Testing and Review Report on Team Bravo Integration Repository Code

COS301 Software Engineering Research Paper Management System University of Pretoria

Team Echo

Surname, First Name (Initial)	**Student Number**
11089777	Broekman, Andrew (A)
12153983	Andrews, Stuart (SD)
13133064	Shneier, Jedd (J)
14006512	Singh, Emilio (E)
14009936	Cromhout, Reinhardt (RR)
14040426	Loreggian, Fabio (FR)
14077893	Jita, Hlengekile (H)
14101263	van Wyk, Gerard (GJ)
14214742	Botha, Matthew (MT)
14446619	Buffo, Gian Paolo (GP)
12026973	Antel, Marc (M)

Contents

1 Background

The client, Vreda Pieterse, from the University of Pretoria has requested a system to keep track of research publications in the Department of Computer Science at the University of Pretoria. The scope of the system is managing the administration involved in tracking of research publications within the department. However, collaboration on research papers is outside of the scope as a version control system is in use currently. The system is required to keep track of all publications and the associated metadata around the publications.

This report evaluates the implementation of the support research system by the Bravo Integration team of phase 3. This report specifically evaluates whether the team has complied with the stated functional and architecture requirements as set out in the respective documents provided by the client.

The source code reviewed can by found at the following Github repository `https://github.com/DillonHeins/Bravo`

2 Functional Testing Report

2.1 Notifications

The Application Requirements and Design Specification provided by the client for the Research Support System states that the Notifications module is to provide the ability to send once-off or reminder(i.e. in intervals) messages to users. The following types of notifications need to be sent using this module:

- Reminder
- Activity
- Report
- Broadcast

As per the scope in the specification, this means the following business methods need to be implemented in order to provide these services:

- sendReminder
- sendActivityNotification
- sendReport

The Notifications module itself in the current implementation of this system does provide this functionality but none of the implementations are correctly done. This module successfully considers the sending of broadcast messages by taking in email lists in the function parameters, but in none of the methods is the email sent to multiple users. In fact this module fails to send the messages at all in any of the module services, this being the modules key purpose. For notifications that are to be sent in intervals such as the reminder notification, there is no use of scheduling or even the receiving of the precondition of interval concepts such as duration or dates in the method signatures.

However this should not have affected the integration of the notifications modules expected functionality because where the current implementation of the Notifications module failed in implementing the required functionality, they would simply mock what was required to allow the system to be functional despite the modules failures.

For the Research Support System as a whole to provide this functionality, integration of this module would require functionality to interact with various modules of the system. In the case of the Notifications module this implies providing the services of the Notifications module to the external aspects through providing a standard way for the client aspects to communicate with this server-side module.

To do this, integration would need to call the Notifications services, pass the required information in the right format, depending on the type of notification required and the business service to be used, and process a response of a certain format and forward it in the required format.

This functionality includes the ability to process information from these external aspects which would come in as JSON objects, and forward it to the Notifications module so the necessary service can be run using that information. In addition to that the information sent back by the module would need to be processed back into a format for client aspects of the system. Thus the use of Request and Response objects were to be used.

Unfortunately Integration of this module failed because none of the above outlined functionality is provided(mocked or not). The only function the current Research Support System provides in terms of Notifications, is the ability to get a notification from the Notifications module and return it as a RESTful web service.

There is no processing of information never mind any acknowledgment of information from the client aspects of the system. Thus the information is never sent to the Notifications module itself. In the mocked version of the module, there is no evidence of any of the expected functionality, that is sendReport, sendReminder or sendActivityNotification, or the fact that there are various types of notifications that require different information in the forms of requests.

The integrated system also does not deal well with the various possibilities for responses. Whether the notification was able to send successfully or not. It only gets a String from the notifications module which it immediately sends to a RESTful web resource.

The integration of this module thus has dire shortcomings and lacks all the key required functionality.

2.2 People

According to the Application Specifications, the persons module is responsible for maintaining demographic information about the persons (researchers) themselves, some of which may be users of the system and others not as well as groupings of persons into research groups.

The scope of the persons module includes

- adding persons and modifying their details including the researcher category a person is assigned to.
- adding groupings of persons into research groups as well as grouping groups into higher-level groups.
- defining and changing group memberships of persons.
- creating and modifying researcher categories with associated research output targets.

On the first point, adding and modifying, Bravo Integration provided sufficient mock objects to create a Person object. In the person class, various mutator methods were provided to edit the various Person details, such as the name, surname, id, email and privileges.

