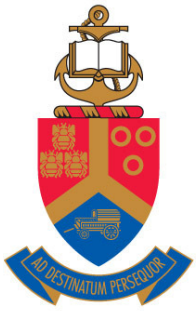

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS 301

Main Project

Architecture Requirements Specification

Team Codex

Andreas du Preez	12207871
Frederick Ehlers	11061112
Azhar Mohungoo	12239799
Gift Sefako	12231097

Contents

1 Architectural Requirements

1.1 Architectural Scope

1. Supply a persistence framework which supports integration with databases and database technologies, and provides secure access to persistent data.
2. Provide a reporting infrastructure, which is: distributed, consistent, accurate, timely, and reliable.
3. Supports system integration with front-end clients, the bare minimum of which should be a desktop web client and an Android mobile client.
4. Provide functionality benchmarking infrastructure.
5. Provide integration with authentication frameworks.
6. Cater for concurrent stateless access to services to at least one hundred clients.
7. Support system flexibility in adding or removing existing modules without necessitating system shutdown.

1.2 Quality Requirements

1. **Convenience:** The system shall allow the voters to cast their votes quickly, in one session, and should not require many special skills or intimidate the voter.
2. **User-Interface:** The system shall provide an easy-to-use user-interface. Also, it shall not disadvantage any candidate while displaying the choices.
3. **Transparency:** Voters should be able to possess a general knowledge and understanding of the voting process.
4. **Accuracy:** The system shall record and count all the votes and shall do so correctly.
5. **Eligibility:** Only authorized voters, who are activated, should be able to vote.
6. **Uniqueness:** No voter should be able to vote more than once for the same poll.
7. **Auditability:** It should be possible to verify that all votes have been correctly accounted for in the final election tally, and there should be reliable and demonstrably authentic election records, in terms of physical, permanent audit trail, which should not reveal the users identity in any manner.
8. **Confirmation:** The voter shall be able to confirm clearly how his vote is being cast, and shall be given a chance to modify his vote before he commits it.
9. **No Over-voting:** The voter shall be prevented from choosing more than one party.

10. **Documentation and Assurance:** The design, implementation, and testing procedures must be well documented so that the voter-confidence in the election process is ensured.

11. **Cost-effectiveness:** Election systems should be affordable and efficient.

12. **Authenticity:** Ensure that the voter must identify himself (with respect to the registration database) to be entitled to vote.

13. **Anonymity:** Ensure that votes must not be associated with voter identity.

14. **System Integrity:** Ensure that the system cannot be re-configured during operation.

15. **Data Integrity:** Ensure that each vote is recorded as intended and cannot be tampered with in any manner, once recorded. Votes should not be modified, forged or deleted without detection.

16. **Privacy:** No one should be able to determine how any individual voted.

17. **Reliability:** Election systems should work robustly, without loss of any votes, even in the face of numerous failures, including failures of voting machines and total loss of network communication. The system shall be developed in a manner that ensures there is no malicious code or bugs.

18. **Availability:** Ensure that system is protected against accidental and malicious denial of service attacks.

19. **Simplicity:** The system shall be designed to be extremely simple, as complexity is the enemy of security.

20. **System Accountability:** Ensure that system operations are logged and audited.

21. **Authentication and Control:** Ensure that those operating and administering the system are authenticated and have strictly controlled functional access on the system.

22. **Distribution of Authority:** The administrative authority shall not rest with a single entity. The authority shall be distributed among multiple administrators, who are known not to collude among themselves.

1.3 Integration and Access Channel Requirements

- The aim of the Electronic Voting system is to allow users to participate in elections remotely, i.e. that they do not have to go into a voting station to cast their vote. The easiest way to achieve this is by providing both a web interface as well as an Android interface (seeing as Android is currently the most widely used operating system for mobiles in South Africa).

- The ultimate goal is that users will be able to easily download and use the Android application from the Google Play Store. The web interface will obviously be accessed from the users prefered browser so we will, whilst developing the system test it currently with multiple web browsers too to ensure that users are never disadvanatged according to their preferred browser.

- Both web interface and the android interfaces will be implemented using a MVC architectural.

- The Electronic Voting system requires the use of Blockchain to maintain elections, votes, voters and candidates as well a generic version of the system which allows participants to partake in surveys.

  Blockchain Explained:

  > The block chain is a shared transperent ledger on which the entire Bitcoin network relies. All confirmed transactions are included in the block chain.
  >
  > Once a transactibon is entered in the blockchain, it can never be erased or modified. Blockchain also allows us to ensure that voters cannot vote more times than allowed. So once a vote has been cast, a voter is sure that their vote was counted for the right candidate.

- The backbone of our server will be the Linux based MultiChain implementation which will allow us to create and manage nodes of the which we will use to implement our own local Blockchain. All of this will be managed from a centralized server which the interfaces can access as in the MVC architectural pattern.

