
UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

# Software Requirements Specification
# and
# Technology Neutral Process Design
# Research Paper Management System

## University of Pretoria

Team Echo

| **Surname, First Name (Initial)** | **Student Number** |
|---|---|
| Bode, Elizabeth (EF) | 14310156 |
| Bondjobo, Jocelyn (JM) | 13232852 |
| Broekman, Andrew (A) | 11089777 |
| Loreggian, Fabio (FR) | 14040426 |
| Schutte, Gerome (GC) | 12031519 |
| Sefako, Motsitsiripe (MG) | 12231097 |
| Singh, Emilio (E) | 14006512 |

# Table of Contents


**1 Vision** **1**

**2 Background** **1**

**3 Architecture requirements** **1**

3.1 Access channel requirements . . . 1

3.1.1 Human Access Channels . . . 1

3.1.2 System Access Channels . . . 2

3.2 Quality requirements . . . 2

3.2.1 Performance . . . 2

3.2.2 Reliability . . . 2

3.2.3 Scalability . . . 3

3.2.4 Security . . . 3

3.2.5 Flexibility . . . 4

3.2.6 Maintainability . . . 4

3.2.7 Auditability/Monitorability . . . 4

3.2.8 Integrability . . . 5

3.2.9 Cost . . . 5

3.2.10 Usability . . . 6

3.3 Integration requirements . . . 6

3.4 Architecture constraints . . . 6

3.4.1 Representational State Transfer (REST) . . . 6

3.4.2 Service Oriented Architecture (SOA) . . . 6

3.4.3 Framework . . . 7

3.4.4 Dependency Injection (DI) . . . 7

3.4.5 Unit Testing . . . 7

3.4.6 Architectural Patterns . . . 7

**4 Architectural tactics or strategies** **9**

**5 Reference architectures and frameworks** **9**

**6 Access and integration channels** **9**

**7 Technologies** **9**

7.1 Backend System . . . 9

7.1.1 Programming Languages . . . 9

7.1.2 Frameworks . . . 10

7.1.3 Libraries . . . 10

7.1.4 Database System . . . 10

7.1.5 Operating System . . . 11

7.1.6 Dependency Management and Build Tools . . . 11

7.2 Web Interface . . . 11

7.2.1 Programming Languages . . . 11

7.2.2 Frameworks . . . 11

7.2.3 Libraries . . . 11

7.2.4 Database System . . . 11
7.2.5 Operating System . . . 11
7.2.6 Dependency Management and Build Tools . . . 11
7.3 Android Client . . . 12
7.3.1 Programming Languages . . . 12
7.3.2 Frameworks . . . 12
7.3.3 Libraries . . . 12
7.3.4 Database System . . . 12
7.3.5 Operating System . . . 12
7.3.6 Dependency Management and Build Tools . . . 12

**8 Functional requirements and application design 13**
8.1 Use case prioritization . . . 13
8.1.1 Critical . . . 13
8.1.2 Important . . . 13
8.1.3 Nice to have . . . 13
8.2 Use case/Services contracts . . . 14
8.2.1 User Login - User holds no tokens . . . 14
8.2.2 User Login - User holds access and refresh tokens . . . 14
8.2.3 Log Out User . . . 15
8.2.4 Register User . . . 15
8.2.5 View/Edit User . . . 16
8.2.6 Create Paper . . . 17
8.2.7 Edit Paper . . . 17
8.2.8 View Paper . . . 18
8.2.9 Create Author . . . 18
8.2.10 View Author . . . 19
8.2.11 Edit Author . . . 19
8.2.12 Add/Remove Author to Research Paper . . . 20
8.2.13 Create Research Group . . . 21
8.2.14 View Research Groups . . . 21
8.2.15 Edit Research Groups . . . 22
8.2.16 Add/Remove Users to Research Groups . . . 22
8.2.17 Import Historical Paper . . . 23
8.2.18 Generate Bibliography . . . 23
8.2.19 Create Venue . . . 24
8.2.20 View/Edit Venue . . . 25
8.2.21 View Log . . . 25
8.3 Required functionality . . . 26
8.3.1 Login with Token . . . 26
8.3.2 Login without Token . . . 26
8.3.3 Edit Research Group . . . 27
8.3.4 Edit User . . . 27
8.3.5 View User . . . 27
8.3.6 User Log Out . . . 28
8.3.7 Create Paper . . . 28

