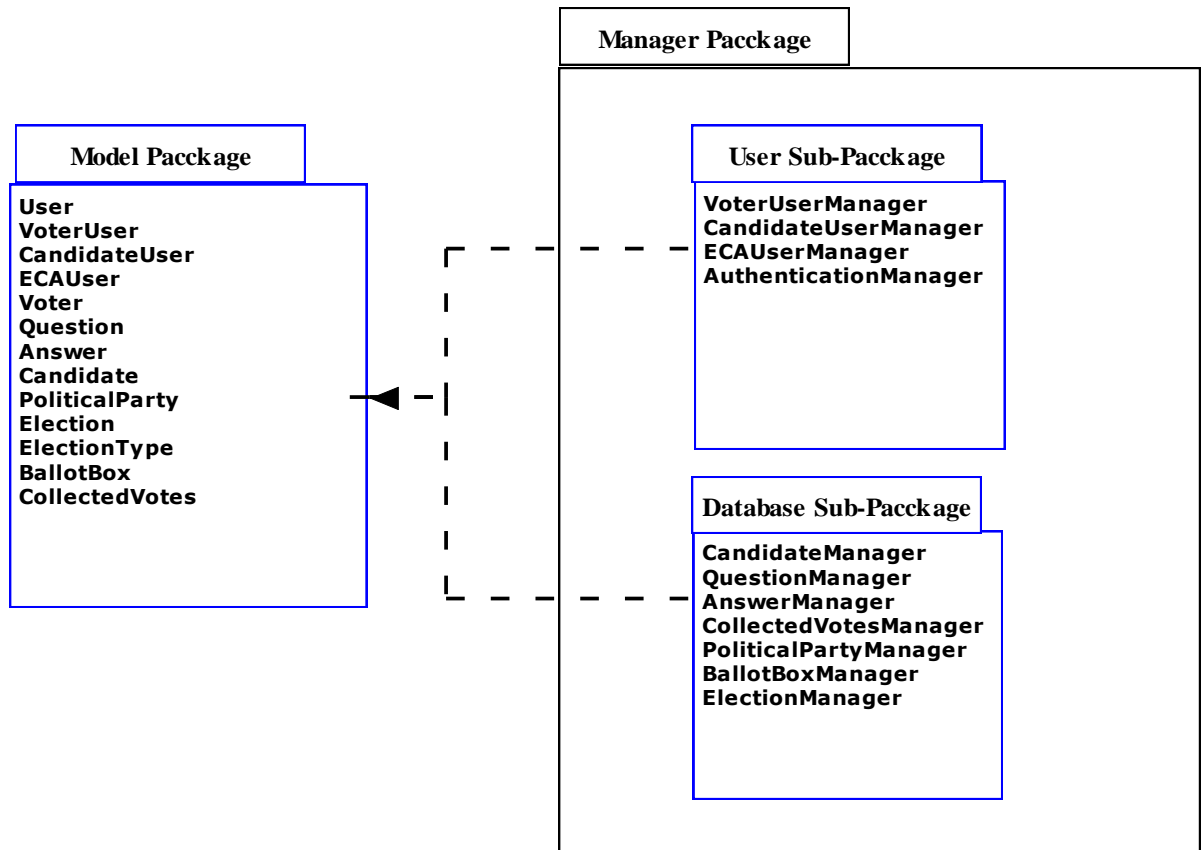


Figure 19: General Overview of Packages

7.1. Model Package

Classification: package

Definition: This package contains classes(as shown in figure 20) that correspond to the database tables for every table in the database there is a corresponding class to represent its attributes once data is queried.

Responsibilities: To provide object oriented presentation of data from the database.

Constraints: It provides no functionalities other than easier data presentation.

Composition: There is no any sub-package of this package.

User/Interactions: It interacts with Manager Package. Manager package uses this package for holding data from database.

Resourcing: There are no any resources that are needed by this package.

