TMVS26 – SPA17 Team B

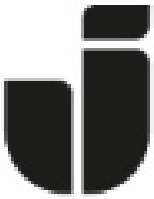
MECALS

ARCHITECTURE DOCUMENT

School of Engineering, Jönköping University, Sweden

Ajay Kumar Rajamundry, Benjamin Couillaud, Onete Bogdan Gabriel, Théo Ardouin

March 2017



JÖNKÖPING UNIVERSITY

School of Engineering

MECALS Teams

MECALS is been developed by 18 members dived into 5 teams with specific roles.

Team A: ARCHITECT's

ARDOUIN THÉO	arth163k@student.ju.se
COUILLAUD BENJAMIN	cobe16zc@student.ju.se
ONETE BOGDAN GABRIEL	onbo1693@student.ju.se
RAJAMUNDRY AJAY KUMAR	raaj1617@student.ju.se

Team B: FRONT END DEVELOPERS

QASIM MUHAMMAD	qamu1678@student.ju.se
DESCHAMPS MAXIME	dema16n5@student.ju.se
DARCHY OLIVIER	daol166b@student.ju.se
AL-GHAREEB MEELAD	alme1324@student.ju.se

Team c: BACK END DEVELOPERS

CARLSSON FREDRIK	cafr1392@student.ju.se
ERIKSSON KLAS-GÖRAN	erkl1213@student.ju.se
MILOX Thomas	mith16j2@student.ju.se
PRENAT HUGO	prhu16e4@student.ju.se

Team D: TESTERS

DARSI VENKATA NARAYANA	dave1648@student.ju.se
KASPRZAK MILAN	kami16cm@student.ju.se
LACOUR ARTHUR	laar16ze@student.ju.se
NIRMAL KUMAR NIRMAL KUMAR	nini1616@student.ju.se

Team E: MANAGEMENT TEAM

NEUBAUER ADRIAN	nead1657@student.ju.se
SARDA JEAN	saje168j@student.ju.se

Revision History

NOTE: *The revision history cycle begins once changes or enhancements are requested after the initial version of the Software Architecture Document has been completed.*

Date	Version	Description	Author
17/02/2017	0.1	Initial version of Front End Software Architectural Description	Ajay Kumar Rajamundry
07/03/2017	0.2	Added all subparts expected. Added Use Cases. Added Deployment and Context Diagrams.	Ajay Kumar Rajamundry
09/03/2017	0.3	Updated Context and Deployment Diagrams. Patched Architectural Design Decisions and Use Cases. Updated project based on recent changes.	Benjamin Couillaud Onete Bogdan Gabriel Théo Ardouin
11/03/2017	0.4	Added Backend Design Decisions, Architecture Diagrams, and explanations	Ajay Kumar Rajamundry Onete Bogdan Gabriel (with help from current Team C)
12/03/2017	0.5	Added Frontend Class & Sequence Diagram, and Backend Architectural Representation	Onete Bogdan Gabriel (with help from current Team B & C)
13/03/2017	0.6	Corrected Fields, added Backend Class & Sequence, Use Case Diagram, NARMS Documentation, Edited layout	Benjamin Couillaud Onete Bogdan Gabriel Théo Ardouin (with help from current Team B)
15/03/2017	0.7	Added Testing Environment, and patches	Onete Bogdan Gabriel (with help from current team Darter)
16/03/2017	0.8	Added Team List, updated References and Context Diagrams, and added patches	Ajay Kumar Rajamundry Onete Bogdan Gabriel

Table of Contents

Table of Figures

Software Architecture Document

1. Introduction

This Software Architectural document provides a overview and explains the whole architecture of MECALS Front End Development. It is gives description of the purpose, scope, definitions, acronyms, abbreviations, references, goals of the architecture, different views of the modules and documents that define the domain and technical standards in detail to maintain consistency when project is under maintenance or while updating the system. Development/Maintenance team will have a better understand the problems which may rise and how to solve them.

1.1. Objective

The purpose of this Software Architecture Document (SAD) is to describe the architecture of the MECALS project. The architecture will contain diagrams of various types, depending on the views they are based on, as well as explanations to justify the architectural choices, in order to fulfil all requirements. Additionally, technical notes will be attached as well, all of this in order for the document to act as a guide in order to fully comprehend, develop, deploy, setup and maintain MECALS.

To give a short understanding of the software as accurately as possible, this document is based on the "4+1" model view of architecture [Reference for 4+1].

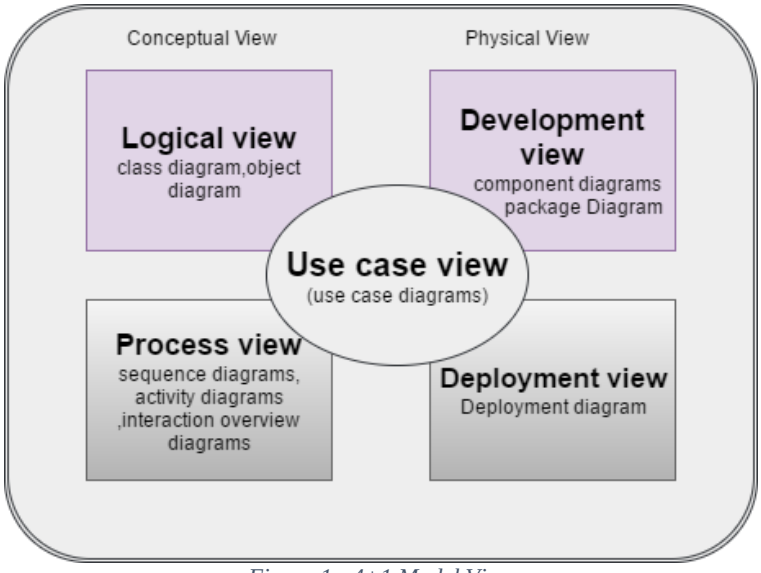


Figure 1 - 4+1 Model View

1.2. Scope

The scope of this SAD is to give the architecture details of the MECALS Front End Development by Team B of Software Product Architecture, and the MECALS Back End Development by Team C of Software Product Architecture.

1.3. Document Description

Title	MECALS: Architecture Document
Creation Date	--/03/2017
Publication Date	--/03/2017
Product Owners	
Authors	Ajay Kumar Rajamundry raaj1617@student.ju.se
	Benjamin Couillaud cobe16zc@student.ju.se
	Onete Bogdan Gabriel onbo1693@student.ju.se
	Théo Ardouin arth163k@student.ju.se
Subject	Architecture Document

1.4. Definitions, Acronyms, Stakeholders and Abbreviations

Stakeholders					
Customers	**Developers**	**Users**	**Testers**	**Communicators**	**Production Engineers**
SAMO	Team A Team B Team C	ATCOs ASM(System) NARMS(System)	Team Darter	Team A Team E	Team Darter Team A

Abbreviation	Meaning
ASM	Air Space Management
ATCO	Air Traffic Controller
CALS	Controller Automated Login System
MECALS	Mobile Enhanced Controller Automated Login System
NARMS	New ATOCs Reporting & Monitoring System
SAD	Software Architecture Document
SAFAPS	Stress And Fatigue Audit and Prediction Service
SAMO	State Airspace Management Organization

1.5. Overview

The reason for MECALS (android application) is to replace card readers ,which log events to the CALS server. The reason behind this is that the card readers are aging and some of them have stopped functioning correctly, in addition to them no longer being produced.
The android application will scan a QR code to get the IP address of the workstation so that an ATCO can sign in and log events. After that, the logs will be sent to the MECALS server.