8.3.8 View Paper . . . 28
8.3.9 Edit Paper . . . 29
8.3.10 Register User . . . 29
8.3.11 Create Author . . . 30
8.3.12 View Author . . . 30
8.3.13 Edit Author . . . 30
8.3.14 Create Research Group . . . 31
8.3.15 View Research Group . . . 31
8.3.16 Import Historical Paper . . . 32
8.3.17 Generate Bibliography . . . 32
8.4 Process specifications . . . 32
8.4.1 Login with Token . . . 32
8.4.2 Login without Token . . . 33
8.4.3 Edit Research . . . 33
8.4.4 Edit User . . . 34
8.4.5 Log Out . . . 34
8.4.6 Register User . . . 35
8.4.7 View User . . . 35
8.4.8 View Author . . . 35
8.4.9 Create Author . . . 36
8.4.10 Edit Author . . . 36
8.4.11 Create Research Group . . . 37
8.4.12 View Research Group . . . 37
8.4.13 Import Historical Paper . . . 38
8.4.14 Create Paper . . . 38
8.4.15 View Paper . . . 39
8.4.16 Edit Paper . . . 39
8.4.17 Generate Bibliography . . . 40
8.5 Domain Model . . . 40

**9 Open Issues** **41**
9.1 GitHub Repository . . . 41

**Appendices** **41**

**Appendix A Data Dictionary** **42**

**References** **43**

# 1 Vision

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.

The department currently consists of various research groups, with various contributors, all producing research publications which should be tracked. As the University of Pretoria is rewarded for research produced, it is important for both the heads of research groups and the Head of Department to monitor the progress of current publications, and to gather other metrics to assist in other departmental planning.

The most important requirements as set out by the client are:

1. Monitoring the research output by the department through metrics by the Head of Department and heads of the research groups
2. Tracking the status of research publications
3. Being able to produce customized reports based on user defined values

# 2 Background

This project was commissioned based on the following three problems faced by the client

- Limitations of current system:
  - The current system used by the client involves utilizing one common Microsoft Excel spreadsheet document to keep track of the status of current research publications. A problem with this solution is that it causes data integrity problems due to the fact that it is not scalable.
- Increased productivity and transparency
  - A common issue faced by researchers in the Computer Science department is that an increased workload at due dates leads to the spreadsheet becoming unstable and subsequently crashing. Necessary data recovery on the spreadsheet leads to increased downtime, which, while also necessitating repeated data entry, decreases user productivity. Clearly, not a user friendly system, causing many a frustrated user.
- Monitoring and management of funding
  - Currently the department receives funding based on research produced. The funding model, and hence, the amount, is based on various factors pertaining to the research publications, such as whether it is published, presented at a conference and so on. Therefore, having access to projections of the number and type of publications that are to be produced in coming years is very important for strategic departmental planning, as is monitoring historical performance of research groups and individual researchers.

# 3 Architecture requirements

## 3.1 Access channel requirements

### 3.1.1 Human Access Channels

The human access component must allow for both a desktop web client and an Android mobile client allowing the user to manage the metadata associated with research papers they are involved in.

The web client will be the primary contact point for users of the system as most research papers are created, edited and maintained on personal computers that either owned by the contributor in question or the University of Pretoria. The web client will be based on an MVC based SPA architectural pattern.

The Android application will be made available in the Android App Store to allow contributors to the system from outside the University of Pretoria to easily gain access to the system. The Android application should be accessible by all mobile devices running Android version 4.2 and upwards.