Processing: Data storage and presentation.

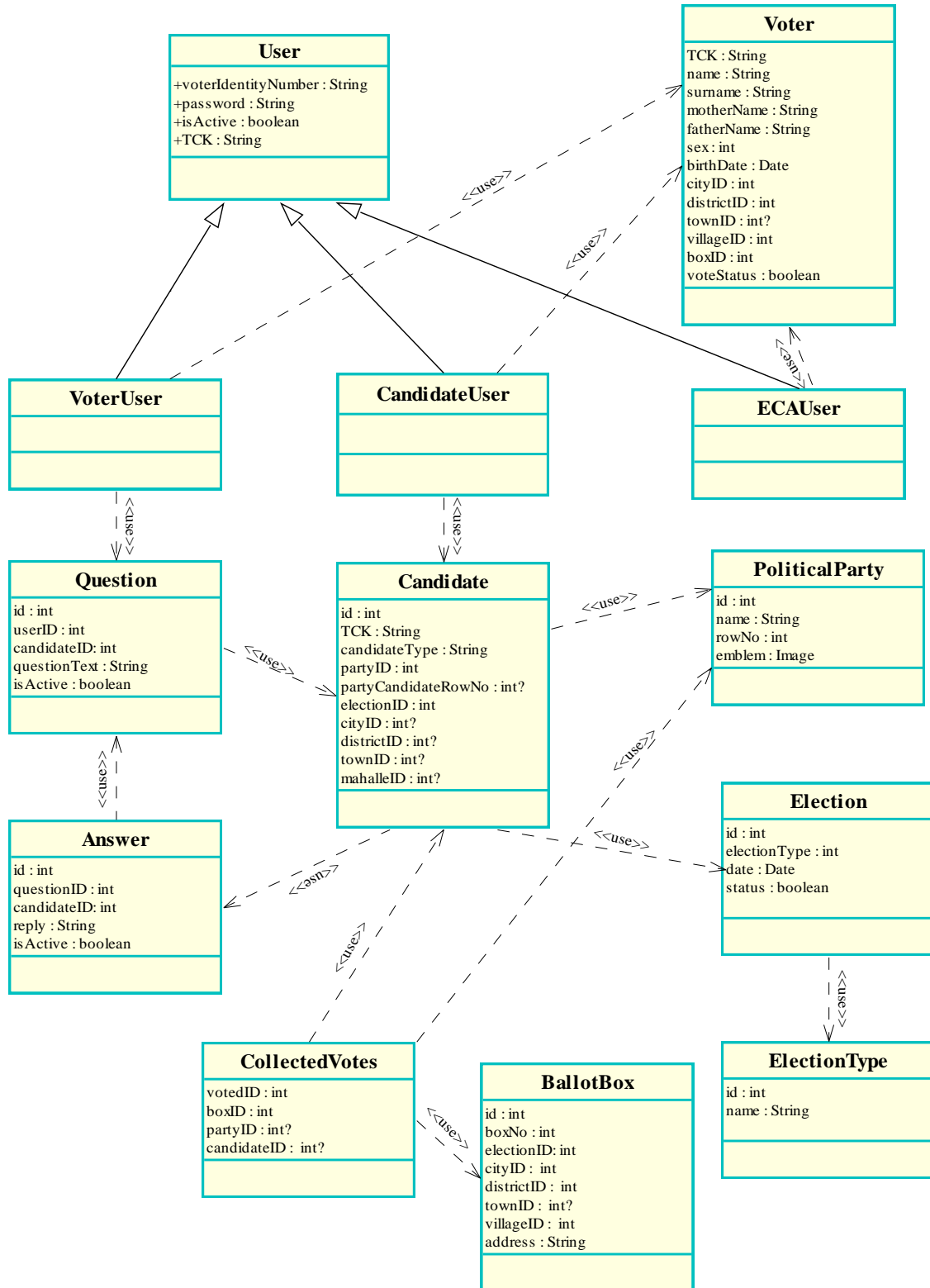


Figure 20: Class Diagram of Model Package

7.2. Controller Package

Classes defined in this package will be used for communicating with the database, i.e. retrieving/updating/deleting/inserting data. The functionality beneath the user interface will be realized by means of this group of classes. This package consists of two sub packages.

