



UNIVERSITEIT VAN PRETORIA  
UNIVERSITY OF PRETORIA  
YUNIBESITHI YA PRETORIA

---

DEPARTMENT OF COMPUTER SCIENCE

COS 301 - SOFTWARE ENGINEERING

---

## OnlyRugby Architectural Requirements

---

*Authors:*

Herman Keuris

Johan van Rooyen

Estian Rosslee

Ivan Henning

Muller Potgieter

*Student number:*

u13037618

u11205131

u12223426

u13008219

u12003672

February 23, 2016

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Architecture Requirements</b>	<b>1</b>
2.1	Architectural Scope . . . . .	1
2.2	Quality Requirements . . . . .	1
2.2.1	Critical: . . . . .	2
2.2.2	Important: . . . . .	2
2.2.3	Nice to have: . . . . .	4
2.3	Architectural Constraints . . . . .	4
<b>3</b>	<b>Architecture Patterns or Styles</b>	<b>5</b>
3.1	Layered Architectural Strategy . . . . .	5
<b>4</b>	<b>Architectural strategies and tactics</b>	<b>6</b>
<b>5</b>	<b>Use of reference architectures and frameworks</b>	<b>8</b>
<b>6</b>	<b>Access and integration channels</b>	<b>8</b>
6.1	Human access channels . . . . .	8
6.2	System access channels . . . . .	8
6.3	Integration Channel Used . . . . .	9
6.3.1	REST - Representational State Transfer . . . . .	9
6.4	Protocols . . . . .	9
6.4.1	HTTP - Hypertext Transfer Protocol . . . . .	9
6.4.2	IP - Internet Protocol . . . . .	9
6.4.3	SMTP - Simple Mail Transfer Protocol . . . . .	9
6.4.4	TSL - Transport Layer Security . . . . .	9
<b>7</b>	<b>Technologies</b>	<b>10</b>
7.1	Programming technologies . . . . .	10
7.1.1	Android Studio . . . . .	10
7.1.2	MySQL . . . . .	10
7.1.3	SQLite . . . . .	10
7.2	Web technologies . . . . .	10
7.2.1	RESTful web service . . . . .	10

# 1 Introduction

This section deals with the software architecture requirements of the OnlyRugby App. This includes:

- The architectural scope.
- Quality requirements.
- The integration and access channel requirements.
- The architectural constraints.
- Architectural patterns and styles used.
- The architectural tactics and strategies used.
- The use of reference architectures and frameworks.
- Access and integration channels.
- Technologies used.

## 2 Architecture Requirements

### 2.1 Architectural Scope

The following responsibilities need to be addressed by the software architecture:

1. To be able to host and provide an execution environment for the services/business logic of the system.
2. To provide an infrastructure that provides a mobile access channel
3. To provide an infrastructure that handles both local and remote persisting and provides access to domain objects.
4. To integrate with the OnlyRugby social platform's MySQL database

### 2.2 Quality Requirements

The following quality requirements are in order of the most important to least important (can be changed as the project progresses).

### **2.2.1 Critical:**

#### **Usability**

The average individual should be able to use the OnlyRugby app without any prior training and not be discouraged from using the system again.

- The OnlyRugby app's interface must be efficient, easy to use, and intuitive to navigate. To accomplish this the interface should be kept minimalistic and logical to allow the user to easily identify and use the services provided by the app.
- Initially only English needs to be supported, however the system must allow for translations to other internationally spoken languages (To allow for an increased user base once the system is made available for regions outside South Africa)
- Usability tests will involve typical users using the OnlyRugby app in a realistic (yet simulated) environment. Tests will be recorded on video as this medium provides task completion time and allows for observation of the user's behaviour, emotions, and difficulties while using the app.

#### **Scalability**

Since the system is planned to be made available for use internationally, it is vital to the success of the system that a large number of users can be accommodated.

- Management of resources available to the app across a range of Android devices needs to scale to ensure that every user of the OnlyRugby app will be presented with a satisfactory experience.
- The system must be able to operate effectively under the load of one user per team for all teams that are registered on the OnlyRugby social platform. (Ball park figure of 1500 after complete deployment in South Africa, 1526 clubs registered with the Rugby Union of South Africa)

### **2.2.2 Important:**

#### **Availability**

Due to the live data capturing requirements of the application and the future planned international release it is important that the OnlyRugby system is always accessible to the user, at the very least during all rugby games registered on the OnlyRugby social platform.

- The OnlyRugby app must be accessible on the multitude of android devices running Android 4.0 and above.
- Data should be captured and stored locally before being pushed to the server in order to avoid data loss should the device being used lose connection to the server

for any reason. This way even if the server is down the core functionality of the app is still usable.

## **Integrability**

The system must be able to communicate with the OnlyRugby social platform.