A sufficient Person constructor method also creates the correct Person mock object.

Each person has ResearchGroupAssociation's linked list associated with their object, containing the groups that they're part of, as well as another list specifying research category.

On the second point, each user is successfully grouped as per their privilege levels.

On the third point, the functionality required, that being able to define and change groups, only the former was correctly implemented. The group is defined using the Group.java class, where the name etc. of said group is set. As well as setting the name of the group, it allows PersonEntities to be add to the group, this last part keeping with defining user group associations.

The changing group association, is also implemented correctly. Through the PersonEntity.java class, there is an option to setGroup, which changes the users current group.

On the fourth and final point, creating and modifying researcher categories, the requirement was to be able to create and modify research categories, but they also needed associated research output targets. The ResearchCategory.java allows the user to define categories. Including a data, which is associated to the research output targets.

On the whole of it, the People section mock object was implemented correctly and should behave correctly.

2.3 Publications

According to the functional requirements, the publications module should provide the functionality of maintaining information to track publications through their life cycle, from being envisaged to ultimately being either published or abandoned.

2.3.1 addPublication

The addPublication service should create a new publication with its own state trace represented by a time-ordered sequence of state entries. It should accept an AddPublicationRequest as an argument and return an AddPublicationResponse.

This service was correctly defined in the Publications interface. However, the request and response objects were not implemented. As such, the post conditions for this service contract were not met: a publication with an initial state entry was not created, it was not persisted, and author users who requested notifications were not notified of the creation of the publication.

A basic functioning mock object was created that checks against the pre-conditions associated with adding a new publication, such as whether a publication with the same title already exists for the same set of authors, and whether the request is valid or authorised.

A rudimentary unit test was created but did not test any actual functionality, as it merely defaulted to a failed test.

2.3.2 changePublicationState

A publication has a sequence of state entries representing the state trace for that publication. This is effectively an implementation of the *memento* design pattern, with each state entry being a memento capturing a snapshot of the state of the publication. Therefore, changing the state of a publication should add a new state entry as described above. The changePublicationState service should accept a changePublicationStateRequest as an argument and return a changePublicationStateResponse.

This service was correctly defined in the Publications interface, and a PublicationState entity was created. However, only basic getters were implemented, in terms of retrieving the state of the latest publication. This does not allow one to view the state history of a publication. Additionally, no checks were made to see whether the person attempting to change the publication state was an author of the paper or the research leader of a research group which one or more of the authors are members of. Furthermore, no checks were made to prevent the title of the publication from being changed to a title that already exists for the authors, checking that the publication exists and ensuring that it has not yet been published. As such, none of the pre-conditions were met.

In terms of post-conditions, basic setters were implemented to alter parts of a publication, such as the publication target. However, the new publication state was not added to a sequence of state entries. Additionally, no notification functionality was implemented.

A rudimentary unit test was created but did not test any actual functionality, as it merely defaulted to a failed test.

2.3.3 addPublicationType

Administrators should be able to add new publication types, each with a unique name and an initial state. This state should typically be an active state to which zero or more accreditation points are assigned.

This service was correctly defined in the Publications interface. Functionality for adding a PublicationType entity was included, but contained mostly rudimentary functionality, such as returning the state of the latest publication.

The addPublicationType service should accept an addPublicationTypeRequest and return an addPublication-TypeResponse. The aforementioned request and response were not implemented. As such, the pre-conditions that users must have administrator rights and that a publication type with the same name does not exist yet were not met. Although a framework was set up to make use of the JPA, the publication type was not persisted. Thus the post-condition was not met either.

A rudimentary unit test was created but did not test any actual functionality, as it merely defaulted to a failed test.

2.3.4 modifyPublicationType

A publication type should also have a state that can change at any time. For example, the number of accreditation points linked to a publication type can change from some effective date onwards (while still remaining active) or by being deactivated on some effective date. It is once again important to maintain a state history so that the number of credits for a publication of a particular publication type can be correctly calculated.

This process involves creating a modifyPublicationType service, which receives a modifyPublicationTypeRequest as an argument and returns a modifyPublicationTypeRequest as a result. Once again, the service was correctly defined in the Publications interface and the request and response objects were declared, but never implemented.

The PublicationTypeState entity was created, which included provision for the effective date after which accreditation points for a specific publication type become valid.