7.2.1. User Sub-Package

Classification: Sub-package

Definition: This package contains classes that manage (delete, update, add) their corresponding classes in Model package and entities in database. Every class in Model package has its corresponding controller class. This package contains manager classes for user related data.

Responsibilities: To retrieve data from database and create corresponding Model class and to update/delete data from database.

Constraints: This sub-package provides no more methods other than retrieving, updating and deleting data from database and creating corresponding Model class.

Composition: There is no any sub component.

User/Interactions: It interacts with model classes and database. The interaction between Application Server and Database is done through Controller package.

Resourcing: The Model classes and database tables are the resources of this sub-package.

Processing: Manipulation of database data and instantiation of new class object.

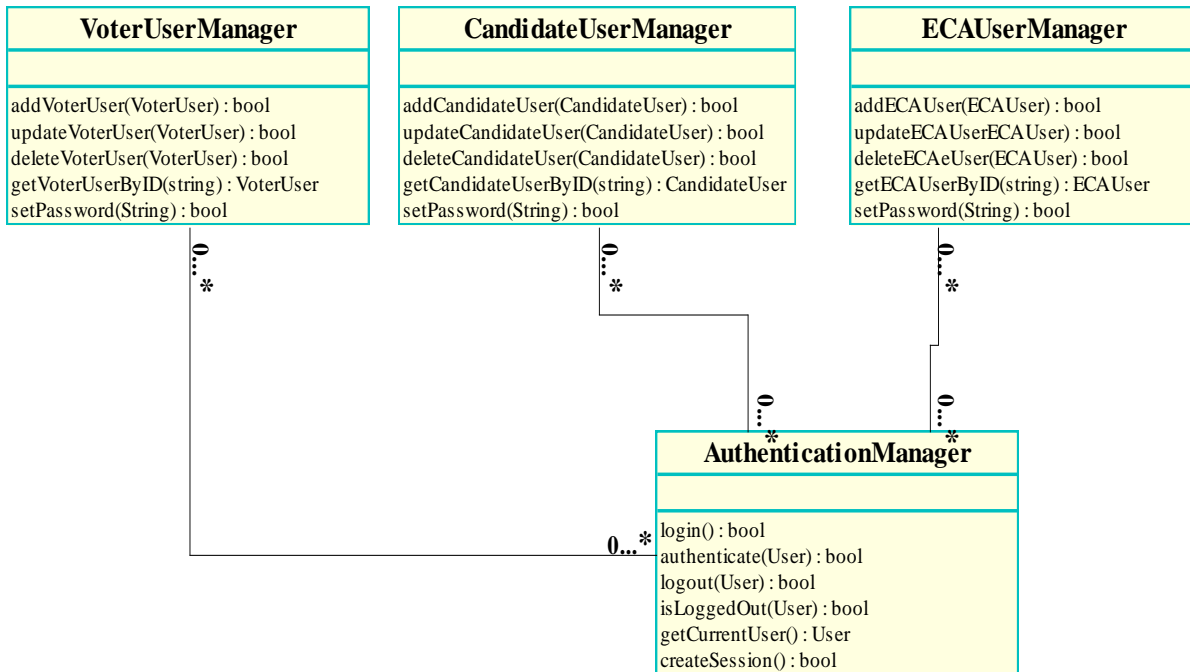


Figure 21: Class Diagram of UserManager Sub-Package

7.2.2. Database Sub-Package

Classification: Sub-package

Definition: This package contains classes that manage (delete, update, add) their corresponding classes in Model package and entities in database. Every class in Model package has its corresponding controller class. This package contains all manager classes that are not related to user.