Technology	Purpose
Android Studio (java)	Develop the phone application.
Nodejs	Develop the back-end and network communication.
Travis	Continuous integration.
Github	Version control.
Slack	Communication between team members.
Trello	Track progress, see individual activity, manage tasks & the product backlog.
Google Drive	Manage documents.
Astah Community	Create views for architectural documentation

2. Architectural Representation

2.1. Mobile Application – Front End

MECALS is a mobile application developed for the Android platform, which sends requests to the MECALS servers. It includes a Restful API that handles login and authorization. This API can be called by the mobile application via certain HTTP clients, namely ATCOs. Besides sending data from the user, MECALS also displays data wrapped in a user interface displayed on the screen, which is human readable.
The MECALS API is compliant with the 4+1 architecture.

2.2. Server – Back End

The Backend has 3 endpoints:

2.2.1. Login

This endpoint allows you to login as an employee

HTTP Request: POST http://193.10.30.169/login

Query Parameters: (Parameters you must pass in the form of a JSON body)

Property	Type	Required	Description
email	string	true	Email of the user you want to login with
password	string	true	Password of the user you want to login with

Return Codes:

Code	Description
200	Success – OK
401	Unauthorized – Your credentials are wrong

2.2.2. Logout

This endpoint allows you to logout of the workstation
HTTP Request: POST http://193.10.30.169/logout

Query Parameters: (Parameters you must pass in the form of a JSON body)

Property	Type	Required	Description
user_id	string	true	Id of the user you want to logout
workstation_id	string	true	Id of the workstation where the user is logged in
facility_pub_id	string	true	Id of the facility where the user is logged in

Return Codes:

Code	Description
200	Success – OK
401	Unauthorized

Global Architecture

The architecture of MECALS presents distinct views, with their own audience and purpose.

This document details the architecture using the views defined in the "4+1" model [KRU41], but using the RUP naming convention. The views used to document the MECALS application are:

- **Use Case view**

  Audience: all the stakeholders of the system, including the end-users.
  Area: describes the set of scenarios and/or use cases that represent some significant, central functionality of the system. Describes the actors and use cases for the system, this view presents the needs of the user and is elaborated further at the design level to describe discrete flows and constraints in more detail. This domain vocabulary is independent of any processing model or representational syntax (i.e. XML).
  Related Artifacts: Use-Case Model, Use-Case documents

- **Logical view**

  Audience: Designers.
  Area: Functional Requirements: describes the design's object model. Also describes the most important use-case realizations and business requirements of the system.
  Related Artifacts: Design model

- **Process view**

  Audience: Integrators.
  Area: Non-functional requirements: describes the design's concurrency and synchronization aspects.
  Related Artifacts: (no specific artifact).

- **Module Decomposition view**

Audience: Programmers.
Area: Software components: describes the modules and subsystems of the application.
Related Artifacts: Implementation model, components

- **Deployment view**

  Audience: Deployment managers.
  Area: Topology: describes the mapping of the software onto the hardware and shows the system's distributed aspects. Describes potential deployment structures, by including known and anticipated deployment scenarios in the architecture we allow the implementers to make certain assumptions on network performance, system interaction and so forth.
  Related Artifacts: Deployment model.

3. Architectural Design Decisions

3.1. Server side

Node.js – JavaScript framework for runtime events.
Reason for node.js: Very simple to use language made for backend software, using this vastly reduce amount of code and work required compared to for example C#.

Express – Web interface built for node.js.
Reason for express: As node.js is used this was an obvious choice as it was built for it.

Operating system – Does not matter, the server can be deployed on either Windows or Unix. Since our equipment is mostly based on Windows , it was chosen as the OS.

3.2. Client Side

Programming Language: Java
MECALS is coded in Java. This language has the following advantages:
Object-Oriented: allows modularization and reusability of code
Distributed: designed to make distributed computing easy with the networking capability that is inherently integrated into it
Safe: The Java language, compiler, interpreter, and runtime environment were each developed with safety in mind.
Robust: emphasizes on early checking for possible errors, as Java compilers are able to detect many problems that would first show up during execution time in other languages.
Multithreaded: perform several tasks simultaneously within a program, through the use of asynchronous HTTP requests.

Besides these aspect, Java was also chosen since it is native to Android. Java and Android were also a viable choice because there is an abundance of documentation for both of them available.

TODO in future development:
Service: Bluetooth

The workstation communicates with the mobile application via Bluetooth. It's behavior must be as follows:

- The Bluetooth **MUST** be enabled on the phone.
- The Bluetooth **MUST** be paired with the server as soon as the login request is accepted.
- On a fair distance, when the signal is lower than a specific value (*empirically* determined), the application **SHOULD** send to the user a warning message.
- On a long distance, when the signal is lower than a specific value (*empirically* determined), the pairing between the server and the application MUST be stopped, and a logout request **MUST** be send.

Figure 2 - Bluetooth Signal

4. Testing Environment

4.1. Configuring the auto deploy to the servers from Travis:

A simple ftp was installed on the server which is going to run the backend (vsftpd)
A new user was created on the server, so that Travis can use it to upload the successful build.
The username is: "travis_deploy", the password is: "tr4v1sd3pl0y"
Travis couldn't upload to the server through FTP at first.
One solution could be to use the auto deployment feature of Travis to the Heroku platform.
There was a port problem on the server. After contacting the system administrator, the correct ports have been open and the FTP was working correctly.
A new utility was installed on the server, "incron". It has the ability to watch the filesystem for a specific event. It was configured so that every time a new version of the backend is uploaded on the server, it stops the previous version and start the new one, using a shell script.

4.2. Android Application Continuous Integration

While waiting for the ports to be open for the FTP server, work was started on the Android App continuous integration.
The .travis.yml file was added on the App repository on a new branch at first, adjusting it so it can work with the correct SDK.
There were problems in finding the correct versions of all the different elements, but is currently up and running on the master branch.
Every push on the application repository triggers a build and notifies us if it passed.
Travis could also start an emulator and run the application, we didn't need this functionality at this moment.

4.3. MECALS: Android Application Test Scenario

1.

Title	Start
Description	Starting the application
Steps	Press the application icon
Result	The application starts

2.

Title	Camera permission
Description	Verify that the application can access the camera of the device
Steps	At the first start, the application will ask the user permission to use the camera
Result	The user says “yes”, and the application can now freely use the camera

3.

Title	QR Code scanning
Description	Verify that the application can correctly scan a given QR Code
Steps	Place a QR Code in front of the camera and center it
Result	The application read the QR Code and proceed to the next step

4.

Title	Login
Description	Having the QR Code scanned, the application now try to login
Steps	Send the login credentials to the server to login
Result	With correct credentials, the server returns “OK” and information. With wrong credentials, the server returns an error.

5.

Title	Logout
Description	Logout of the application
Steps	Press the logout button
Result	The server return a message saying that the user is correctly logged out.

5. Use-Case View

Use case functionality diagram below describes how design elements provide the functionalities identified in the significant use-cases. Use cases are displayed as functionalities for the system. Functionality may enclose more than one use-case.

