<UNDECIDABLES>

COSBAS Architectural Requirements Documentation

Git: `https://github.com/undecidables/Requirements-Documentation`
Organisation: `https://github.com/undecidables`

The Team:
Elzahn Botha *13033922*
Jason Richard Evans *13032608*
Renette Ros *13007557*
Szymon Ziolkowski *12007367*
Tienie Pritchard *12056741*
Vivian Venter *13238435*

March 2015

Contents

1 Architectural Requirements

1.1 Introduction

The software architecture requirements for the COSBAS system.

1.2 Architectural Scope

1.2.1 Persistence

Persisting system valuable information will take place in a database environment. In the scope of this project an non-relational and noSQL database will be used. MongoDB was chosen as the best solution for the COSBAS project.

1.2.2 Communication

A client-server architecture will be used to communicate. We will use server requests sent from either the web browser or from the rasberry-pi computers.

1.3 Quality Requirements

1.3.1 Reliablity

One of the most important quality requirements for the system is that it should be reliable. The system should therefore produce the correct action/output at any given time when the system is in use.

The system should,

- never **give access** to someone whom should **not have access**.
- never **refuse access** to someone that **have access**.
- **never lock someone in**. If someone gain authorized access to the building the system should allow him/her to exit the building as well.

1.3.2 Security

The system needs to be secure, since personal information is stored on the system. Security as a quality requirement means that the system should prevent malicious actions/attacks.

The system should,

- protect the biometric data and other personal information of all users against unauthorized modification/access.
- protect the biometric data against third party applications.
- prevent loss of information (Personal information, Appointments etc.)

If security as a requirement is not met then the system can be deemed as unimplementable and worthless, since anyone can just gain access to the building.

1.3.3 Scalability And Generic

The system should scalable and generic.

The scalability includes,

- **Functional scalability** - Which means to add functionality to the system with mininal effort from developer(s). The system initially will only have facial recognition and fingerprint scanning as the biometric methods, but it should ideally be expanded to be able use any type of biometric method such as voice recognition for example.
- The system needs to **scale out** (scale horizontally) - Which indicates that the system should be able to add more nodes,that is the system should add more biometric devices, replace existing ones and also add more entry points (entrances/doors), without the loss of performance or any other quality requirement.

2 Architectural Patterns or Styles

2.1 MVC Architectural Pattern

2.1.1 Description

MVC (Model-view-controller) is a software architectural pattern which devides the software application into three interconnected parts, so as to seperate the internal representation from the way the information is represented to the user.

2.1.2 Reason for use

- Client-Server communication
- Reduced code complexity
- Efficient code-reuse
- Decoupled code

2.2 Adapter Design Pattern

2.2.1 Description

The adapter design pattern changes or converts the interface of a class into another interface the client expects. The design pattern makes classes that would normally not be able to work together, interact seamlessly.

2.2.2 Reason for use

- Increased plugability of the system - Because many different biometric access points as well as non-biometric access points will have to interact with the system. This makes it easy for a new type of access point to be added to the system.

2.3 Strategy Design Pattern

2.3.1 Description

The strategy design pattern abstracts the deep implementations of a concrete class, defining a family of algorithms that can be called during runtime. This makes it easier to call functions based on the way the user wants the pogram to react dynamically.

2.3.2 Reason for use

- Increased plugability of the system - We use this for the creation of a calendar service. The use of a strategy design pattern makes it easier to create and initialize an instance of a Calendar object by a chosen service such as Google or Microsoft Outlook.The initial implementation of the COSBAS system only focusses on the Google Calendar Service.

2.4 Factory Method Design Pattern

2.4.1 Description

By making use of the Factory Method design pattern we can create an object without exposing the creation logic to the client and refer to newly created objects using a common interface.

2.4.2 Reason for use

- Optimize code readability and enhance maintainability - The easier code is read and abstracted from the actual implementation, the easier it is to modify or maintain the system to the changing needs of the client and the COSBAS system users.