Responsibilities: To retrieve data from database and create corresponding Model class and to update/delete data from database.

Constraints: This sub-package provides no more methods other than retrieving, updating and deleting data from database and creating corresponding Model class.

Composition: There is no any sub-package of this package.

User/Interactions: It interacts with model classes and database. The interaction between Application Server and Database is done through Controller package.

Resourcing: The Model classes and database tables are the resources of this sub-package.

Processing: Manipulation of database data and instantiation of new class object.

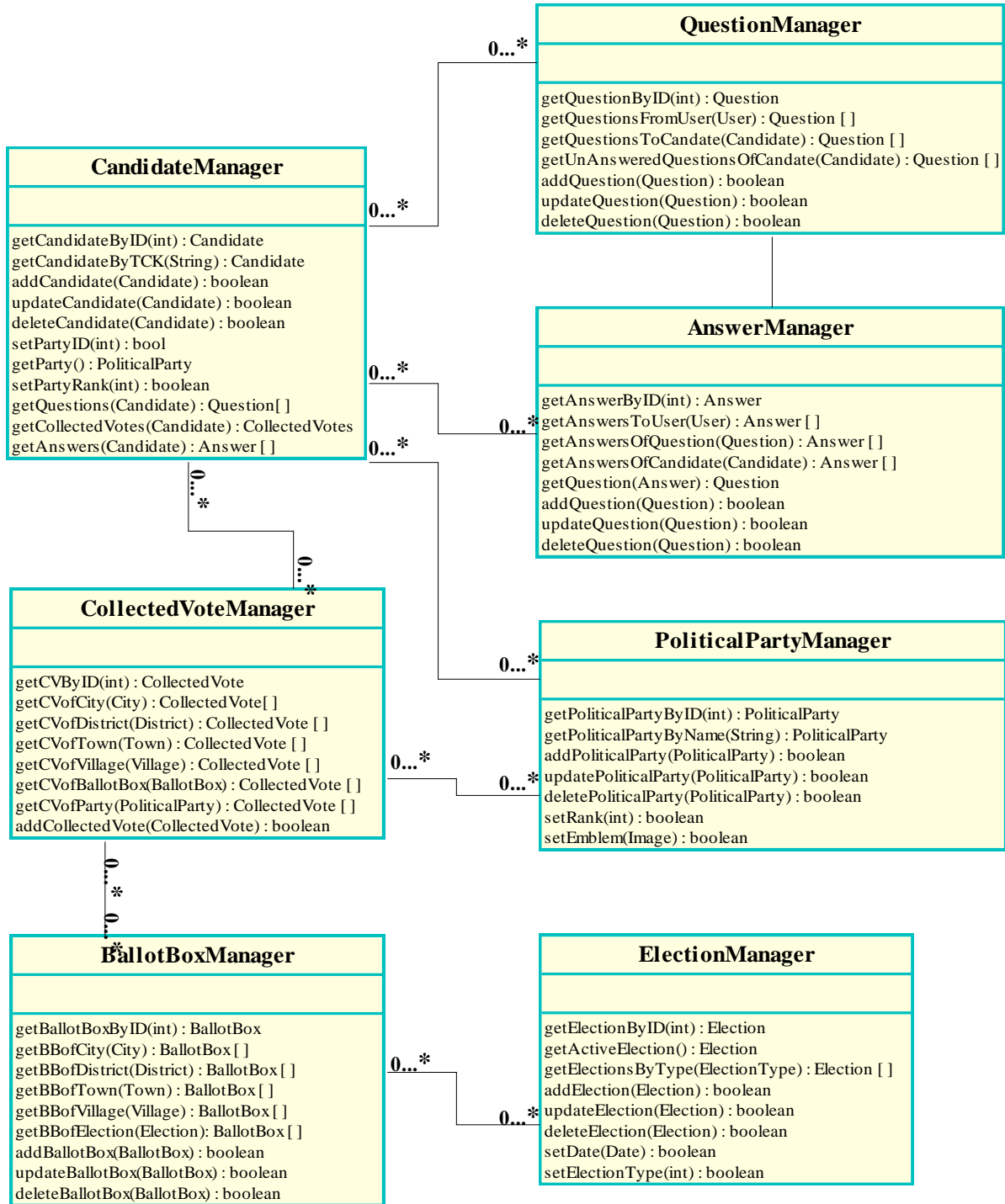


Figure 22: Class Diagram of Database Manager Sub-Package

8. Libraries and Tools

For system design the following tools will be used during the implementation process.

- **UML:** The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems
- **J2EE:** J2EE (Java 2 Platform, Enterprise Edition) is a Java platform designed for the mainframe-scale computing typical of large enterprises. Sun Microsystems (together with industry partners such as IBM) designed J2EE to simplify application development in a thin client tiered environment.
- **Ajax:** Ajax (sometimes called Asynchronous JavaScript and XML) is a way of programming for the Web that gets rid of the hourglass. Data, content, and design are merged together into a seamless whole.[4]
- **DB2:** DB2 is a family of relational database management system (RDBMS) products from IBM that serve a number of different operating system platforms. According to IBM, DB2 leads in terms of database market share and performance.
