E-lection

System Design

V0.5

08.12.2017

Abdulwahab A. Sani

Berkay Yılmaz

Can Utku

Recep İlkay Depe

Prepared for

SE301 Software Engineering



Table of Contents

[1. Introduction 1](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996772)

[1.1. Purpose of the System 1](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996773)

[1.2. Design Goals 1](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996774)

[1.3. Definitions, Acronyms, and Abbreviations 1](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996775)

[1.4. References 1](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996776)

[2. Current Software Architecture 1](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996777)

[3. Proposed Software Architecture 1](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996778)

[3.1. Overview 1](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996779)

[3.2. System Decomposition 1](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996780)

[3.3. Hardware Software Mapping 2](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996781)

[3.4. Persistent Data Management 2](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996782)

[3.5. Access Control and Security 2](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996783)

[3.6. Global Software Control 2](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996784)

[3.7. Boundary Conditions 2](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996785)

[4. Subsystem Services 2](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996786)

[5. References 2](file:///C:\Users\Berkay%20YILMAZ\Downloads\2.SDD%20for%20SE301%20(2).docx#_Toc433996787)

SYSTEM DESIGN DOCUMENT[1]

# Introduction

This document starts with an introduction to the architecture and the design goals to be considered. Then it presents the proposed system architecture by describing the subsystem decomposition and the subsystem services. The hardware/software mapping is defined and the management of persistent data is explained. Access control and security issues are addressed. The global software control and boundary controls are described.

E-lection is an online election management, analysis and publishing system that works on web. It is designed as a collection of components that interact with each other. Based on Angular , a framework for the development of general Javascript, The software languages of E-lection are HTML and CSS.

E-lection was designed considering the following design goals:

• Multiple Users

The system should support tasks that are performed by multiple users in concert, supplying

each with the necessary information at the appropriate time.

• Usability

Since the end-user will be using the system while performing work, it is essential for the

system to be intuitive and easy to use. For example the user should be able to understand the procces.

• Scalability

The system must be scalable in terms that it can support many users communicating

or retrieving information at the same time.

• Reusability of Code

To minimize implementation time and improve efficiency, each part of the system has

been designed as a component.

• Understandability

User should be able to understand the components of the system and can use them without any instructions.

• Reliability

System should be reliable which means it should take the stress of the systems and not make any failure.( A measure of success with which the observed behavior of a system confirms to the specification of its behavior)

• Location-Transparency

Server might itself be distributed, but provides a single "logical" service to the user

• High Performance

Client optimized for interactive display-intensive tasks; Server optimized for CPU-intensive operations

• Flexibility

User interface of client supports a variety of end devices (PDA, Handy, laptop, wearable computer)

• Service Portability

Server runs on many operating systems and many networking environments

## Purpose of the System

## Design Goals

## Definitions, Acronyms, and Abbreviations

## References

# Current Software Architecture

On the survey of current software architecture styles for our projects , similarly most of the web applications serving for a purpose like E-lection or not using the client Server architecture style. As technology continues to evolve, so does web application architecture. One such trend is the use of and creation of service-oriented architecture. This is where most of the code for the entire application exists as services. In addition, each has its own HTTP API. As a result, one facet of the code can make a request to another part of the code–which may be running on a different server.

Another trend is a single-page application. This is where web UI is presented through a rich JavaScript application. It then stays in the user’s browser over a variety of interactions. In terms of requests, it uses AJAX or WebSockets for performing asynchronous or synchronous requests to the web server without having to load the page.

The user then gets a more natural experience with limited page load interruptions. At their core, many web applications are built around objects. The objects are stored in tables via an SQL database. Each row in a table has a particular record. So, with relational databases, it is all about relations. You can call on records just by listing the row and column for a target data point.

With the two above trends, web apps are now much better suited for viewing on multiple platforms and multiple devices. Even when most of the code for the apps remain the same, they can still be viewed clearly and easily on a smaller screen. So according to our researches best practices for good web application architecture is going to be like;

# Solves problems consistently and uniformly

# Is as simple as possible

# Supports the latest standards include A/B testing and analytics

# Offers fast response times

# Utilizes security standards to reduce the chance of malicious penetrations

# Does not crash

# Heals itself

# Does not have a single point of failure

# Scales out easily

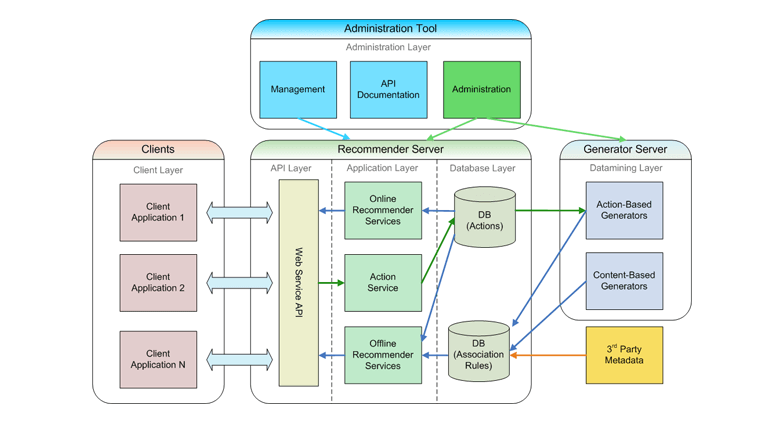
# Allows for easy creation of known data