3 Access and Integration Channels

3.1 Access Channels

3.1.1 Human access channels

This system will be accessible to humans in the followings ways:

- From a thin client (this can be any computer with the client program but the preference is a Raspberry Pi) which will be installed at each entrance/exit of the building through non-intrusive bio-metrics or keypad.
- Humans can access the web client in the following ways:
  - Web Browser (client)
    - A web browser is needed to display the web pages of the website
    - Specifically Firefox, Chrome, Opera, Safari and IE.
  - Physical Devices
    - Since the website will be responsive, any device which has a web browser will be able to connect to the website
    - PCs, Laptops, Tablets and Smartphones

3.1.2 System access channels

The client(can be computer with the client program but in this case it will be a Raspberry Pi) should be able to access the services provided by the system to authenticate a user who would like to enter or exit the building. This will be done through SOAP based web services.

In order to access the web client, the following hardware will be needed:

- Internet Connection (ADSL) (wired or Wi-Fi)
  - A decent internet connection (at least ADSL level speed 384kbs) will be needed to connect to the website (since its being run on a server)
- Ethernet Connection
  - This is necessary if an Internet Connection is not available specifically if the web server is run locally, local computers can connect to it via Ethernet and thus have no need to use an Internet Connection

3.2 Integration Channels

COSBAS will integrate with the following channels:

- The CS LDAP server in order to retrieve login details of the lecturers.
- The postgrad meeting system in order to help with making appointments.
- Any online calendars used, such as Google calendar or Outlook, in order to gain access to the lecturers' calendars.
- The COSBAS-server to process the data and grant or deny access.
- The spring MVC framework, to help with dependency injection and connecting all the componenets together.
- Our MongoDB database for access to the registered users' stored biometric data as well as the appointment information of the staff members.

4 Technologies

4.1 Software

4.1.1 Build System

This project will use the Gradle build system.

Our reasons for choosing the Gradle build system

- Gradle is generally seen as incorporating the best aspects of Maven and Ant
- Gradle's Groovy-base buildscript is a lot shorter than Maven's XML pom
- Gradle 2.5 allows for multiple build-types which will work excellent for our client app that has one version for access requests and one version for registration.

4.1.2 Server

Programming Language: Java
Platform: The server-program needs to be deployed on a Linux server.
Application Server: Jetty.
Reasoning: We chose a Jetty server over other application servers such as TomCat because the Jetty application server is much more third-party API friendly. Jetty is also much more focused on scalability and better performance with regards to serving static content than TomCat. Jetty's focus on multi-connection HTTP and features such as SPDY can significantly reduce page load latencies. The ultimate focus of Jetty is on speed, scalability and reliability, which is very important for our system.

4.1.3 Frameworks and libraries:

- Spring
  - Spring Framework (MVC, IoC, AOP)
    * We chose the Spring framework over other technologies for the following reasons:
    * Dependencies are explicit and evident in JavaBean properties.
    * Spring enhances modularity.
    * Spring provides more readable code.
    * Spring allows us to have loose coupling between different modules. This means that we can inject dependencies at runtime.
    * Spring provides a more flexible way to do dependency injection - dependencies can be configured by XML based schemas or annotations.
    * Spring makes it easier to implement Inversion of Control because we leave the work of dependency injection to the underlying framework.
    * Spring AOP (aspect oriented programming) is implemented purely in Java, and thus we don't need a separate compilation process. Also, because Spring AOP integrates with the underlying Spring framework, we get an advantage in terms of declarative programming for our security and logging purposes.
    * Spring isn't application server dependent.
    * Spring doesn't require any special deployment steps.
    * Spring simplifies Unit Testing because of its loose coupling, thus it makes it very easy to test a class independently (with or without mock objects).
    * Spring is innovative - for example, Spring was the first to bring out CDI, while other technologies like JavaEE took nearly 3 years to do the same thing. Thus, the possibility exists that Spring will keep innovating their technology and bring out new features. For example, Spring implements Spring Data, Spring Social and Spring Mobile.
    * Spring enables POJO programming which enables continuous integration and testability.
    * Spring is open source and has no vendor lock in.
    * Spring has a layered architecture, which means we only have to use what we need and we can leave what we don't.
    * The main reason why we are using it, however, is because of it's outstanding MVC framework. It is highly configurable with strategy interfaces, which is one of the requirements of our project (because we need to be able to use different types of Biometric Access Systems).