### 3.1.2 System Access Channels

All system access will be made over a uniform interface which will assist in faster development time, ease of maintenance and lower overall cost associated with the system. The common interface that will be used, will be a RESTful API based on the dissertation Architectural Styles and the Design of Network-based Software Architectures of **fielding** All client side interaction will be done via the web client or Android application which will in turn use the RESTful API set of the system to accomplish the given task.

## 3.2 Quality requirements

### 3.2.1 Performance

1. **Description**
   Application performance can be defined as the amount of work that can be accomplished by an application in question in a measured time interval. The time interval is normally measured in seconds, where the amount of work can be defined as the throughput, latency or data transmission time.

   - **Throughput**
     The number of requests and responses which can be processed by the system in a given time interval.
   - **Latency**
     A time interval measured as the time it takes to service a request.
   - **Data transmission time**
     The time it takes to transmit a request to the server from a client or a response from the server to the client. This is heavily influenced by the size of the request, as well as the quality of the network medium.

   The aim for this system, is to increase the throughput and decrease the latency. As the developer has no control over the network medium used, he/she must aim for a minimal request and response payload as to decrease the data transmission time.

2. **Justification**
   The current system suffers from performance issues, and hence, it is important for this new system to address these issues adequately. Our aim for this system is to increase throughput, decrease latency and data transmission time. This will ensure we have a system that is responsive at all times, including peak times and delivers an excellent user experience.

3. **Requirements**

   - Network requests must be benchmarked, monitored and aggregate statistics about network requests must be made available.
   - Function calls must be timed and benchmarked and this data should be logged.
   - Precompiled SQL must be used in statements.
   - Network responses should be cached on server side to lighten the load on database as well as decreasing the round trip time of request - response.

### 3.2.2 Reliability

1. **Description**
   The design of this system is largely based on the fact that the current system used by the client is not reliable and suffers from system crashes and data loss. The newly designed system needs to be accessible from both inside University of Pretoria campuses as well as from other networks, especially on other campuses on the TENET network. The system should be reliable in both its accessibility and in the data that the system will use to calculate contributor metrics.

2. **Justification**
   The current system is largely unreliable in that it crashes under large workloads, and in that, because only one user can work on it at a time, many users get denied service during work critical times when workloads are high. This causes a detriment in user productivity. To ensure that the system can be used confidently as a tool to increase productivity and ease the work process, the development aim must be for the system to be as reliable and accessible as possible.

3. **Requirements**

   - To enable offline activity and access of data from the Android app, data synced must be made differentiable with timestamps.
   - The system must resolve synchronization conflicts using timestamps.
   - Hot swapping of system modules should not affect system service reliability.

#### 3.2.3 Scalability

1. **Description**
   Scalability refers to the application in question's ability to handle an above normal workload for extended time periods and the mechanisms employed to facilitate this. The client has specified minimum conditions under which the system needs to function. However, the client has indicated that due to the business requirements, the system will experience higher workloads during the end of the month.

2. **Justification**
   The system needs Scalability because it needs to support 10 research groups, with 100 members each working with 50 concurrent users. But these requirements may increase in future and hence the system should be able to handle additional loads.

3. **Requirements**

   - The system should be able to handle 10 research groups with each research group containing 100 members.
   - The system should be able to facilitate 50 concurrent user sessions at any time.

#### 3.2.4 Security

1. **Description**
   Security in application software refers to authentication, authorization, data security and accounting. Authentication refers to the systems' ability to provide a way of identifying a user, normally with a combination of a user name and password, or by using access tokens. Authorization refers to the systems' ability to determine whether the user in question has the required authority to execute certain commands or tasks. Data security refers to data only being accessible through authorised channels. Finally if the user has been authenticated and has the required authorization then accounting needs to be done to be able to determine how the users action has changed the system.

2. **Justification**
   Security is a important aspect of any software product. In terms of information security, we are concerned about integrity, availability, confidentiality and non-repudiation in that order.