- **Eclipse:** Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. It is written mostly in Java and can be used to develop applications in Java and, by means of various plug-INS, other programming languages.
- **WebSphere:** WebSphere is a set of Java-based tools from IBM that allows customers to create and manage sophisticated business Web sites. The central WebSphere tool is the WebSphere Application Server (WAS), an application server that a customer can use to connect Web site users with Java applications or servlets.[5]

9. Time Planning (Gantt Chart)

9.1. Term1 Gantt Chart

Number	Task	Start	End	Duration	2010			2011
					October	November	December	January
1	<u>ANALYSIS</u>	10/1/2010	11/20/2010	51				
2	Topic Selection	10/1/2010	10/14/2010	14				
3	Field Research	10/15/2010	10/24/2010	10				
4	Technologies	10/25/2010	11/7/2010	14				
5	Marketing Research	11/8/2010	11/20/2010	13				
6	<u>INITIAL DESIGN</u>	11/20/2010	12/18/2010	29				
7	Meeting With YSK	11/20/2010	11/22/2010	3				
8	Components	11/23/2010	12/1/2010	9				
9	Interfaces	12/2/2010	12/10/2010	9				
10	Data Specification	12/11/2010	12/18/2010	8				
11	<u>DETAILED DESIGN</u>	12/19/2010	1/8/2011	21				
12	User Interface Drafts	12/19/2010	12/24/2010	6				
13	Database Design	12/25/2010	12/29/2010	5				
14	Class Hierarchy	12/30/2010	1/8/2011	10				
15	<u>IMPLENTATION</u>	1/9/2011	1/22/2011	14				
16	Server Design	1/9/2011	1/14/2011	6				
17	Database Design	1/15/2011	1/19/2011	5				
18	Interface Design	1/20/2011	1/22/2011	3				
19	<u>PROTOTYPE DEMOS</u>	1/23/2011	1/24/2011	2				

Figure 23: Gantt Chart of Term1

The Gantt Chart in figure 23 above outlines the activities of the project to be accomplished in the fall term of 2010-2011.