  - Spring LDAP
  - Spring Data MongoDB
    * Enabled easy integration with MongoDB in Java.
    * Provides build in operations/functions for the CRUD operations of the database.
    * Has CDI support for the Mongo Repositories that enables the system to have custom query functions.
  - Spring Security
- Thymeleaf template language for HTML and XML
- OpenCV graphics processing library
- JasperReports

4.1.4 Database

COSBAS will have a MongoDB database for persistence of the Biometric Data of the staff members as well as the appointment information of each staff member and the authentication keys that is generated for each appointment.

The reasons for using MongoDB:

- The flexible data model allows us to persist our images and sounds that will be used as the biometric data to authenticate a staff member.
- The scalability that MongoDB provides enable us to reach our scalability quality requirement (Horizontal Scaling) by the means of splitting the database up to run over two servers if we need to accomodate more staff members or visitors in the future.

4.1.5 Client

The client application will be developed in Java and will run on Linux. It will communicate with the server using HTTP requests. Each HTTP request's data will contain the client's Door ID, whether the user is entering or exiting and the types of biometric devices and the captured data.

The client application should run on Linux (Raspbian)

4.1.6 Web Access Channel

- HTML5
- Javascript
- JQuery
- Ajax
- Bootstrap

4.1.7 Testing

We will use JUnit for unit testing.

4.2 Hardware

Our client gave us the task of choosing the equipment we would need in order to build a fully working system that will be implemented at a later stage. Below is a list of equipment we would need for this project and their requirements.

4.2.1 Server

The system will be deployed on an existing server owned by the Department of Computer Science.

4.2.2 Client

The client will be a device installed at the entrance and exit of the buildings where we would want to control access.

The following requirements, listed below, must be met by the client in order to be compatible with the system.

- Capable of running a Linux based operating system.
- Can connect to a network.

COSBAS will be using a **Raspberry PI 2 Model B** as the **client**. This device has a low power consumption, small form factor and is inexpensive. It also meets the requirements listed above.

4.2.3 Camera

The camera will be used to capture the facial features of a user, which will be stored temporarily on the client. The client will then send this data to the server for authentication.

The camera must satisfy the following requirements:

- Has a minimum horizontal resolution of 1920.
- Can connect to the Raspberry PI 2 Model B.
- Is supported by the operating system on the client.

COSBAS will be using the **Logitech C525** as the **camera** since it met the requirements above.

4.2.4 Fingerprint Scanner

Fingerprint scanners will allow us to capture a user's fingerprint which will then be authenticated on the server against our database.

For the fingerprint scanner to be compatible with the system it is must meet the following requirements:

- Has a minimum resolution of 500 pixels per inch, which is the standard for a fingerprint scan.
- Can connect to the Raspberry PI 2 Model B.
- Is supported by the operating system on the client.

COSBAS will be using the **Futronic FS80H** as the **fingerprint scanner** since it met the requirements above.

4.2.5 Keypad

Visitors that do not have biometric access will use the keypad with the temporary access code they received when booking their appointments.

Requirements:

- Can connect to the client.
- Is supported by the operating system on the client.

References

[MongoDB 2015a] MongoDB. 2015. MongoDB. [ONLINE] Available at: https://www.mongodb.org/. [Accessed 30 July 15].

[MongoDB 2015b] MongoDB. 2015. Purpose of Sharding. [ONLINE] Available at: http://docs.mongodb.org/manual/core/sharding-introduction [Accessed 30 July 15].

[Microsoft 2015] Msdn.microsoft.com, (2015). Chapter 16: Quality Attributes. [online] Available at: https://msdn.microsoft.com/en-us/library/ee658094.aspx [Accessed 29 May 2015].

[Spring 2015] Spring. 2015. Spring Data MongoDB. [ONLINE] Available at: http://projects.spring.io/spring-data-mongodb/. [Accessed 30 July 15].

[1] ...