   The reason we are most concerned about data integrity is that this data will be used to monitor and determine performance of researchers as well as assist in departmental strategic planning, which implies that the system must be available at all times to the heads of the research groups and the head of the department.

   Following this, is the aspect of data confidentiality, as we want to ensure passwords of the users stay secure as well as the metadata of research papers. The storing of metadata and subsequent leaking of this metadata can lead to other researchers publishing and patenting ideas first.

   Finally, as this system is used to measure performance, it is important that non-repudiation of information is available, as researchers must be accountable for any information changes they make.

3. **Requirements**

   - System should be resistant to SQL injections.
   - A user group hierarchy system should be used to manage user access rights.
   - Authentication credentials such as username and password should not be stored on Android user device. A token based authentication approach should be utilized so that only tokens are stored on end user device, allowing for the easy revocation of device access if a device is lost. Available technologies, such as OAuth, should be utilized.
   - Password hosting with a unique salt for each user should be used.

- A key derivation function should be used with passwords.
- After N invalid attempts, account should be locked for M min, where N and M would be agreed upon customer and developer based on current best practices.
- Database access must require authentication.

#### 3.2.5 Flexibility

1. **Description**
   Flexibility refers to the ability of the system to be changed dynamically either by hot swapping certain components in a live system or by extending the system with some kind of plugin.

2. **Justification**
   Flexibility is important for any system. A non-flexible system is restricted to using technologies that were hard coded into it, and this necessitates, at best, large scale refactoring every time an upgrade is available since new technologies need to be reintegrated, makes adding new features tedious, and risks the system becoming archaic. A flexible system requires minimum effort to upgrade and expand, allowing for the system to easily grow in usefulness and function beyond the original vision, and is directly in line with the requirement for the system to be cheap to maintain and upgrade, not just financially but also in labour.

3. **Requirements**

   - The system should allow unregistered users to use existing third party web service login credentials to register or log in to the service.
   - The system should be decoupled from the database technology it uses and allow the client to select and change the database it uses in future.
   - Authentication mechanisms used should be decoupled from the system, allowing them to be interchangeable.
   - Modules should be decoupled from one another, allowing the system to be extensible without a break in service which is achieved by integrating new modules and swapping out existing ones.

#### 3.2.6 Maintainability

1. **Description**
   The system is to be designed in such a way that it is easily updated, modified or extended by the client in the future. In order to achieve these requirements, design patterns and best practices such as coding style guides are normally used to ensure uniformity and modularity across the system.

2. **Justification**
   Many systems require regular changes, not because they were poorly designed or implemented, but because of changes in external factors. For example we might need to update informations to meet the Computer Science laws and regulations.

3. **Requirements**

   - All code should be documented in the applicable language documentation framework, such as JavaDocs for a Java based system, DOxygen for a C/C++ based system, etc.
   - A coding style guide/manual should be set up and associated with the project, such that all developers use similar coding styles and conventions, to allow for more readable code that is easier to maintain.
   - System should be separated in distinct, concise and independent modules relating to separate concerns, to allow for easier maintenance.

#### 3.2.7 Auditability/Monitorability

1. **Description**
   The system is to be designed to be verbose and transparent in its workings, and to ensure maximum data security, to allow role players to have insights into how the system is used and how it may be improved. These requirements are achieved by making the maximum amount of relevant data available to authorized users, logging performance critical information, and by enforcing strict constraints on the data that is stored.

2. **Justification**
   This is an important process in Software Engineering, where all the informations must be correct so requiring all the developers to see who made changes and when so that consistency must be kept in order to keep the database accurate and reliable.

3. **Requirements**

   - Data in database should always be consistent. This implies that all data should adhere to constraints placed on the data by the data model, such as regex patterns, minimum and maximum length, non nullable fields, etc.
   - No data should ever be deleted.
   - It should always be possible to see which user created which objects at what time.
   - It should always be possible to see which user last changed objects and at what time.
   - An immutable log of user actions should be kept and should only be accessible by administrative staff.
   - Logging of stack traces and crash analytics should be implemented in the mobile client, to ensure the developers can see to client reliability.