Due to the lack of implementation, the pre-conditions regarding whether a user is an administrator or whether the given effective date is after the last effective date in the state history of a specific publication type were not met. Functionality for creating a new state entry is present, but it does not add this entry to a history of other state entries.

A rudimentary unit test was created but did not test any actual functionality, as it merely defaulted to a failed test.

2.3.5 getPublicationsForPerson

This service should return all publications for an author which either have been published, accepted or are envisaged to be published for a user within a specifed time period. If no time period is specifed, then all matching publications over all time should be returned.

This service was correctly defined in the Publications interface. Functionality for retrieving the list of publications is correctly implemented, however no autherization is done. The is no pre conditiosn for this contract and post condition of matched results being returned is met.

There is a unit test which succesfuly shows the correct workings of the service ,however dependencies are hard coded in. According to specifications these should have been handled by dependency injection.

2.3.6 getPublicationsForGroup

This service is similar to the getPublicationsForPerson service except that it should return all published, accepted or envisaged publications for a period for a group. The group may be a frst level group (e.g. a specifc research group) or a higher level group (e.g. the department).

This service was correctly defined in the Publications interface. Functionaly however it is inccorectly implemented as the group level is not taken into account in the request object. Besisdes this the functio ngets a list for a specified group name.

The unit test went much like the getPublicationsForPerson unit test. Though it is inccorect as it cannot distinguish from group level. Also fails to implement dependency injection.

2.3.7 calcAccreditationPointsForPerson

This service should find all publications published, accepted or envisaged to be published for a period by a person and sums up the accreditation points earned.

This service was correctly defined in the Publications interface. Itsfunctionality is correctly implemented in the Bean and all requests a nd responses were implemented. the post and preconditions of the service contract are implemented.

The unit test succeded but fails to once again make use of dependency injection isntead hard coding mock objects.

2.3.8 calcAccreditationPointsForGroup

This service is similar to the calcAccreditationPointsForPerson service except that it is supposed to accumulate the accreditation points for all persons who are part of that group. The group may be a frst level group (e.g. a specifc research group) or a higher level group (e.g. the department). All members of a sub-group are also members of any higher level grouping.

This service was correctly defined in the Publications interface. Its core functionallity is correctly implemented, but the request object is incorrert as it does not allow higher level group searches and does not take into account group-member heirarchy grouping.

The unit test succeded but fails to once again make use of dependency injection isntead hard coding mock objects.

2.4 Reporting

The Reporting module should generate visualisations of information. Two different types of reports were expected to be generated. One that returns statistical information of accreditation Units that comply with certain restrictions, such as a date period and/or author. Another that returns the progress status of publications indicated in percentage.

Due to a lack of implementation and a mock on the reporting team, integration bravo was tasked with making their mock for reporting. The mock that was created did not create a report, but instead just returned a string stating which report was meant to have been created. Integration alpha's system doesn't the reporting methods that were required in the specifications. They created reports depending on the types of graphs instead of an accreditation unit report and progress report.

Functionality testing was very limited due to no input requirements from a user and basic output from the system.

2.5 Import/Export

The following import and export requirements were outlined in the functional requirements:

- Importing and exporting of persons and research groups from a CSV file
- Importing and exporting of publications for a person from a CSV file
- Exporting published papers for a user or a group to a bibtex file

None of the above requirements have been met in the system due to the fact that no import or export implementation or mocking exists.

3 Architecture Compliance Analysis

3.1 Architectural responsibilities

The team did successfully implement the Web Access architectural responsibility by providing and exposing RESTfull web services using Java EE.
The ProcessExecutionEnvironment responibility was addressed by the glassfish appilication server which is discussed under the heading "Architecture design & tatics".
Furthermore the Reporting, PersistenceAccess, and Persistence responisbilities were not addressed by the integration team as was required.
The MobileDeviceAccess and BrowserAccess were both access the appliction through the Web Access responsibility.

3.2 Architecture constraints

The system was deployed on a Linux server and Glassfish does provide the required support for hot deployment as required.

3.3 Architecture design

3.3.1 Architectural components addressing architectural responsibilities

- The ProcessExecutionEnvironment responsibility is partially fullfilled by the application server Glassfish. However the Glassfish server was not deployed as a docker image which was required.
- The responsibility of persisting domain objects to a database was partially implemented using JPA. Some Mock Objects used the correct annotations while other Mock Objects were not created at all or were not correctly annotatated Using for example the @Entity and @Id Annotations.
- The responsibility of providing an environment for specifying and executing reports to a reporting framework was not implemented.