# Errors logged in a user-friendly way

# Automated deployments

The reason the above factors are necessary is because, with the right attributes. Not to mention, by supporting horizontal and vertical growth, software deployment is much more efficient, user-friendly and reliable

So how the web application architecture works today? We can see the basic working principle by looking at the figure below:



# Proposed Software Architecture

E-lection is going to work as a web application so we decided to use the client-server architecture. The main thing to take away from the last paragraph is that in a web application, there are basically two programs running at the same time:

* The code that lives on the server and responds to HTTP requests.
* The code that lives in the browser and responds to user input.

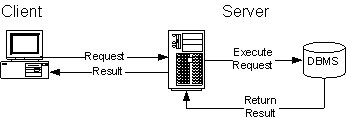
We want to use that architecture because of the following benefits of the both client side and server as shortly shown in down below.

Server-Side Code

* Languages/frameworks include but are not limited to Ruby (Rails), Javascript (Node.js), Python (Django), PHP, C#, and Java; but the list of possibilities is infinite for example we are going to use angular for this project. Any code that can run on a computer and respond to HTTP requests can run a server.
* Stores persistent data (user profiles, instatweets, mybook pages, etc.).
* Cannot be seen by the user (unless something is terribly wrong).
* Can only respond to HTTP requests for a particular URL, not any kind of user input.
* Creates the page that the user finally sees (this is generally only true in web applications that choose to render most of their layouts on the server).

Client-Side Code

* Languages used include: HTML, CSS, and Javascript. Nothing else. But don’t worry, there’s a million frameworks and transpiles-to-[CSS|HTML|JS] languages to choose from (and keep yourself updated on) anyway.
* Parsed by the user’s browser.
* Reacts to user input.
* Can be seen and edited by the user in full.
* Cannot store anything that lasts beyond a page refresh.
* Cannot read files off of a server directly, must communicate via HTTP requests.
* Creates the page that the user finally sees (this is generally only true in single page applications).



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 .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.

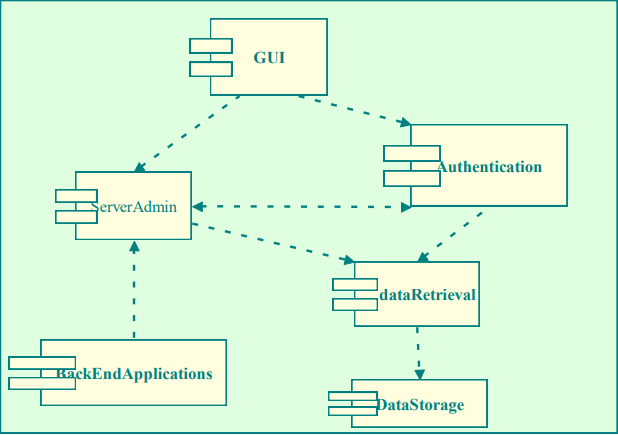
Our system can be subcategorized into six components according to major activities

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).



## Overview

## System Decomposition

## Hardware Software Mapping

## Persistent Data Management

## Access Control and Security

## Global Software Control

The following section will describe the software control implementation. Centralized design will be good for our global software control. Centralized Design;One control object or subsystem ("spider") controls everything.

• Pro: Change in the control structure is very easy

• Con: The single control object is a possible

performance bottleneck

**3.6.1 External control flow (between subsystems)**

• Control flow is distributed within the E-lection system. This means that there is no

central control instance, but each service has its own control flow.

• Services request input (”needs”), wait for it and resume control when it arrives.

• There is a dispatcher (”Service Manager”) coordinating communication between processes and balancing needs and abilities between single services.

• The services use asynchronous callbacks to communicate with each other.

**3.6.2 Concurrent control**

• Service Manager

The E-lection service manager handles communication between single services and balances

abilities and needs of the services. Failure of one service must not affect the service

manager and the other services.

• Multithreading

The service manager uses threads, so that a large number of services is able to use the

service manager simultaneously. The service manager also handles asynchronous events

within the system.

• Callbacks Between Subsystems

The E-lection service manager uses asynchronous callbacks for interprocess communication.

**3.6.3 Internal control (within a single process)**

• Callbacks Between Subsystems

Like the E-lectionservice manager, the services use asynchronous callbacks for interprocess

communication. Every single service uses internal methods to notify the other

subsystems of its own status, its needs and abilities.

• Avoiding Deadlocks

Callbacks must not block the caller. So only status information is updated in the callee.

All other internal work within a single service must be done by other worker threads.

• Worker Threads for Each Service

Each service has an own thread for communication, which communicates with the service

manager and other services.

## Boundary Conditions

# Subsystem Services

# References