#### 3.2.8 Integrability

1. **Description**
   The system should allow for future external integration with other platforms such as security authentication providers, other external research meta-data databases etc.

2. **Justification**
   The system necessitates integrability to allow for maximum usability, as the integrability of the system is directly related to how usable it is. To be usable, the system must allow for easy migration, not just from previous systems, but also to future systems and future data storage mediums. The usability is also largely determined by how well the back-end system integrates with front end clients, and which clients are supported.

3. **Requirements**

   - The system should allow technology neutral importing and exporting of data.
   - The back-end system should integrate with a desktop web client and Android mobile app clients.
   - The system should be able to integrate with different back-end authentication services.

#### 3.2.9 Cost

1. **Description**
   The cost of the system entails any initial expenses as well as any ongoing expenses which the client may incur at some point. Such expenses arise from software licenses, external computing resources required as well as future maintenance of the system in terms of time.

2. **Justification**
   The expenses made and software licenses cost must be taken into account in order to set the price of the software.

3. **Requirements**

   - System should be cheap to operate, maintain and extend. If the quality and maturity of technologies available allow it, technologies used must be freely available/usable.
   - As far as possible, open source compatible, mature technologies should be used, to ensure system stability as far as possible.

#### 3.2.10 Usability

1. **Description**
   Usability refers to ease with which humans, and to a lesser extent, servers, interact with the system in question. Usability can be measured in various ways such as using quantifiable scientific measures or more subjective measures with a key question point being if the API follows conventions and so on.

2. **Justification**
   It is important that the new system is usable as it is a user-centric system. Ensuring that the system is usable will ensure that users capture accurate and correct information into the system which will for better performance and departmental strategic planning. As this system will also be used by parties outside of the Department of Computer Science of the University of Pretoria, it is important that the system conveys a professional image, as this will reflect on the image of the University of Pretoria.

3. **Requirements**

   - Each view in the desktop and mobile clients should be related to a single topic only.
   - The web client should render properly and be fully functional in modern web browsers.
   - Mobile devices running Android 4.2 and upwards should be fully supported.
   - Material design UI guidelines prescribed by Google must be used, to ensure that the clients feel modern and familiar.
   - Android app and web client should support the full back-end API specifications.
   - The user must be allowed to elect whether they would like their data to be available offline.
   - Mobile client users should be able choose between downloading data over Wi-Fi or 3G.

### 3.3 Integration requirements

The web interface should be accessible to most modern PCs with a modern browser. The website will be making use of an SPA MVC based design which will query a RESTful API, resulting in a very light, mobile and responsive web interface that will it to work across various devices.

The Android application will be built to work on all Android version 4.2 and upward devices. The mobile application on the Android device should be sensitive to the amount of data that is transmitted over the network due to the high cost of mobile internet in South Africa. To allow for minimal communication but maximum functionality, the MVC based design will be followed in the Android application design. Furthermore, the use of accessing RESTful APIs will allow objects to be transmitted on the wire with a very minimal representation, thereby saving on client bandwidth costs.

The RESTful API will be implemented using the HTTP protocol as a high level protocol. The reason that HTTP is chosen is that it is a widely supported and deployed protocol, allowing a lot of flexibility. Furthermore, most JavaScript libraries, and the Android platform itself, have excellent support for RESTful APIs over HTTP.

### 3.4 Architecture constraints

The archicterural constarints are used to improve the architectural analyses of quality characteristics of the software to be developed. Our system architecture will be based on the following technologies which will be used:

#### 3.4.1 Representational State Transfer (REST)

REST which depends on stateless client server and communications protocols, and uses the HTTP protocol in most cases. It is an architectural style used to design networked applications. All the four CRUD operations are used in REST.