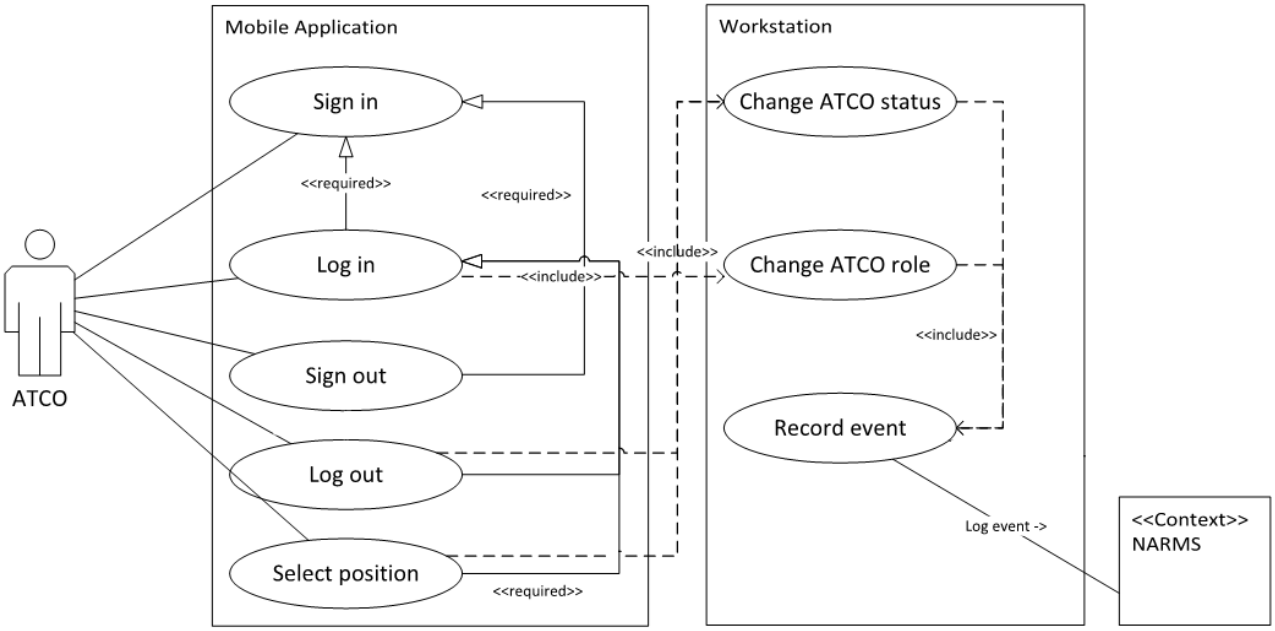


Figure 3- Use Case View

Definitions:

- ATCO: the actor interacting with MECALS
- Mobile application: Interaction point for the ATCO. Application deployed on mobile device, allows the actor to interact with MECALS.
- Backend: contains business logic. Deployed on workstation.

Hard Requirements:

- The mobile applications uses Bluetooth for proximity checks, to check if the mobile device is in range to the workstation.
- "Custom Codes" for login with workstations.
- The mobile application needs to run on mobile devices with Android API 21 and higher
- The Backend can run on any Windows or Unix operating system.
- The ATCO needs to authenticate with the mobile application.

Use Cases:

5.1. Log in ATCO to workstation

The ATCO logs in to the workstation. The system captures the log in event and stores it. This can lead to a status change for other ATCOs currently working at the same workstation and/or in the same control room.

5.1.1. Pre-Conditions

- The ATCO is signed in and authenticated with the mobile application.
- The ATCO is in Bluetooth range of the workstation.

5.1.2. Triggering Events

- The ATCO scans the “custom code” on a workstation.

5.1.3. Post-Conditions

- The ATCO is logged in the workstation.

5.2. Log out ATCO from workstation

The ATCO ‘logs out’ from the workstation when he leaves the workstation. The system captures the log out event and stores it. This can lead to a status change for other ATCOs currently working at the same workstation and/or in the same control room. The Logout event triggers if either the ATCO manually logs out or the mobile device leaves the proximity.

5.2.1. Pre-Conditions

- The ATCO is logged into a workstation.

5.2.2. Triggering Events

- The ATCO presses the logout button.
- The ATCO leaves the vicinity of the workstation. (Bluetooth connection interrupts; TODO)

5.2.3. Post-Conditions

- The ATCO is logged out.

5.3. Sign in ATCO to ASM

The cards and card readers which were used for user-authentication will be replaced, therefor the need for a new authentication process arises. [Technical details/flow yet to be determined]

5.3.1. Pre-Conditions

- The ATCO is not signed in.

5.3.2. Triggering Events

- The ATCO starts the mobile application.

5.3.3. Flow of Events

1. The ATCO starts the mobile application.
2. The mobile application asks the ATCO for credentials.
3. The ATCO types in his credentials.
4. The mobile application tries to authenticate the user with the ASM system.
5. If authentication fails, back to 2
6. If authentication succeeds, continue.

5.3.4. Post-Conditions

- The ATCO is signed in with the ASM facility and can now log-in at workstations.

5.4. Sign out ATCO from ASM

The ATCO needs to be able to sign out when he leaves the ASM facility.

5.4.1. Pre-Conditions

- The ATCO is signed in.

5.4.2. Triggering Events

- The ATCO presses the logout button.

5.4.3. Flow of Events

- If the ATCO is still logged in at one or more workstations, display a warning message.
- ALT: automatically logout the ATCO.
- ALT2: do not allow to sign-out if still logged in.

5.4.4. Post-Conditions

- The ATCO is signed out.
- The ATCO is not logged in at any workstation.

5.5. Select workstation position

The ATCO selects his position at the workstation. See specification documents for details (TODO).

5.5.1. Pre-Conditions

- The ATCO is logged in.

5.5.2. Triggering Events

- The ATCO selects his workstation position.

5.5.3. Post-Conditions

- The ATCOs status gets updated.

5.6. Change status for ATCO

The ATCO changes his status at the workstation. See specification documents for details (TODO).

5.6.1. Pre-Conditions

- The ATCO is logged in.

5.6.2. Triggering Events

- The ATCO selected his workstation position.
- The status of another ATCO at the workstation got changed.
- The ATCO or another ATCO at the workstation logged out.

5.6.3. Post-Conditions

- The status of all ATCOs at the workstation got updated.

5.7. Change role for ATCO

The ATCO role is determined by the ASM system. Documentation missing about WHEN roles change/who decides on what parameters etc. (TODO)

5.7.1. Pre-Conditions

- The ATCO is logged in.

5.7.2. Triggering Events

- Unforeseen situations

5.7.3. Post-Conditions

- The ATCO role updated based on new conditions.

5.8. Record Event in MECALS and NARMS

Sends a log event to both MECALS and NARMS.

5.8.1. Pre-Conditions

- none

5.8.2. Triggering Events

- The ATCO logs in and the role gets updated.
- The ATCO logs out and the role gets updated.
- The ATCO selects a position and the status gets updated.

5.8.3. Post-Conditions

- The ATCO role updated based on new conditions.