- The backend of the system will be implemented using Java since it intergrates easily with the Spring Framework. The RESTful API will be used with HTTP as a high level protocol as HTTP is widely supported and the RESTful API is supported greatly by the Java and Android libraries.

- PostgreSQL will be used for the database as it provides several external authentication tools and it provides support for the use of JSON(this is also how MultiChain communicates with browsers in the MultiChain Explorer that provides a visual representation of the local blockchain), which will assist with the integration of the server-side communication.

1.4 Architectural Constraints

1. Blockchain

   (a) We're using Multichain to represent the Blockchain component of the Electronic Voting system. The Multchain allows us to manage transactions that happen when a user casts a vote, instantiation and management of all the types of nodes in the chain as well retrieving the balances of all the Party nodes when required.

   (b) The constraint which has been imposed with regards to the Multichain servers is the fact that there is only Linux support for the servers.

2. Ionic

    (a) The Ionic framework allows for develop hybrid mobile applications. The biggest drawback with Ionic since it is not native (Android or IOS) development is that it must support multiple platforms with a single codebase. In the greater scheme of things this is a minor issue since most web browsers are webkit based.

    (b) Hybrid development is best when there wont be intense functionality such as 3D graphics on the application/website itself. So the architectural constraint with regards to the front end is basically that interfaces of the system should be basically be used for simple message passing.

3. Postgre

    (a) Postgre databases have an inherent speed constraint because of its representation of database records as fully instantiated objects in code, this might slow down pre-processing of the system, more especially when it is compared to MySQL.

    (b) Does not support the entire ANSI SQL 92' standard, much less the ANSI SQL 99' standard

4. Android

    (a) The front end of the system will be a website component as well as an android component. The Android component will only be supported by Android 4.3 to more recent releases.

1.5 Architectural Patterns and Styles

Electronic Voting will make use of a combination between the Client-Server and Component-Based architecture styles. The system will be devided into a 'client side' and a 'server side', with a RESTFull web service running on the server side as a communication medium between the two. Component-Based architecture style will be used on the server's side, where each component contains eveything necessary to execute only one desired aspect of the system. We will be using Component-Based architecture to seperate all of the functionalty for Voter, Activator, Party, Admin, Security, Blockchain and Database into their own respective modules. Reasons for Client-Server style:

1. Accessibility:

    (a) EVoting can be accessed form various platforms on a local network or over the Internet.

    (b) Our server can be accessed remotely by administrators or support teams.

2. Upgradation and Scalability:

    (a) Performance changes on the server can be easily made by upgrading the server.

(b) New resources can be added to the server.

3. Plugability:

   (a) We are using a RESTFull web service on the server end.

   (b) Because of this, it allowes us add any number of client platform types (eg iOS, Windows apps, or even other web applications) to integrate with our system if they support our communication medium.

Reasons for using a Component-Based architecture style:

1. Separation of concern (modularization):

   (a) Re-use of the business logic across the platform.

   (b) Multiple modules can be developed without affecting the other modules.

2. Developer specialisation and focus:

   (a) A developer can focus exclusively on one part of the business logic while another developer can focus on other business logic.

   (b) A developer that has more experience and better skill levels in a certain module, can give better input.

   (c) If there is a bug in a module, only that module needs attention.

3. Parallel development by separate teams:

   (a) Business logic developers can build the classes, while the UI developer(s) can involve in designing UI screens simultaneously.

   (b) UI updations can be made without slowing down the business logic process.

4. Multiple view support:

   (a) Due to the separation of the model from the view, the user interface can display multiple views of the same data at the same time.

5. Testing:

   (a) Different bussiness logic modules can be tested independantly of other modules by using intergration and unit testing.

   (b) Each module can expect that the other modules have a good level of correctness, thus expecting a pre-defined output.

1.6 Architectural Tactics and Strategies

Electronic Voting will make use of a combination between the Client-Server and Component-Based architecture styles. The system will be devided into a 'client side' and a 'server side', with a RESTFull web service running on the server side as a communication medium between the two. Component-Based architecture style will be used on the server's side, where each component contains eveything necessary to execute only one

desired aspect of the system. We will be using Component-Based architecture to seperate all of the functionalty for Voter, Activator, Party, Admin, Security, Blockchain and Database into their own respective modules. Reasons for Client-Server style:

1. Accessibility:
   (a) EVoting can be accessed form various platforms on a local network or over the Internet.
   (b) Our server can be accessed remotely by administrators or support teams.
2. Upgradation and Scalability:
   (a) Performance changes on the server can be easily made by upgrading the server.
   (b) New resources can be added to the server.
3. Plugability:
   (a) We are using a RESTFull web service on the server end.
   (b) Because of this, it allowes us add any number of client platform types (eg iOS, Windows apps, or even other web applications) to integrate with our system if they support our communication medium.