- The responsibility of providing human access channels and external systems web access to the system services to a web services framework was implemented via RESTfull services via Java REST wrapping objects.

3.3.2 Infrastructure

This section analises the layered architectural pattern that should have been implemented.

- The client layers were developed by submoduels and should not have been touched by die integration team.
- The Acces layer was provided by REST wrapping the resources via Java EE which has been discussed already.
- The bussiness proccess layer runs the glassfish applicaiton server which has been discussed already.
- The persistence Access layer was partially implemented in terms of Java JPA and not at all in terms of Jasper Reports.
- The persistence layer was not implemented correctly to provide a connection to a POSTGREsql database.

3.4 Quality requirements

3.4.1 Flexibility

The software architecture specification required implementation of the following flexibility tactics:

- hot-deployment
- contract based software development with dependency injection

The Glassfish application server allows for hot-deployment of an application into a live environment.

The software architecture specification required the use of contract based software development with Java Contexts and Dependency Injection (CDI) dependency injection. The integration team implemented the service contracts and domain models, however the implemented interfaces and domain models don't conform to the contract based software development approach.

In a contract based software development approach service contracts implement the functions which the service provider should expose with Plain Old Java Objects (POJOs) used to transport data between the database and business layer. The integration team defined interfaces for the all POJO's and in the process did not define an interface or service contract for the service providers, against which implementations or so called realizations and mocks could be implemented against.

Further more the integration team used @EJB annotations, the old Java EE annotations for dependency injection, instead of @Injection annotations, which is the newer CDI public standard annotations. An example of this can be seen where the integration team injected objects to do their REST wrapping.

As the integration team is using the Glassfish application server, the application server allows for maintainability through the use of dependency injection through interfaces. However for this to work, the beans should be implemented against an interface, which the integration team failed to do, as the bean consists of only a class which derives from the java.lang.Object object. The way in which the integration team implemented their system is a cause of concern for maintainability and hot deployment, the reason being that other submodules will need to depend directly on the bean object instead of the specified service contract interface.

Futhermore the felxibility requirements specified that the access channels need to be exchangeable. This is avaiable since the system provides restful web services that can be acces via a variety of front end applications. It was also spesified that the persistence and reporting frameworks should be changeable in the future. Since the team partially implemented the JPA standard annotations for the the persistance it would be possible to change the ORM that is used on the server in the future. However the Reporting framework was not implemented correctly at all.

3.4.2 Maintainability

The Java EE reference architecture has various open standards which has at least one open source implementation for an application server. This will allow for future maintainability and hence satisfies the required criteria. Further more the lack of implementing beans against a service contract, will increase the difficulty and cost of maintaining the source base. And as mentioned before the implementation of the repoting framework is heavily lacking and will make it harder to add reports in the future.

3.4.3 Scalability

The use of Glassfish as an application server will deliver the required functionality of thread-pooling, object-pooling, connection-pooling and clustering as specified by the architecture requirements.

3.4.4 Performance

The use of Glassfish as an application server will deliver the required functionality of thread-pooling, object-pooling, connection-pooling and clustering to ensure that the required response speeds that are requsted are achieved.

3.4.5 Reliability

As required by the software architecture specification all service methods should utilize managed transactions, specifically using declarative transaction annotations. The integration team however has no transaction control in the service classes, hence this quality requirement has not been met.

3.4.6 Security

Within the current code base there is no security implemented which implies that no declarative role based authorization or security frameworks is used within the system, in this regard the required security requirements have not been meant.

3.4.7 Auditability

The architecture specification required that the auditability implementation must be maintainable, and should be implemented using aspects or interceptors. The current system however has no support for auditability.

3.4.8 Testability

The current unit and integration tests implemented don't fully test the submodules, including the required pre- and post-conditions. This further extends into the integration tests, which doesn't ensure that all client request conform to all pre-conditions.

The integration testing also doesn't extend to testing the database including the persistence layer, either in the application server or outside the application server using an in-memory database such as H2 or Derby.

3.4.9 Usability

The majority of this sections responsibilities lies with the two front end modules. It is however worth noting that the system is currently only avaiable in english and not in Afrikaans and Sepedi as well.

3.4.10 Integrability

Since the backend of the system was developed by combining 4 separate submodules(Publications, People, Notifications and Reporting) it was ensured that all use cases which are available to human users are also accessible from external systems, which was a requirement.