6. Logical View

Frontend Class Diagram:

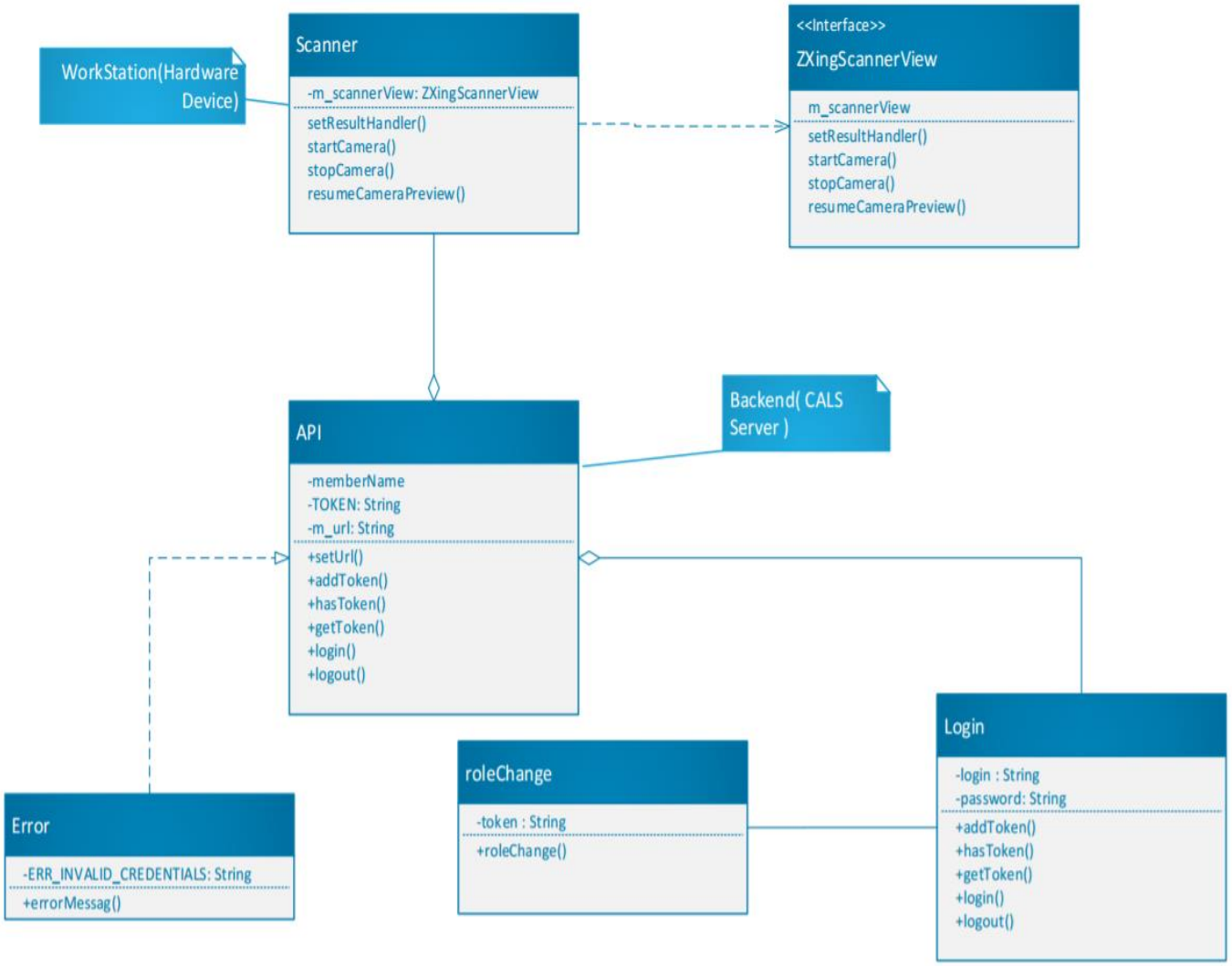


Figure 4 - Frontend Class Diagram

Backend Class Diagram

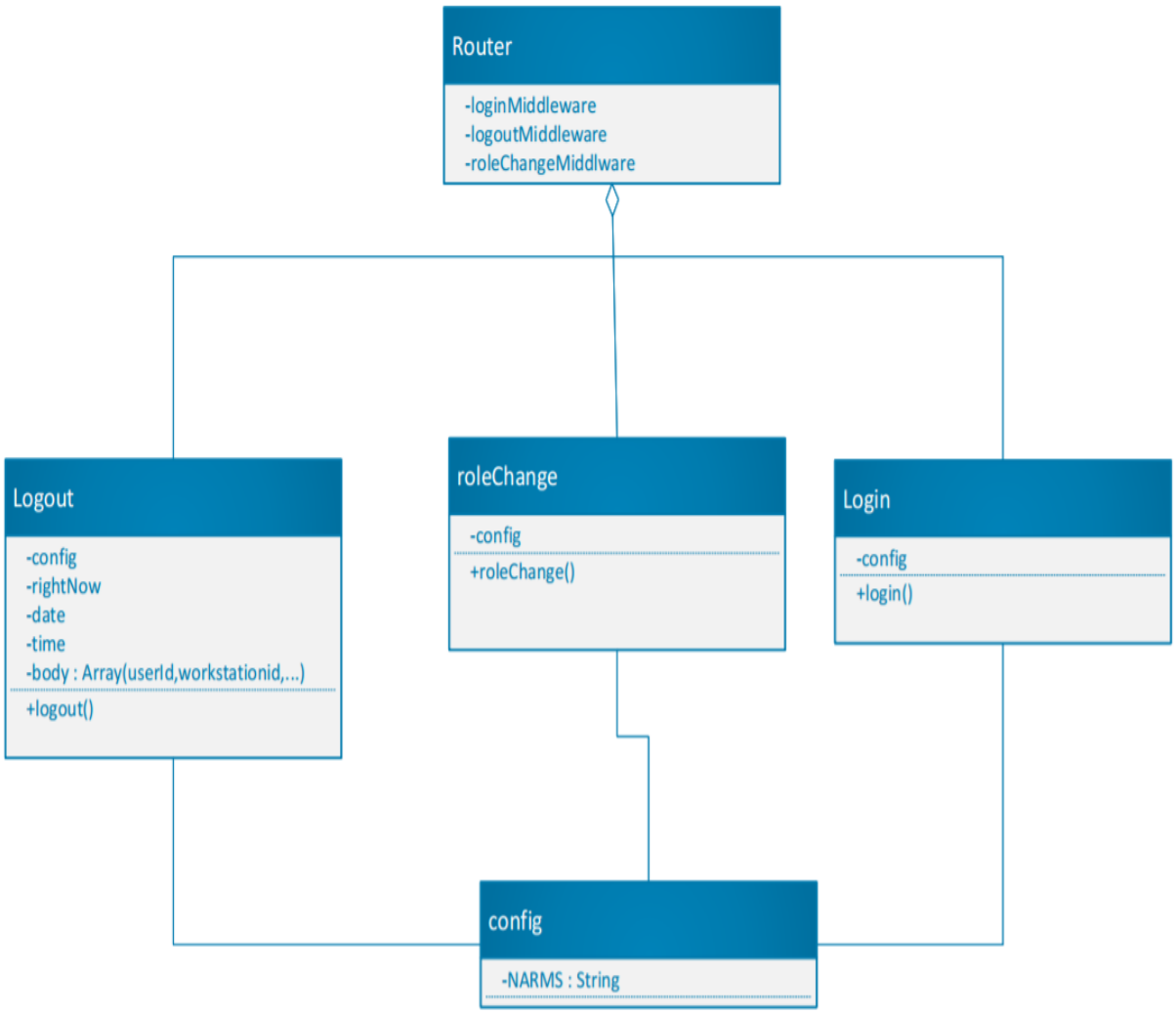


Figure 5 - Frontend Sequence Diagram

7. Process View

Frontend Sequence Diagram

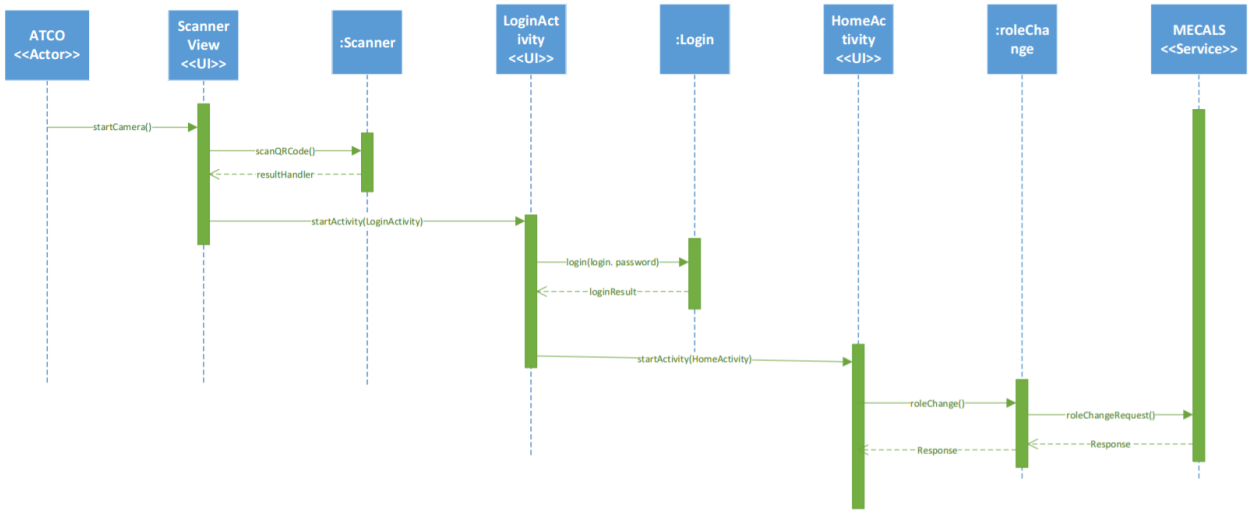