- Must integrate with the social platform's database concerning teams, players, and scheduled games for retrieval of display information. If communication is not possible (e.g. a loss of data connection) then default/place-holder values (e.g. Team 1, Player number 23) should be used for local storage and replaced with actual data once the connection is restored, but before data is pushed to the server.

## **Maintainability**

The OnlyRugby app system must be maintained to keep the system in a continual safe operating state.

- Should the system go down then it will be rolled back to a previous state in which the system was safe. This lets us recover from an error or a system fault.
- Corrective maintenance is implemented to correct discovered problems in the system after the system has broken. This is the most expensive form of maintenance.
- Preventative maintenance is implemented to correct discovered problems in the system after the system has been implemented but before the system breaks down.

## **Testability**

All the different modules of the OnlyRugby app system must be tested thoroughly before they are integrated and deployed in the final system.

- Each service provided by the system must be testable through a unit test that tests:
  - That the service is provided if all pre-conditions specified in the service contract are met.
  - That all post-conditions specified in the service contract hold true once the service has been provided.

## **Monitorability and Auditability**

User activities should be logged so that they can be audited at a later stage. Each action on the system must be recorded in an audit log that can be viewed and queried at a later date should the necessity to do so arise. Information to be recorded must include:

- The identity of the individual carrying out the action.
- A summary description of the action.
- A timestamp indicating when action was performed.

Other information may be logged which may provide useful statistical information about the system while in use. Examples of such data to be logged might include:

- Location data.
- Failure to communicate with server.
- How long data was kept in local storage before being pushed to the server successfully.
- Miscellaneous errors.

### **2.2.3 Nice to have:**

#### **Security**

All system functionality is accessible to users who can be successfully authenticated against the user database. If connection to the server is not available functionality will be granted in a guest mode and the user will be asked to authenticate prior to the data being pushed to the server upon re-establishment of connection.

- All communication of sensitive data must be done securely through encryption and secure channels.

#### **Performance requirements**

Long delays cause user frustration, may lead users to believe the system is not functioning, or that input has been ignored. To prevent this it is necessary to minimize the response time of user input to the system to ensure that the system works efficiently in real time.

- Performance will be measured with a scale of user frustration.
- Steps on the scale indicate how long a service takes to respond to the user.
- The three steps to be used are 0.1s, 1s, and 10s.

## **2.3 Architectural Constraints**

The following architecture constraints have been imposed by the client:

1. The system must be developed using the following technologies

- Laravel PHP framework
- Google's Material Design
- Android 4.0+
- SQLite

## 3 Architecture Patterns or Styles

### 3.1 Layered Architectural Strategy

The OnlyRugby application will make use of a 4-tier layered pattern, as explained below:

1. Presentation Layer: The Presentation Layer is comprised of two main aspects:
  - (a) Interface: Provides an interface/front-end through which users/clients can access and interact with the Application Layer.
  - (b) Client Data Access: Captures the client's input and passes it on to the Application Layer. Also responsible for validating input (but not for authentication).
2. Application Layer:
  - Provides back-end services of the system (i.e. all functions and forms of data processing/manipulation).
  - Manages access to the web-services layer.
  - Manages client login authentication.
  - Manages persistence to the main database.
  - If the app can not currently reach the database then this layer will provide a space to temporarily store all relevant data which needs to be uploaded to the database at a later time.
3. Web-services Layer: This layer is where the server is situated.
  - Receives the processed client input from the Application Layer and adds it to the database (i.e. allows reading from- and writing to database).
  - Provides other server-side computation/services such as:
    - Conflict resolution when two different clients upload match statistics.
    - Sending emails to clients (for example when registering with OnlyRugby).
4. Data Layer: This layer is where the database is situated.
  - Stores all added information, even when the client is not communicating with the database (i.e. persists data).

Reasons for choosing a layered architecture:

- Complexity is reduced by abstracting and separating a number of well defined layers which are then weakly coupled with each other. This allows the responsibilities of the system to be divided between the different layers and can prevent certain aspects of the system from becoming too dependant on (or needlessly intertwined with) another aspect of the system.
- It allows for the possibility of layers, at times, being re-used across the system (for example the Presentation Layer relies on the Application Layer to verify all input whereas the Web-services Layer relies on the Application Layer to format the input so that it could be stored in the database).
- Separating certain aspects (such as interface and implementation) makes it easier to separate test different parts of the system.
- The reduced dependency of various parts of the system on each other will also improve maintainability as it allows parts of the system to be updated without requiring any unnecessary changes to other parts of the system.

## 4 Architectural strategies and tactics

1. Authorization: The users profile on the application is authorized against the one on the database. JSON web tokens are passed between the application and the database. These tokens are very small, but can contain all the relevant data that needs to be passed and making use of these tokens makes it so that the database does not have to keep of sessions. These tokens are widely used, as they can be easily authorized and encrypted.

