Architectural Requirements Specification

Organisation: `https://github.com/Hyperperform`

Developers:
Rohan Chhipa *14188377*
Avinash Singh *14043778*
Jason Gordon *14405025*
Claudio Da Silva *14205892*

Updated October 13, 2016

Client

MagnaBC
`http://www.magnabc.co.za/`

Contents

1 Software Architecture Scope

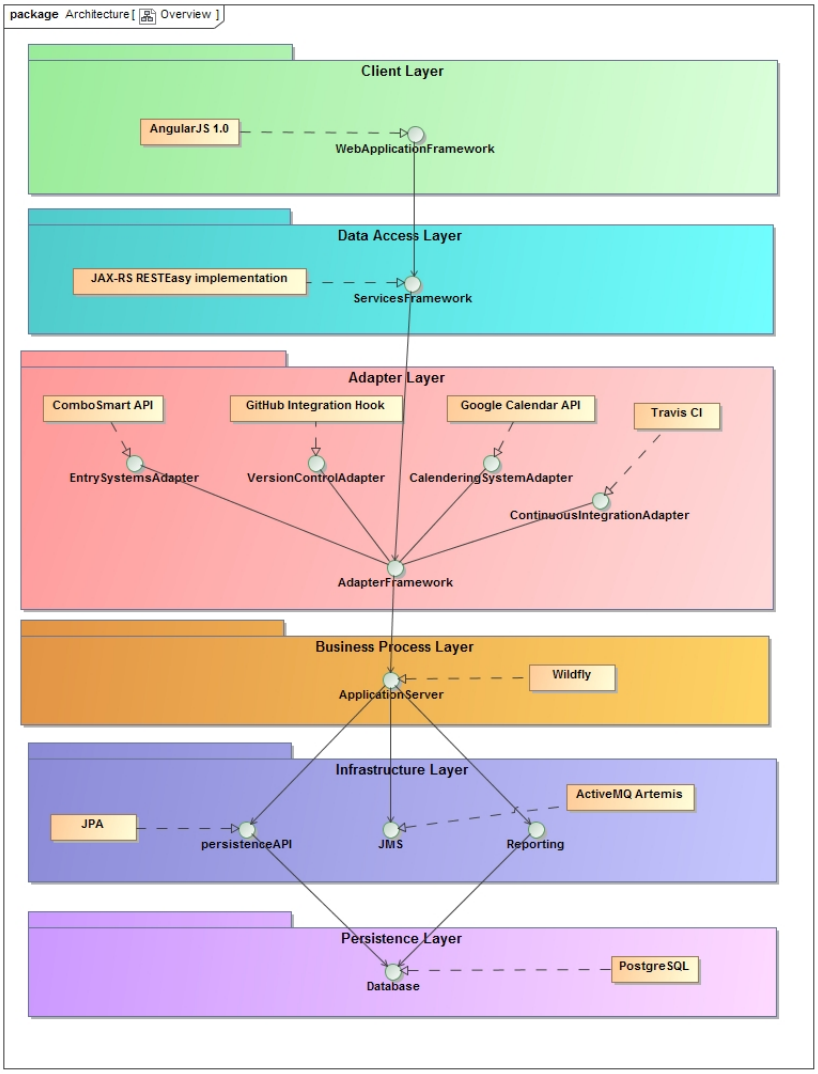

Figure 1: A high level break-down of the architecture to be used

An explanation of some of the adapters follow:

- The EntrySystemsAdapter, provides an interface which various entry systems such as card readers, biometric scanners and the like, can map onto.
- The VersionControlAdapter, provides an interface which various version control systems such as GitHub, BitBucket and GitLab can map onto.
- The CalenderingSystemAdaoter, provides an interface for various calendering systems such as Google Calender, or Outlook calender to map onto.
- The ContinuousIntegrationAdapter, provides an interface which various Continuous Integration systems, such as Travis or Jenkins, can map onto.
- The BugSystemAdapter, provides an interface for which bug tracking systems such as Waffle.io or Trello can map onto.

Extended release adapters will include:

- The WearablesAdapter, which should provide an interface for wearable devices such as Fitbits and Samsung Gears can map on to.

Further information will be provided under integrations.

2 Overall Software Architecture

2.1 Quality Requirements

The following section will focus on the quality requirements of the overall system at its highest level of granularity, and all the requirements defined are defined for all components of the system.

2.1.1 Flexibility

The most important element of the system is its flexibility. Each component of the system is decoupled from every other component, thus allowing you to change certain components without affecting anything else. The main focus of the software is to source data from various integrations that change with the current market trends and business environment. Thus the need for additional integrations and replacement of default integrations will arise, allowing for a system fully embracing the principles of IOT (Internet of Things).

The system is decoupled in the way that it is not limited to specific architectures, thus it should not be locked down to a specific provider for either the application server or the database.

2.1.2 Maintainability

Another very important aspect of the system is the maintainability of the system, to the extent that the system is easy for future contributors to change, maintain and further develop.

- Future developers are able to easily understand the system, and modify it as needed provided that the specified contracts are adhered to.
- The technologies chosen for the system can be reasonably expected to be available for a long time as well as remain open source and available. However due to the front-end being completely decoupled from the back-end it would be easy to change the front-end technology without affecting the back-end functionality.
- The system can be maintained without having the provided services unavailable for large periods of time.