Figure 6 - Frontend Sequence Diagram

Backend Sequence Diagram

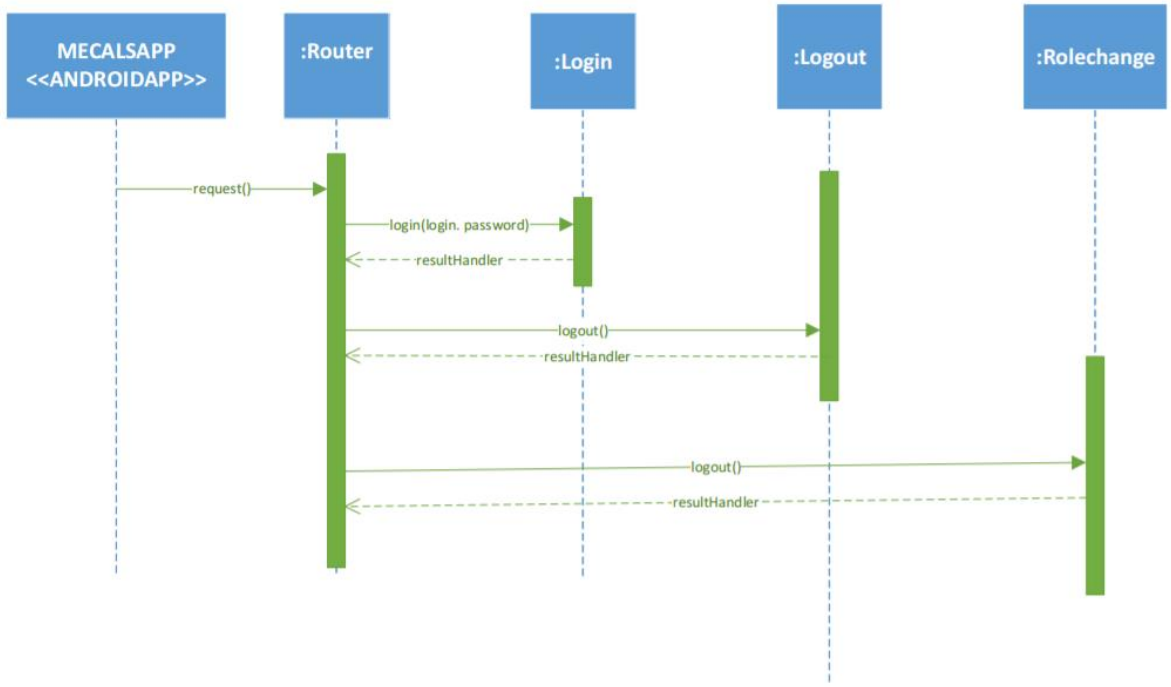


Figure 7 - Backend Sequence Diagram

8. Module Decomposition View

In the initial phases of development, due to poor documentation quality, as well as confusing legacy systems, the initial context diagram should have looked like this:

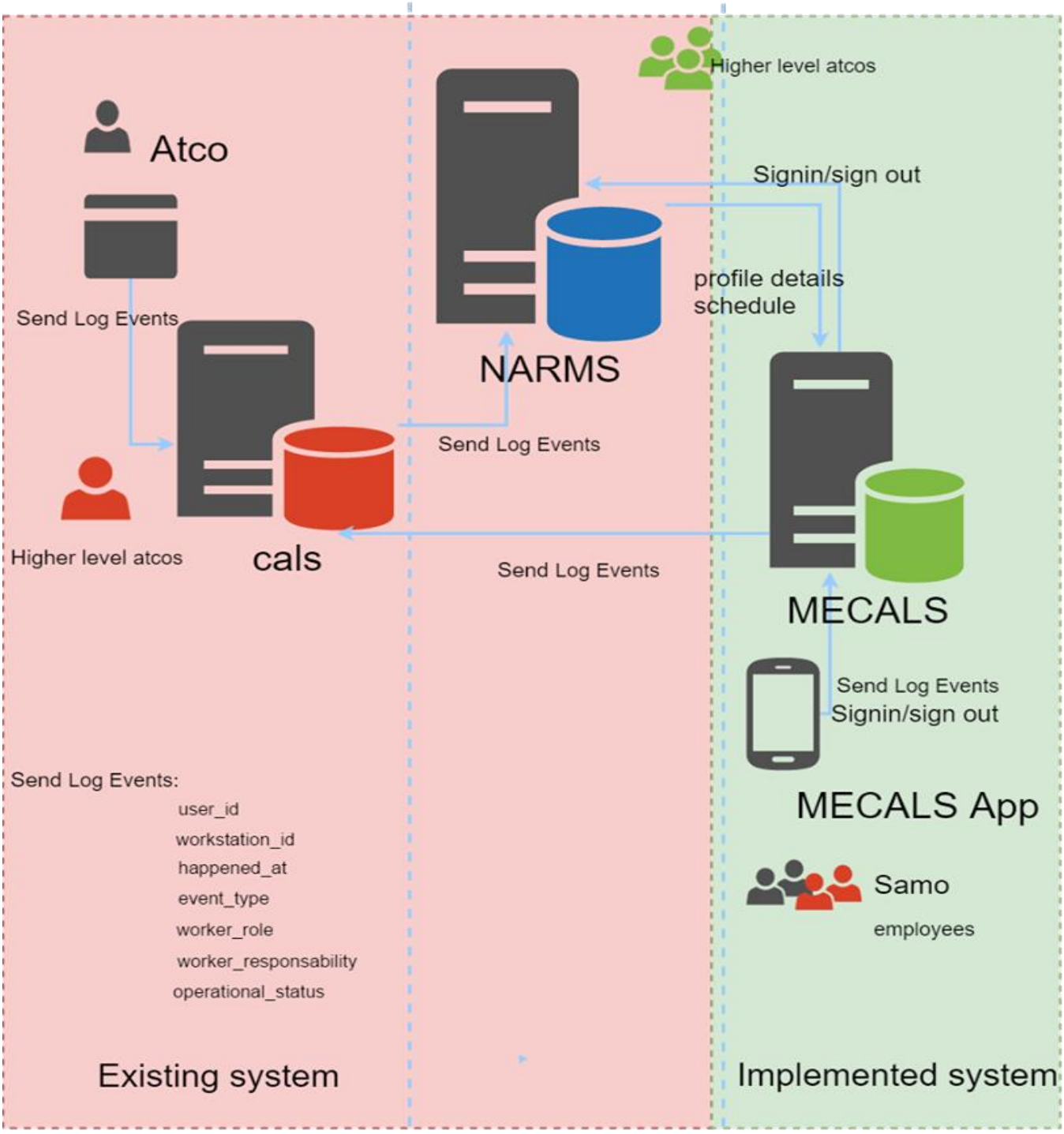


Figure 8 - Initial Context Diagram

However, in the later stages of the development and upon further inspection, it was discovered that the CALS Server module was practically useless, contrary to the documentation given. As such, the new context diagram has been adapted to:
At the end of the project it was discovered that MECALS could not use CALS as intended due to the CALS server not doing what it was suppose to. The CALS client was supposed to send data to the CALS server which should send it to the database, the server was however useless and the data was being sent to the database directly from the client.

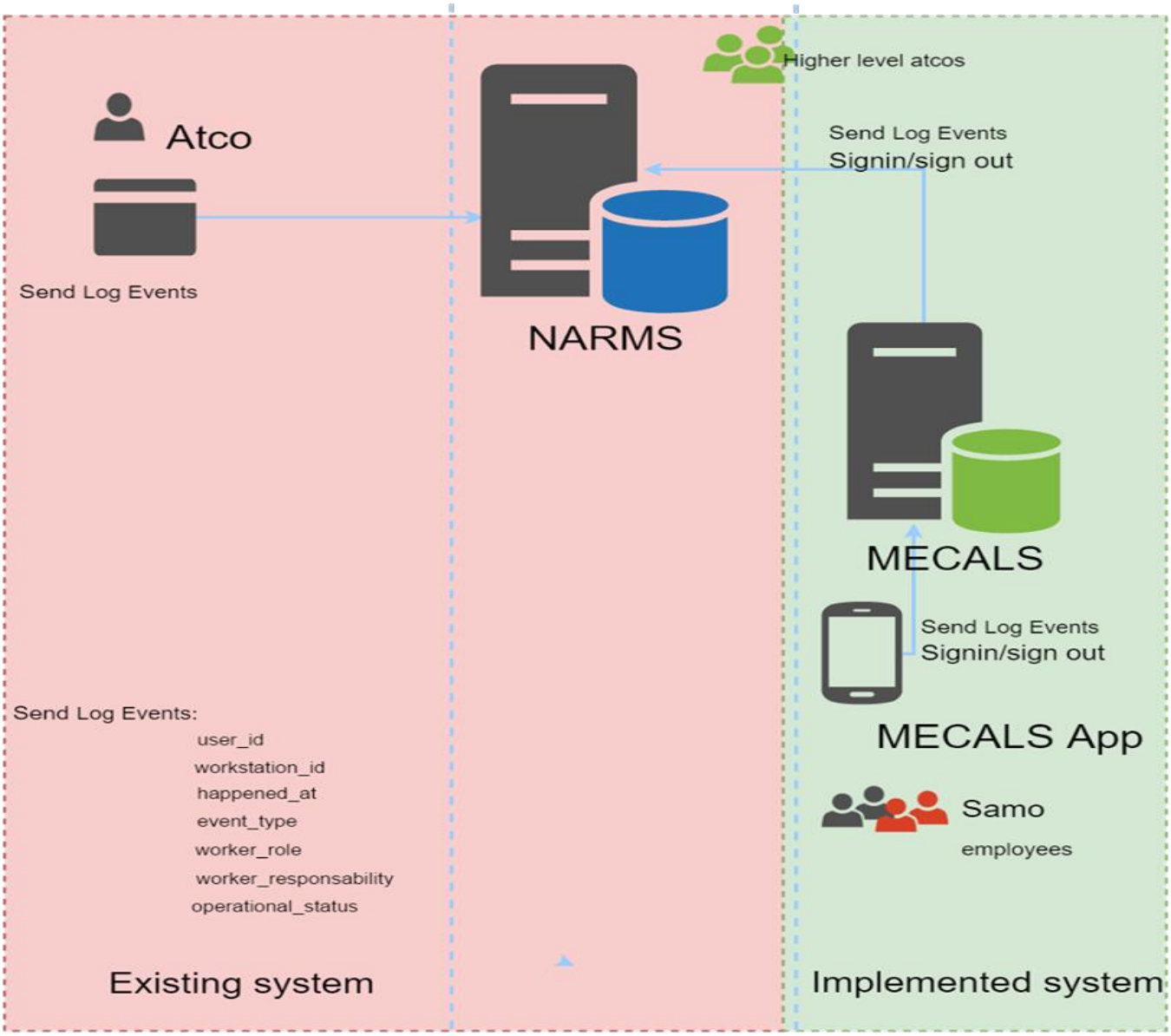


Figure 9 - Final Context Diagram

Documentation received for CALS and NARMS was picturing how it should be structured and not how it was structured. Below are two architectural diagrams of the backend, diagram 10 shows how MECALS was supposed to work and diagram 11 shows how it actually works.