Reasons for using a Component-Based architecture style:

1. Separation of concern (modularization):
   (a) Re-use of the business logic across the platform.
   (b) Multiple modules can be developed without affecting the other modules.
2. Developer specialisation and focus:
   (a) A developer can focus exclusively on one part of the business logic while another developer can focus on other business logic.
   (b) A developer that has more experience and better skill levels in a certain module, can give better input.
   (c) If there is a bug in a module, only that module needs attention.
3. Parallel development by separate teams:
   (a) Business logic developers can build the classes, while the UI developer(s) can involve in designing UI screens simultaneously.
   (b) UI updations can be made without slowing down the business logic process.
4. Multiple view support:
   (a) Due to the separation of the model from the view, the user interface can display multiple views of the same data at the same time.
5. Testing:

(a) Different bussiness logic modules can be tested independantly of other modules by using intergration and unit testing.

(b) Each module can expect that the other modules have a good level of correctness, thus expecting a pre-defined output.

1.7 Reference Architectures and Frameworks

1. **JHipster** is a Yeoman generator used to to create Spring Boot and AngularJS projects. This allows us to create a high performance and robust Java stack on the sever with Spring Boot and a modern, attractive front-end with AngularJS and Bootstrap.

2. **Spring Boot** is one of the features of Spring Framework version 4.0 which gives us a way to proive Java applications quickly and simply through an embeded server this means we can expose components such as REST services independently. This framework will prove usefull when deploying the Electronic Voting system.

3. **AngularJS** is a Javascript framework that be added to a HTML web page. AngularJS extends HTML attributes and binds data to HTML. It will be used to build highly interactive web pages for the Electronic Voting system.

4. **Bootsrap** is a HTML, CSS, and Javascript framework for developing responsive mobile-first web sites. We will use when writing the front-end of the web site and Android interface of the Electronic Voting system.

5. **Yeoman** allows us to kickstart a new project. It uses the Yeoman workflow which is a robust client-side stack comprising of tools and frameworks to help developers build web applications.

   The workflow comprises of 3 types of tools.

   (a) **yo:** a scaffolding tool which helps generate and configure the outline of a new application.

   (b) **gulp:** a system build too used to preview and test a project.

   (c) **bower:** which is package management tool, also used for dependency management download and manage scripts.

1.8 Technologies

The technologies that will be implemented in the system to achieve the desired outcome and allow not only successful completion of the project but doing it in the most effective and efficient manner. The technologies are stated with an explaination why they were chosen for this project.

1. **AngularJS:**

   (a) Implements MVC (Model View Controller) which the is Architectural Pattern that is to be implenmented. Angular manages the components and serves as a pipeline to connect them.

(b) Uses HTML to define the interface which is more intuitive and less convoluted than defining the interface in JavaScript. It is also less brittle to reorganize than an interface written in JavaScript.

(c) The data models are POJO (plain old JavaScript objects) which means there is no need for extraneous getter and setter functions.

(d) Behaviour with directives will allow us to invent our own HTML elements by putting all the DOM manipulation code into directives. They are separated out of the MVC application, therefore the MVC application only concerns itself with updating the view with the new data.

(e) Flexibility with filters which will allow the filters to be standalone functions that will be separate from the applicaiton similar to directives, but are only concerned with data transformations such as formatting numbers or reversing the order of an array.

(f) Consise code because of all the aforementioned points will cause less code to be written. No need to write a MVC pipeline. The view is defined just by HTML, models are simpler because there are no extra getter and setter functions. Data binding means there is no need to put data into the data manually. Parrellel team work is possible since the directives are separate from the application code. The filters allow data manipulation on the view level without changing the controllers.

(g) AngularJS can be unit tested by mocking data into the controller and measuring the output and behaviour.

2. **PostgreSQL:**

   (a) Immunity to over-deployment

   (b) Better support than the proprietary vendors

   (c) Stability and reliability

   (d) Extensible

   (e) Cross platform

   (f) Designed for high volume environments

   (g) GUI database design and administration tools

3. **Maven:**

   (a) Making the build process easy

   (b) Providing a uniform build system

   (c) Providing quality project information

   (d) Providing guidelines for best practices development

   (e) Allowing transparent migration to new features

4. **HTML:**

   (a) Supported by both platforms to be implemented web interface and andriod

5. **Bootstrap:**

(a) Easy to use
(b) Will implement the CSS to allow adjustment to all platforms
(c) Speed of development
(d) Responsiveness
(e) Consistency
(f) Customizable
(g) Support

6. **Java:**
   (a) Simple to use with built in functions to assist and make it more simple to use
   (b) Object-Oriented to assist the MVC (Model View Controller) architecture
   (c) Multi-threaded to assist with the scale of the project
   (d) Distrubed which will be needed for the implementation of BlockChain
   (e) Portable as it is platform independent
7. **JHipster:**
   (a) Facilitate the front end of the project
8. **MultiChain:**
   (a) This what will be used to implement the BlockChain