2.1.3 Integrability

The system allows for easy future integration requirements by providing access to its services using public standards. All external systems should have a preprocessor interface on which to communicate and map on to.

2.1.4 Scalability

The system should theoretically be usable in large industries of thousands of employees. Thus the system has been designed to handle the load and grow as needed. To accomplish this we make use of the message bus architecture.

2.1.5 Reliability

The system is reliable in the fact that many events will be constantly captured. All events will be persisted and used at a later stage thus leading to more accurate calculations.

2.1.6 Security

This is not the main focus of the system however for any system this requirement is important. The following security measures have been put in place:

- Proper authentication is to be preformed in order to gain access to confidential user information from the database.
- Outside components can only gain access through the provided REST endpoints.

2.1.7 Auditability

Logging occurs for:

- Events captured into the queue.
- Access to the users personal information.

Logs contain at least for all different logs:

- An id pertaining to user performing the task.
- A description of the activity happening.
- The time at which this activity happened.

2.1.8 Usability

The system has been created to be intuitive and easy to use, the system is not difficult for a user to learn and interact with. Some things that have been emphasised are:

- Useful and helpful error messages, as well as client side validation as far as possible.
- Labels and hints where possible to guide users.

2.1.9 Testability

The system is testable through Continuous Integration using:

- Automated isolated tests using mocks
- Automated integration tests using the actual realizations

Tests are run ensure that all pre and post-conditions are properly met and that the system is in working order.

2.1.10 Deployability

- The system must be built using only the build tool and scripts.
- The system is be deployable on cross platform environments.
- The system is easily adaptable to different environments with different databases and message brokers.
- The system has been Dockerised, allowing for easy distribution.

2.2 Architectural Constraints

The following constraints are non-negotiable and must be met for the system to properly meet client requirements:

- The system is platform independent, allowing for docker to handle the environment setup.
- All software used in the creation of this project, including frameworks, libraries and application servers must be strictly open source, as well as preferably well supported and long term.
- The web interface runs correctly on all Webkit enabled browsers.

2.3 Integration and Access Channels

In terms of access channels, the project as it stands focuses on two main forms of access:

- The web-client component, which will make use of REST calls in order to interact with the back end server. The web client is written in AngularJS 1.0, and interacts with the server using the RESTEasy implementation of JAX-RS REST wrapping. All services are only exposed through REST wrapping. Thus leaving no other access channel to the services.

With regards to integration, the system has the following adapters on which to map external integrations on to :

- The EntrySystemsAdapter, provides an interface which various entry systems such as card readers, biometric scanners, etc. can map onto.
- The VersionControlAdapter, provides an interface which various version control systems such as GitHub, BitBucket and GitLab can map onto.
- The CalenderingSystemAdaoter, provides an interface for various calendering systems such as Google Calender, or Outlook calender can map onto.
- The ContinuousIntegrationAdapter, provides an interface which various Continuous Integration systems, such as Travis or Jenkins, can map onto.
- Similarly the events that are retrieved will come through the REST wrapping designed specifically for the events. Each integration has its own REST URL to which it sends events.

Thus since the events come through the REST URL's we don't have to continuously poll these systems. Instead the separate systems send the events when ready, these are known as webhooks. Once the events are received they are processed and placed on a messaging queue.

However certain integrations might not support webhooks. A recommended approach to this problem would be to create a separate event poller, which is independent from the HyperPerform system. The poller can then be used to simulate webhook technology by sending the events to the provided REST URL's. Thus the HyperPerform system resources are not wasted on event polling and consistency is kept amongst the event listeners.

3 Architectural Design

3.1 Architectural Tactics

At the system level of granularity, that being the highest level of granularity, we do not specify any architectural tactics in order to concretely address the quality requirements for the system.

3.2 Architectural Patterns

Some of the patterns used at the system level include:

- The layered pattern, which limits access of component within one layer to components which are either in the same or the next lower level layer. A great strength it has is that a can be relatively easily replaced. In particular, one can add further access channels without changing any lower layers (A critical component of this system) within the software system and changing the persistence provider would only require changing the persistence API.
- The Message Bus pattern, an enterprise integration pattern which allows for safe and efficient handling of events via messages, using a response and request queue, one can ensure that events and messages never get lost even if part of a system fails, thus allowing a greater fail safe in the system.

3.3 Reference Architectures

The key architecture at the system level is the Enterprise Service Bus. In order to properly network all devices in a proper IOT (Internet of Things) manner, one would implement a messaging queue system. We chose ActiveMQ Artemis as our realization of this pattern, as it integrates quite nicely with JavaEE, especially the WildFly server we are running, and has decent support, as well as many features we can make use of.

4 Technologies

4.1 Back end

- WildFly as the JavaEE container and server.
- JUnit as a unit testing framework along with Spring.
- RESTEasy implementation of JAX-RS for REST wrapping.
- ActiveMQ Artemis as the message broker for the system.

4.2 Front end

- HTML5, CSS/SCSS and SASS, Bootstrap and JQuery.
- AngularJS 1.0 is used to access the REST services.
- ChartJS to create appealing charts on the front-end.

4.3 Persistence

- PostgreSQL for the database.
- JPA as an Object Relational Mapper.

4.4 Build tools

- Maven for assembling war applications for the back end.
- Gulp for clustering code into one application for the front end.

4.5 Deployment tools

- Docker as a shipping container.
- TravisCI as a Continuous Integration tool.