#### 3.4.2 Service Oriented Architecture (SOA)

SOA makes it easier for software components on computers that are connected on a network to work together with no need to make changes to the underlying program itself. The concept of service is what SOA is based on.

### 3.4.3 Framework

- The Spring framework: is a technology that is associated with developing secure websites and interacting with databases in conjunction with Hibernate. Spring is simply a usable API, with the idea being for the developer to represent Java objects with Spring Beans. The Spring Framework is an inversion of control container which is just a configuration of application components and lifecycle management of Java objects, it is usaully done though dependency injection. Spring is especially an application framework for the Java platform.
- Liquibase: allows tracking, management of database schema changes to relational object databases.
- Apache Log4: allows total control over log statements that are output. The system's configuration is available at run-time using external configuration files, fulfulling the quality requirements of flexibility and auditability.
- Hibernate ORM: provides various a feature such as SQL generation and is an object relational mapper framework for Java, which allows the mapping of Java classes onto relational database tables.
- Java Jersey: provides RESTful web services for Java.

Any Java application can use the framework's core features , but for building web applications on top of a Java Enterprise Edition platform some extensions exists. Even though, the framework does not impose any specific programming model. Therefore, spring can helps us achieve integrability and maintainability by using design patterns and any programming model to ensure uniformity across the system.

### 3.4.4 Dependency Injection (DI)

DI will eases software testing and re-usability of software by using a design based on independent classes/components thus achieving the usability requirement.

### 3.4.5 Unit Testing

Unit Testing to automate the following tasks: compilation of Java source code, running test cases and generating project documentation. It will be used as well because it gives us the following benefits

- Reduced system failure risk,
- Rapid feedback on developed components,
- Improved Maintainability due to unit testing,
- Improved Reusability leading to less code being developed and maintained.

The Java programming language will be used to build this system. The system will be deployable on Windows operating systems and later on Android devices.

### 3.4.6 Architectural Patterns

- **Authentication Enforcer Pattern**
  The authentication enforcer pattern is focused to provide a centralized managed authentication mechanism, to verify the identity of users and encapsulating the details of authentication. This patterns assist in removing security checks from business logic code, thereby providing cleaner, more robust and easier to maintain code.

  Further, since authentication is centralized, this pattern further allows changes to be easily implemented in terms of the authentication procedure, such as using a different authentication flow, or even mechanism, such as authenticating the user against an external provider.
- **Authorization Enforcer**
  Similar to the authentication enforcer pattern, this pattern provides a centralized managed authorization mechanism, thereby abstracting authorization code away from application code. With this pattern, it assists the system in authorizing users, i.e. to determine whether the current user has the required roles or privileges to perform a certain action. All of the benefits referred to in the authentication enforcer pattern are also applicable here.

- **Client-Server Architectural Pattern:**
  A network technology in which each computer connected on the network is either a server or client is called a client server architecture.

  1. Servers: are powerful computers which are used to process the client requests.
  2. Client: is simply a computer connected on the network which is used to send request for some resources to the server.

  Requirement of client server: It models work in a networked environment therefore the processing of an application distributed between the client and the server. Function of client server: An interface/form is provided by the client to allow a user to request services of the user and to display the results the server returns.

  Client-Server Architectural Pattern is used because we need to have reliability on the system. Data can be retrieved for further processing from any computer connected on that network.

- **Layered Architectural Pattern:**
  A design pattern in which software is divided up into individual layers by functionality. The layers interact with one another via requests and responses, and there are typically four of these layers:

  1. Presentation/view: The user interface that the user interacts with, such as a desktop client.
  2. Application/controller: The service API that the presentation layer interacts with in order to perform system functions.
  3. Business logic: The functions that get called by the service API, to interact with system data.
  4. Data access: The layer that provides data access to the business logic layer

  This pattern is used because it lends itself well to decoupling software modules from one another, and allows for separation of concerns. At the very least, the view layer is decoupled from the business logic, and the controller layer is decoupled from the data access. Furthermore, when each layer object realises a contract, all layers are completely decoupled from one another, and allows for layer objects of the same layer type to be interchanged without any hassle.