3.4.11 Deployability

As the system is developed in Java, the system will be able to run on Linux servers as required by the specification, since the Java Runtime Environment is available for the Linux Operating System. However the specification required that the application should be deployable as a Docker container, which is not implemented by the integration team. Hence the integration team has succeeded in partially fulfilling this requirement.

3.5 Application component concepts and constraints

- Service Contracts
  The current implementation of the support research system currently makes no use service contracts for service providers. The service contracts are required by the mock implementation as well as the default implementation to allow one to switch between the different implementations using dependency injection. In the current code base this will not be possible as dependency injection is currently using the class type of the bean, and hence one will not be able to switch the mock implementation with the real implementation.
- Stateless Session Beans
  All beans implemented by the implementation team are currently set up as stateless session beans, as required by the Architecture requirment spesification.
- Java Entities
  Java Entities or so called Plain Old Java Objects are used to transport data between the database and business layer. The current Java entities used by the implementation team currently doesn't fulfil the requirements set out by the JAVA persistence API.

4 Persistence API

The current POJO's utilized by the implementation team currently doesn't comply with the requirements as per the Java Persistence API. Specifically the following requirements are not met

- The classes is not annotated with the javax.persistence.Entity annotation.
- The classes doesn't have a public or protected no-argument constructor.
- The classes don't implement or extend a base entity which implement the java.io.Serializable interface.

Further more, entities also doesn't specify a primary key using the javax.persistence.Id annotation.

The current system also doesn't implement two phase commits across all services modules, which implies that the implementation only partially satisfies the required specification since it does not properly support *transactions*.

5 Web Services Framework

5.1 Architecture Requirements

The team adheared to the public standard and open-source implementation of their RESTful services. Thus it will be easily maintained and itegrated.

5.2 Architecture Design

The team implemented their RESTful services almost perfectly to the desing that is stipulated with Jersey, ie grouping the Peoples services into a PeoplesResource class, then defining POST, GET and PUT actions. They include all their JSON @Consumes where necessary, and JSON @Produces, but where they fell short is defining the paths within the resource and specifically with GET. The implementation required was @Path("id"), the retrieving the object from the ID, they however are expecting the object to to passed through the request header.

6 Web Application Framework

6.1 Architecture Requirements

6.1.1 Access and Integration Requirements

The application is viewable on most well known browers.
The team implemented integration with the REST API, but there was no backend services to connect to, so they connected to mock REST objects.

6.1.2 Quality Requirements

The site is not fully usable.
Because they used to technology assigned to them Ember.js, and they created the application the correct way, it is very maintainable.
The bower and ember components with the template system used, the web interface is very flexible. As it is new technology.
Performance within a site that doesnt work as intended, is hard to gauge but, the technology stack provides great looking and very responsive website.

6.1.3 Architectural Resposibilities

Because the team used Emberjs with querying REST API and a template system. Most responsibilites have been met. As they provide servvices such as Navigation, Maintaing View State, User views, Integration with REST layer(only through mock objects) and Template Processing.

6.1.4 Architectural Constraints

The web interface is using open source technology such as Emberjs and Bower. And conforming to Public standards.

6.2 Architecture Design

6.2.1 Tactics

- Caching
- Templating
- Pre Compilation
- Automatic Model Population
- Virtual DOM

6.2.2 Frameworks and Technologies

The team succesfully utilized Ember.js and all of its components. They unfortunately did not complete all the pages, and did not successfully implement all tactics such as virtual dom, amp and caching, but they utilixed the most important aspect which is templates. Most components within Ember.js that address the responsibilites has not been met as they do not have a fully working system.

6.2.3 Concepts and Constraints for application components

The team has successfully implemented the MVC framework which Ember requires to use, such as the Models created, the Templates used and the Controller which does not connect directly to ORM but rather through REST points.

7 Mobile Application Framework

7.1 Architecture Requirements

The team adheared to the public standard and open-source implementation of their RESTful services. Thus it will be easily maintained and itegrated.

7.2 Architecture Design

The team implemented their RESTful services almost perfectly to the desing that is stipulated with Jersey, ie grouping the Peoples services into a PeoplesResource class, then defining POST, GET and PUT actions. They include all their JSON @Consumes where necessary, and JSON @Produces, but where they fell short is defining the paths within the resource and specifically with GET. The implementation required was @Path("id"), the retrieving the object from the ID, they however are expecting the object to to passed through the request header.