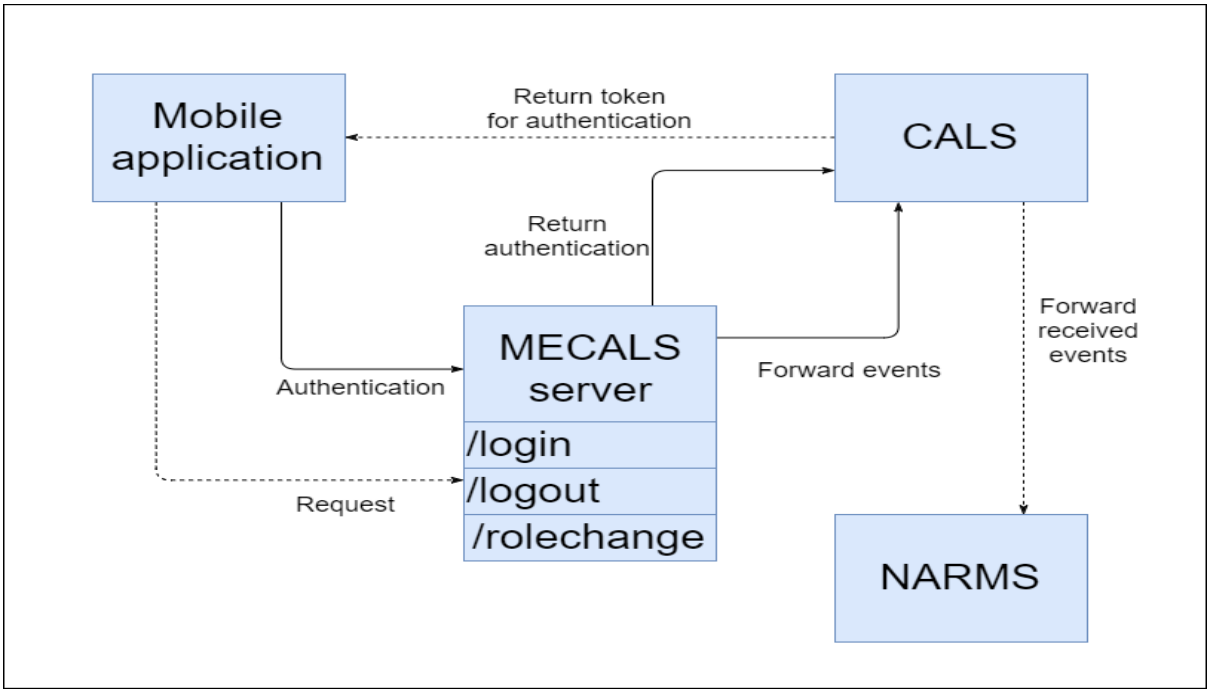


Figure 10 - Initial approach

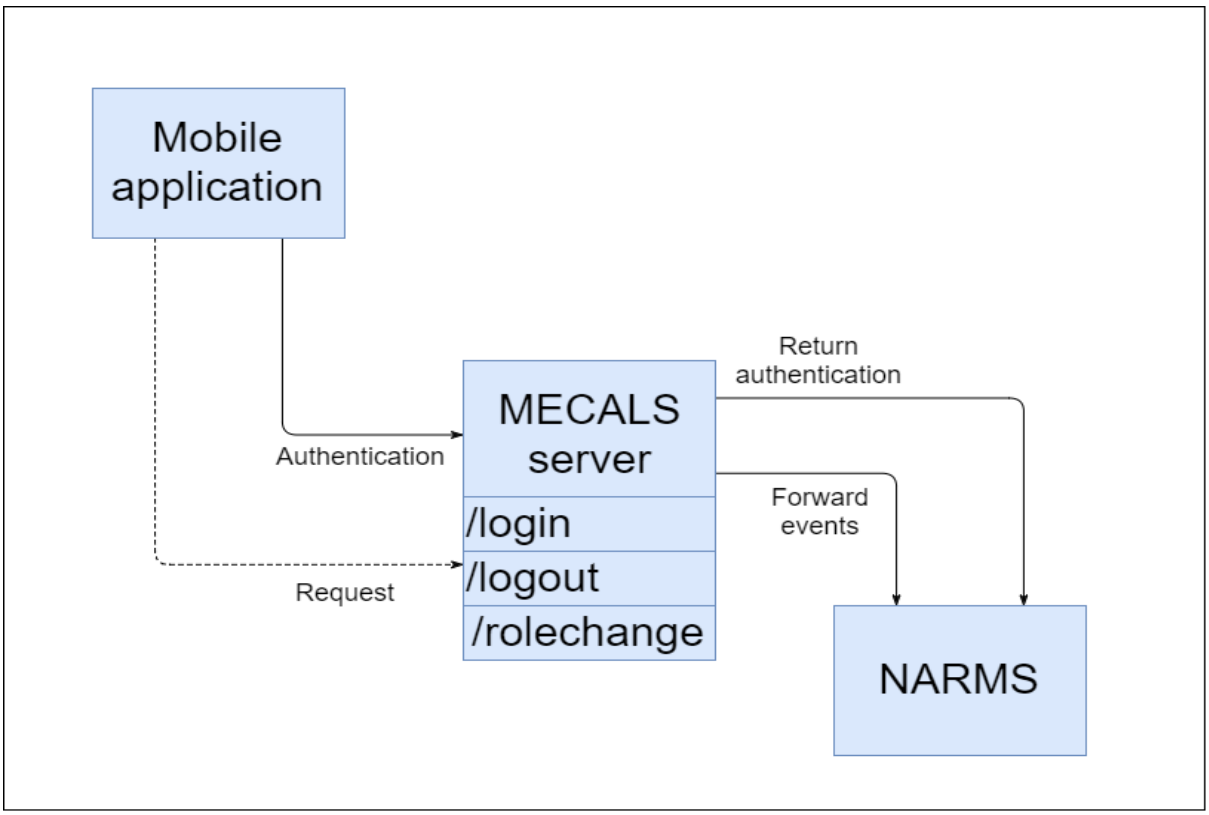


Figure 11 - Final approach

The code for the log events, passed between servers :

{

"user_id": "be7cf567591",
"workstation_id": "d02d3280377",
"happened_at": "2017-03-11 00:10:00",
"event_type": "status_change",
"worker_role": "Procedural Enroute",
"worker_responsability" : "Tactical",
"operational_status": "SC"

}

9. Deployment View

In this view, we see the bigger picture of the MECALS system. It starts with the workstation, which a QR Code printed on it. The MECALS Mobile App is used by the ATCO to Login, then reads the QR Code, which allows him to connect to the workstation. The mobile app sends a Login Event to the MECALS Local Server, which is hosted on a virtual machine, and in turn sends the data to the NARMS Local Server, also via HTTP.