The quality requirements addressed:

- Performance: Because the tokens are small, they will be easy and quick to process.
  - Security: By encrypting the tokens and authorizing the user, private information can be protected.
  - Auditability and Testability: Sending the indormation in these tokens will make it easier to see if the correct data was passed.
2. Queuing and scheduling: Queuing and scheduling can be used to maximize the performance of the application and the server. As new requests are received, they are placed into a queue and sequentially processed. This will assure fairness and that requests are processed in order. This will also be used if a user does not have internet access, but wishes to load new statistics. The data will be queued, until it can be sent.

The quality requirements addressed:

- Availability, by optimizing the processing of the requests.
- Reliability and audit ability. This way, the order in which requests are processed can be guaranteed.



- Auditability and Testability: Sending the information in these tokens will make it easier to see if the correct data was passed.
3. Ping / Echo: Ping is a computer network administration software utility used to test the reachability of a host on an Internet Protocol (IP) network and to measure the round-trip time for messages sent from the originating host to a destination computer and back. By having the application send these pings on a regular basis, perhaps daily and when the user logs in, to the database, it can ensure that it still has access to the central server and that it is up to date with both the latest statistics, as well as the newest version of the application.

The quality requirements addressed:

- Security. As it can ensure that all its security policies are up to date and that it can report any faults/errors.
  - Availability. It will inform the user if it was unable to reach the server.
  - Reliability. Any issues that arise may be reported and it ensures that a connection is available.
  - Monitor-ability and Audibility. It can log faults detected.
  - Testability. It will allow us to test the connection to the server and that it can make use of said connection.
4. Message integrity: We will be exchanging JSON files between the server and the application. These files will include any new data that either party may need. By including a checksum, these files can be checked for any errors that may have occurred during transportation. Should it find a checksum error, a request will be sent and a new JSON file will be transmitted.

The quality requirements addressed:

- Security. This will help to protect the system from corrupted data that may hamper the application and/or server's functionality.
  - Reliability: This will help ensure that erroneous data is not used.
  - Monitorability and Auditability: It will allow us to test if the application and/or server constantly sends erroneous data and address the issue.
5. Multi-Threading: By using multiple threads, the application can process a larger amount of data simultaneously. While the user may be busy updating a game's statistics, the application will assign another thread to process any incoming requests, improving the program's overall performance.

The quality requirements addressed:

- Performance. It will be able to process more data, simultaneously.
- Flexibility. By creating new threads, the program will be able to handle multiple tasks more easily.
- Usability. By keeping the user unaware of any background actions that are taking place, it will be easier for them to use the app.

## 5 Use of reference architectures and frameworks

The reference architecture that will be used is Java Platform Enterprise Edition. This architecture was chosen because our system is based on a layered architecture. The framework that will be used is the Laravel PHP Framework as we will have a RESTful web service.

## 6 Access and integration channels

### 6.1 Human access channels

- Must be able to access services through the OnlyRugby application installed on tablets and phones running Android 4.0+
- The user can easily use the OnlyRugby application to communicate with the server (when registering, logging on and off of their profiles) and through the server they can communicate with the MySQL database.
- Quality requirements addressed:
  - Availability
  - Usability

### 6.2 System access channels

The OnlyRugbyApp system can only be directly accessed through a web page (i.e. using http) using the REST API provided by the server.

Quality requirements addressed:

- Security

### 6.3 Integration Channel Used

#### 6.3.1 REST - Representational State Transfer

- Uses standard HTTP and thus simpler to use.
- Allows different data formats where as SOAP only allows XML.
- Has JSON support (faster parsing than XML).
- Better performance and scalability with the ability to cache reads.
- Protocol Independent, can use any protocol which has a standardised Uniform Resource Identifier (URI) scheme.
- Quality requirements addressed:

- Integrability
- Maintainability

## **6.4 Protocols**

### **6.4.1 HTTP - Hypertext Transfer Protocol**

- Used to respond to requests and transfer data

### **6.4.2 IP - Internet Protocol**

- Allows communications between client and server
- In charge of sending, receiving and addressing data packets.

### **6.4.3 SMTP - Simple Mail Transfer Protocol**

- Used to send emails (useful for registration confirmation)
- MIME (Multi-purpose Internet Mail Extensions) which allows SMTP to send multimedia files.

### **6.4.4 TLS - Transport Layer Security**

- Alternative to SSL.
- Newer and more secure version of SSL.

## **7 Technologies**

### **7.1 Programming technologies**

#### **7.1.1 Android Studio**

Android studio will be used as the IDE and the app will be backwards compatible from Android 4.0.

#### **7.1.2 MySQL**

MySQL will be used for the web server database.

#### **7.1.3 SQLite**

SQLite will be used for the local on device database.

## **7.2 Web technologies**

### **7.2.1 RESTful web service**

Laravel PHP framework will be used to create a RESTFul web api to interact with the web server database and provide data persistence