7.2.1 Quality Requirements

On the subject of quality:

1. The application could not start.
2. The application was developed using the Gradle build system with Android Studio specifically being the development environment. This does mean that the system could in theory, support maintainability as future teams would have easy access to all of the resources needed to maintain the system.
3. What little functions are implemented correctly do not receive a wrapped object, or even parameters, as the unit test is hardcoded in.
4. The android API selected was 6.0 or API level 23. While this is the very latest version possible, it is also the least flexible because only a handful of mobile devices currently support it.
5. While xml files describing the layout of multiple pages is present, they are never rendered and are imageless and nearly colourless.
6. The application could not function so the performance of it cannot be measured.

7.2.2 Architectural Responsibilities

The primary architectural responsibility, to read and display information from a database, is not met. The mobile application has only bits and pieces of functionality and even then only on a local scale within objects/methods, never is information read from or sent to a database.

8 Android Application Framework

8.1 Architecture Requirements

8.1.1 Access and Integration Requirements

The mobile application was not viewable from multiple, or any, contexts. It did not render on any Android device emulator nor was viewable from any phone. There was implementation of REST API but no backend services meant that this what functionality existed was as a result of the provision of mocked REST objects.

8.1.2 Quality Requirements

There were a number of quality requirements issues present:

1. The application was not useable at all.

2. The application was developed using the Gradle build system with Android Studio specifically being the development environment. This does mean that the system could in theory, support maintainability as future teams would have easy access to all of the resources needed to maintain the system.
3. The android API selected was 6.0 or API level 23. While this is the very latest version possible, it is also the least flexible because the fewest number of mobile devices currently support it.
4. The performance of a non-functioning application cannot be accurately gauged.

8.1.3 Architectural Responsibilities

The primary architectural responsibility, to implement a REST API querying system, to ensure a stateless client, was not met. As such, no services could have been accessible to the mobile client nor processed. What was evident,however,was local capacity to realise those services but again, that falls short of the initial requirement.

8.1.4 Architectural Constraints

8.2 Architecture Design

8.2.1 Tactics

- Caching
- Templating
- Pre Compilation
- Automatic Model Population
- Virtual DOM

8.2.2 Frameworks and Technologies

1. **Jasper Reporting:** Jasper Reporting is a framework to provide an efficient and effective method of wrapping data for the purposes of report presentation. You can specify the report structure using XML and then populate the XML skeleton using requests for data from the actual database. This allows you to serialize the report as a JSON object and send them to the client that can unpack and process the report in any format that they would like. However, the Android team did not use Jasper whatsoever, cannot unpack reports to render to the device, or generate reports. It is clear that there is no functionality to receive and process Jasper reports sent by the server. This does not comply with the architectural requirement to use Jasper reporting.
2. **Gradle:** The project has a Gradle build file that enables it to be built and constructed by the Gradle build system and this includes compilation of source files and testing of said files. The Gradle build file is correctly set up. When asked to build the project, it performed all the correct steps, including compilation, testing and initialization. Furthermore, the build file has the correct dependencies. However, because the project had compilation errors, code had to be added and/or commented out in order to get Gradle to run. When Gradle did run, it ran properly.
3. **Android Studio:** The project was developed using Android specifically with the Android Studio Development Environment.
4. **Android SDK API:** The choice of Android SDK API is an issue. Although chosen, and certainly a valid Android SDK API, API 23, or Android 6.0, is the very latest version of Android and is only supported on the most high-end devices. As a result, this means that there is an extremely limited scope for client phones as few phones will readily have this API. This is compounded by the fact that this was an unnecessary choice as all of the features implemented or required by the Android API would have been provided by API level 10(Android 2.3.3) or Android 3.0. Either API provides all of the necessary functionality but is far greater in scope as many devices are available to run this API as all devices are backwards compatible.
5. **Requests and Responses:** According to the architectural specification, the system is supposed to be RESTfull and stateless. Both the client and the server should only be able to communicate via HTTP requests, specifically GET and POST. However, the Android interface has no access to the server at all, let alone through HTTP requests. This does not comply with any of the requirements on a fundamental level, as the client is completely isolated from the server, defeating the purpose of a client-server model.

8.2.3 Concepts and Constraints for application components

9 Reporting

The current code base of the support research system currently has no reporting functionality, hence the architecture specification of needing to use Jasper Reports for reporting has not been satisfied.