9.2. Term2 Gantt Chart

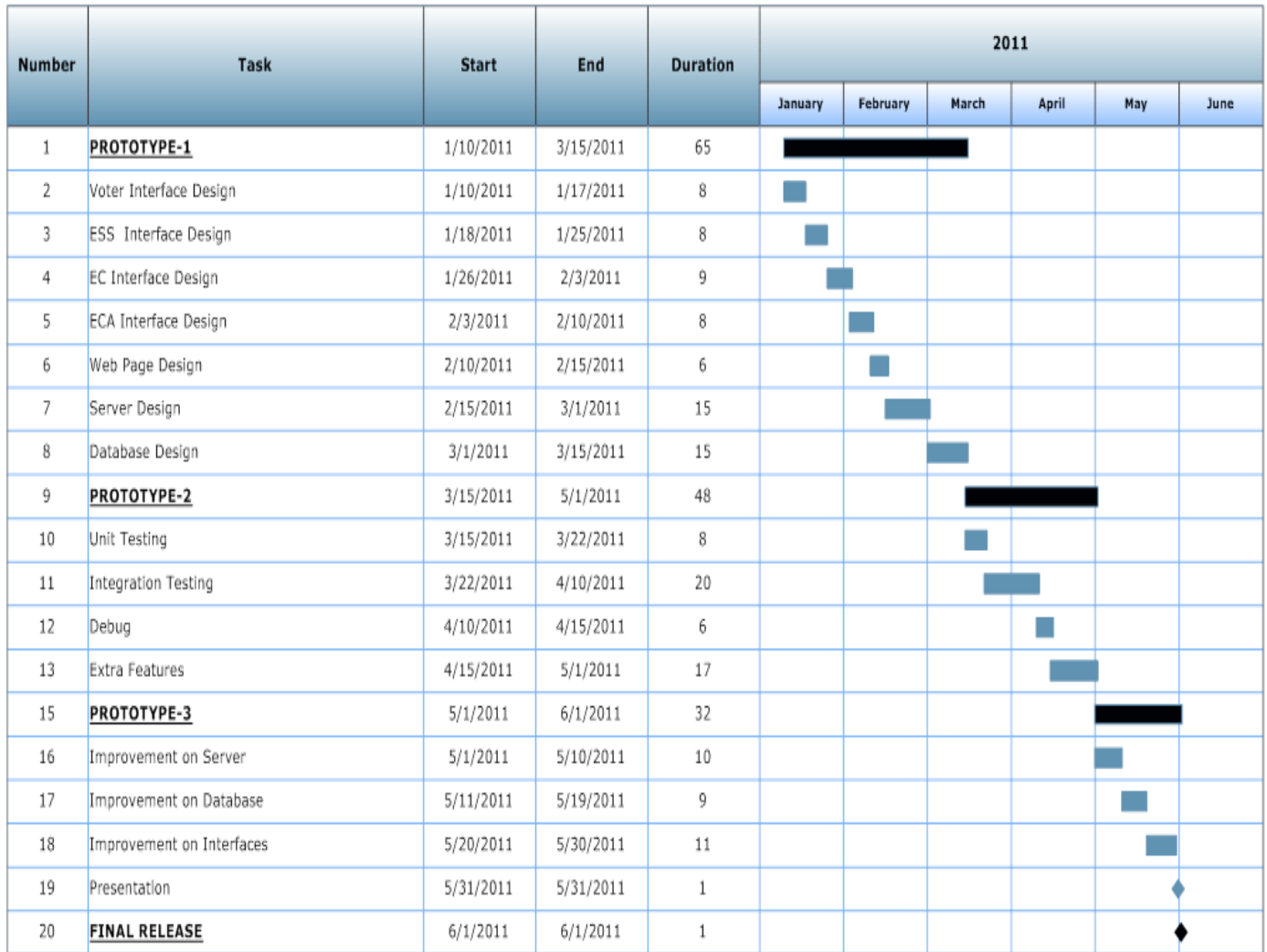


Figure 24: Gantt Chart of Term2

The Gantt Chart in figure 24 above outlines the activities of the project to be accomplished in the spring term of 2010-2011.

10. Conclusion

This document describes the design levels of the ONEV project conducted by iTeam4. The system architecture of the ONEV and data representations is stated through the document. Furthermore, class diagrams with data flow diagrams and design of the user interfaces are showed in the document in detail. Consequently, this document is prepared to conduct better design approaches to ONEV project at implementation.