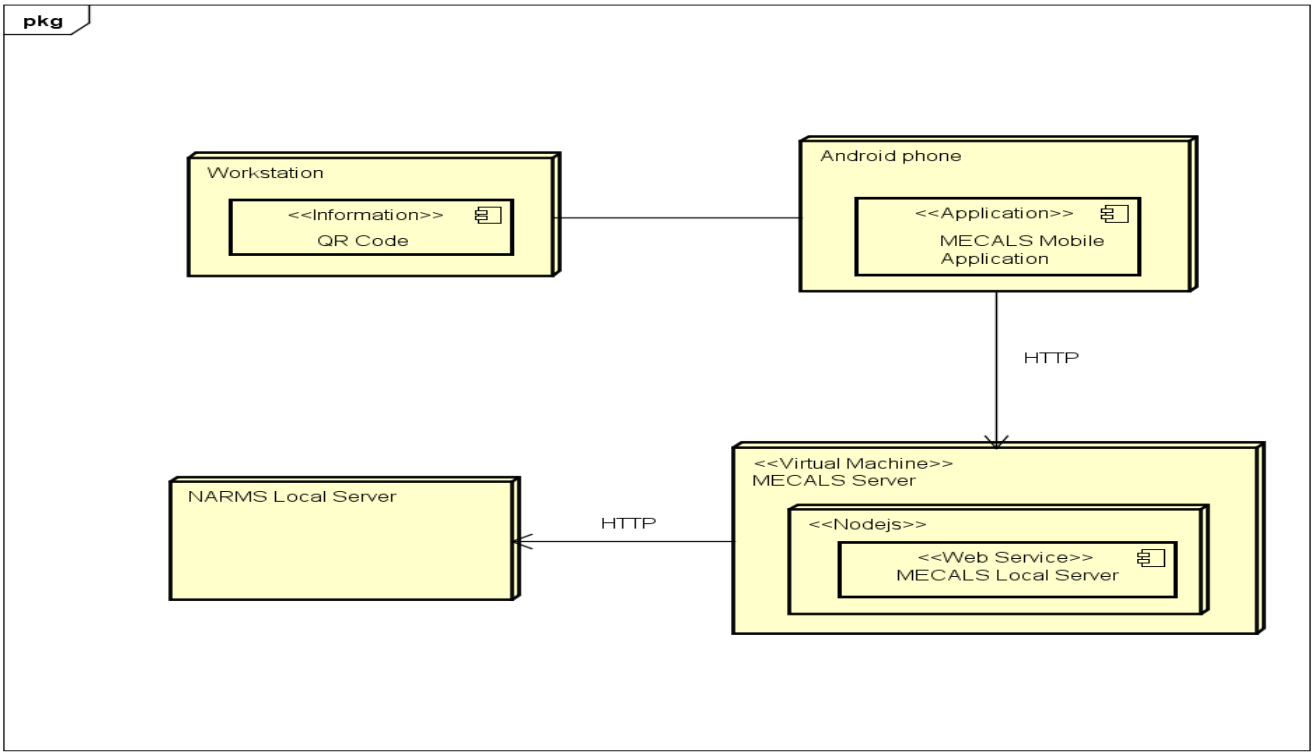


Figure 12 - Deployment Diagram

10. NARMS API

Absolute URL: http://193.10.30.126/api/cals

URL	HTTP Met	Params	Request Headers	Request body	HTTP Status	Response Body

/auth/login	POST	~	Content-Type: application/json	/** Use the same email and password that you use to login to NARMS through the login page
				*/
				
				{
				"email": "jdeuf@gmail.com",
				"password": "jdeuf"
				}
					200	/** This is just an example response. A user can have multiple Worker Profiles for different facilities. In case the authentication is successful, the response object has “True” for the key “success”. I return full info for the worker profile and facility so you have everything you need.
						*/
						
						{
						"success": "True",
						"worker_profiles": [
						{
						"worker_profile": {
						"id": 3,
						"pub_id": "be7cf567591",
						"active": true,
						"name": "Deuf",
						"first_name": "John",
						"date_of_birth": null,
						"created_at": "2017-03-08T21:30:39.000Z",
						"updated_at": "2017-03-08T21:30:39.000Z",
						"facility_id": 2,
						"user_id": 3
						},
						"facility": {
						"id": 2,
						"pub_id": "d53524a0330",
						"active": true,
						"name": "Nordica 2",
						"location": null,
						"created_at": "2017-03-08T21:30:39.000Z",
						"updated_at": "2017-03-08T21:30:39.000Z"
						}
						},
						{
						"worker_profile": {
						"id": 9,
						"pub_id": "813ed6021bf",
						"active": true,
						"name": "Deuf",
						"first_name": "John",
						"date_of_birth": null,
						"created_at": "2017-03-08T21:30:39.000Z",
						"updated_at": "2017-03-08T21:30:39.000Z",
						"facility_id": 3,
						"user_id": 3
						},
						"facility": {
						"id": 3,
						"pub_id": "2464b839666",
						"active": true,
						"name": "NextJet Jonkoping",
						"location": null,
						"created_at": "2017-03-08T21:30:39.000Z",
						"updated_at": "2017-03-08T21:30:39.000Z"
						}
						}
						]
						}

					200	/** In case the authentication is NOT successful, the response object has “False” for the key

						“success”. There is nothing else in the response in this case.
						*/
						
						{
						“success”: False,
						}

/$facility_pub_id/log_events	POST	The additional parameter here is $facility_pub_id which you add to the URL. For example, the URL to which you post can be http://193.10.30.126/api/cals/2464b839666/log_events, where 2464b839666 is the pub_id of the facility.	Content-Type: application/json	/**
				I will give you a brief explanation for some of the fields. Some of the are obvious.
				user_id – worker profile pub_id
				workstation_id – workstation pub_id
				event_type – this is the event type. This field accepts strings. For example, it could take the following values: “log_in”, “log_out”, “status_change”, “role_change”, “random_string”. I suggest you stick to only the first 3 values, because theoretically a role_change is done by the ASM system, which we don’t have.
				worker_role – keep this one constant because the role will not change. For example, it can always have the value “Radar Enroute”.
				worker_responsability – Keep this one constant.
				operational_status – it can take the following values: “SC”, “MCU”, “MCM”, “MCT”, “MCI”. To find out more about what they mean, read the MECALS Requirements document, page 17. Everything is explained there. MECALS backend implement some of this logic. Now it’s up to you if can manage to implement it.
				*/
				
				{
				"user_id": $worker_profile_pub_id,
				"workstation_id": $workstation_pub_id,
				"happened_at": "2017-03-08 21:30:39",
				"event_type": "log_in",
				"worker_role": "Radar Enroute",
				"worker_responsability": "Planning",
				"operational_status": "SC"
				}
					200	/** This is what you get in case the operation succeeds and the log event was stored in NARMS.
						*/
						
						{
						“success”: True
						}

					200	/** This is what you get in case the operation fails if for example the worker profile or the workstation doesn’t exist.
						*/
						
						{
						“success”: False
						}

Here is a list of user accounts you can use:

- jneimard@gmail.com/jneimard (shift mananger)
- tcrowford@gmail.com/tcrowford (shift manager)
- jdeuf@gmail.com/jdeuf (atco)
- mrouana@gmail.com/ "password is the same as the username" (atco)
- mmicoton@gmail.com (atco)
- aconda@gmail.com (atco)
- ajavel@gmail.com (atco)

11. Quality Attributes

11.1. General Quality Attributes

Priority	Title	Description
1	Availability	Because of the fact that an ATCO has to manage air space then has many lives on his hands, the tools he uses must always work properly and be highly available.
2	Security & Safety	For the same reason, MECALS must be secure and safe. A malicious attack on the system could have a several consequences.
3	Robustness	MECALS must endure unusual situation in order to insure its first priority as the availability. Air space control depend on a hazardous environment like unpredicted delays on arrivals planning.
4	Resilience	MECALS must endure sudden burst of load for the same reasons quoted earlier.

11.2. MECALS Server Quality Attributes

Priority	Title	Description
5	Efficiency	The serve must be responsive.
5	Usability	The application must be easy to use by the intended users.

11.3. MECALS Application Quality Attributes

Priority	Title	Description
5	Efficiency	The mobile application must load quickly and be responsive to any device.
5	Usability	The mobile application must be easy to use and understand by the intended users.

12. Issues and concerns

	Change	Reason
1	From iOS to Android	Lack of testing and platform specific devices.
2	From Xamarin to Java Android API	There is no need for Xamarin since there will be no need for a cross-platform application (only for android). Xamarin is platform dependent. However, since the initial decision for iOS was changed, it is no longer needed.
3	From C# to Nodejs for the back end	Nodejs is more suitable to implement the backend for MECALS. This decision was made after a discussion between the development team, product owner and the architects.
4	From C# to Java for the front end	Android Studio works with Java, and since we decided not to use Xamarin, C# was dropped. There are not many differences between C# and Java, except for specific cases.
4	From AppVeyor to Travis	Travis will be used instead for continuous integration. There is no need for AppVeyor since there will be no iOS app.

13. References

Kruchten, P. (1995, November). Retrieved from http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/Pbk4p1.pdf

Paul Clements, R. K. (2013). *Software architecture in practice.* New Jersey: Upper Saddle River, N.J. : Addison-Wesley .