- **Representational State Transfer (REST) Architectural Pattern:**
  REST is an architecture pattern for designing networked applications. The idea is instead of using complex mechanisms such as SOAP, CORBA or RPC to connect between machines or to make calls between machines, simple HTTP is used.

  The Uniform interfaces which is combined of resource names and HTTP, GET, DELETE, POST, PUT. A REST service is:

  1. Language-independent (C# can communicate to Java, etc.),
  2. Can be used with firewalls,
  3. Platform-independent (MAC, Windows, UNIX) and,
  4. Standards-based (runs on top of HTTP)

  Representational State Transfer Architectural Pattern can be used because we need flexibility on our system like to be platform independent.

- **Secure Data Logger Pattern**
  A requirement stated by the client is that the proposed system should contain a full audit trail functionality, which would allow administrators to view any action that executed by users, as well as the modification that execution had on the data.

  For this requirement the Secure Data Logger Security Pattern can be utilized. This pattern aims to log application system log to a secure tamper-proof log system, which would prevent users from modifying the audit trail. This pattern also goes further by ensuring the integrity of the log message, which can be achieved by various means, such as cryptographically signing the message, or encrypting the message before it is sent to the store.

  This pattern provides two important security features, depending on how the logging system, as well as the message are secured in the system. The first protection, is that should a malicious user gain access to the logging system, they will be unable to use the data to plan further attacks on the system. The second protection provided by this pattern, is that any modification of the log data will be detectable by administrators.

- **Services Orientated Architecture/ Microservices Architecture:**
  An architectural pattern in software design in which system use cases are divided into one or more service operations, and these service operations are then implemented, either individually or combined with other service operations, by a reusable SOA service. The SOA architecture itself comprises five layers:

  1. Consumer interface: The GUI for end users accessing application services
  2. Business process: The representation of system use cases
  3. Services: The consolidated inventory of all available services.
  4. Service components: The reusable components used to build the services, such as functional libraries.
  5. Operational system: Contains data repository, technological platforms, etc.

  Not to be confused with an API, this architectural pattern provides an aggregated collection of services which implement the use cases of the system. The user or developer themselves can choose which of these available services to call on. This architectural pattern was chosen because it is built around the principles of decoupling modules. Service objects must be decoupled from one another and can be distributed on different machines and implemented using different technologies, but implement a single interface, and communicate with one another using well defined interfaces, either locally or over a network. This also lends itself well to system reliability and scalability. More than one system provider may be used, enabling the system to switch to another service provider should one fail, or instantiating more service providers if system load increases.

# 4 Architectural tactics or strategies

# 5 Reference architectures and frameworks

# 6 Access and integration channels

# 7 Technologies

## 7.1 Backend System

### 7.1.1 Programming Languages

- Java

**Description** Java in this instance refers to the programming language. Java is an object-oriented language that supports concurrency and classes.

**Performance** The reader may note that historically, Java has been considered to have worse performance than languages such as C and C++. Primarily, this is due to the fact that Java programs run on the Java Virtual Machine as opposed to the computer's processor. However, to assay those fears, the performance of Java in recent versions has improved dramatically by the use of the Just-in-Time compilation. Combine this with the ability to compile Java code once and then run the program without recompilation on a different computer, means that Java is still an ideal contender for a programming language for use in the system.

**Scalability** When referring to the Scalability requirement, Java has relatively high performance when used for High Performance Computing, HPC, and is similar to Fortran on the more intensive benchmarks. However, there does exist a scalability issue for the Java Virtual Machines to perform intensive communication on grid networks. To again assay fears of viability in this system, much of the scalability issue with Java Virtual Machines lies in the hardware setup and not on the software side.

**Flexibility** For the Flexibility requirement, Java excels as it operates on the WORA or Write Once, Run Anywhere principle. Java code can run on all platforms that support Java without the need for a recompliation. This is accomplished via the fact that Java applications are compiled to bytecode. This bytecode can run on any Java virtual machine and is totally independent of the computer architecture. This means that Java code, once writtten, can be easily and quickly ported and deployed throughout the various system components.

**Cost** The latest version of Java is Java 8 which is supported for free by Oracle meaning that its use in this system does not contribute towards system cost.

**Usability** From a programming perspective, Java offers a number of features that make it an adaptable and useful language for programming. One of the core features is that Java provides an automatic memory management service or automatic garbage collector. This means that the programmers are freed from the concerns and complexities of memory management, and the contiguous issues associated therein.

From a syntactical perspective, Java offers a similar syntax to Frameworks C and C++ meaning that learnability of Java is typically much lower, especially with the use of IDEs such as Netbeans or Eclipse.

### 7.1.2 Frameworks

- Spring Framework
- Spring Boot
- Spring Cloud
- Spring Data
- Spring Security
- Spring Social
- Hibernate ORM
- Liquibase

**Description** Liquibase is an open source, database independent library that makes provisions for tracking, managing and applying any database schema changes. Liquibase will be invaluable in ensuring that we can track and manage the growth and use of whichever database we choose to use.

**Scalability** Liquibase supports over 10 database systems which provides our system with a lot of potential growing space in terms of database systems that it will use.

**Maintainability** In terms of Maintainability, Liquibase supports a variety of functions that allow for developers to maintain databases. These functions include rollback x changes to the database or even to a particular time. This means that even over time, should reverting to an earlier database state be needed, it can be done allowing for easier maintenance in the long term.

**Auditablility/Monitorability** With Liquibase, we are able to perform a number of functions that provide for necessary monitoring/auditing functions. These include database difference reports, the ability to generate documention for database change, and the ability to split change logs into multiple files, enabling easier management of changelogs for auditing purposes.

**Cost** Liquibase is open source and that means that using it contributes nothing to software costs of the system.

**Usability** Liquibase does indeed support the Usability requirement because it can be executed via a variety of means such as command line, Apache Ant and Maven or the Spring framework.

- SLF4J
- Apache Log4J
- Java Jersey

**Description** The Jersey RESTful Web Services framework or Java Jersey for brevity, is an open source framework for developing RESTful Web services in java. It provides its own API that extends the existing JAX-RS toolkit as well. This framework is considerably useful in our situation as we are using both Java and a RESTful API approach to making the system.

**Flexibility** In terms of the Flexibility of Java Jersey, as the underlying language of the system will be Java, Java Jersey will contribute to the Flexibility requirement as the latest version, 2.7, Java Jersey can compile and run applications using at least Java 7. Java Jersey is built and installed using Apache Maven. This means that applications that use Jersey, or depend on Jersey modules also need the application dependencies of the 3rd party modules that Jersey itself depends on.

However, since Jersey is designed as a pluggable component, this means that it can be plugged into different component architectures but at the cost of requiring that external Jersey dependencies needed will have to be included in the application dependencies as a whole.

**Cost** The Jersey framework is open source so no additional project costs will be incurred by its use.

**Usability** A core, and vital feature, of Swagger is the ability to generate specifications that describe REST APIs in a format that can be read by both machines and humans. This ease of use, especially between human and computer is a vital service that needs to be in play as this system will be based on a REST API.

- Swagger

**Description** Swagger is a framework that is used for describing and visualizing RESTFul APIS. It defines a standard, language-agnostic or language independent, interface for REST APIS. This enables users, both human and computer, to find and discover service capabilities without requiring source code access or by network traffic inspection. This is useful because it means that we will have a means both computers and developers can find service capabilities without violating access to certain code components or contributing unnecessarily to network traffic.

**Maintainability** The documentation generated by, and using, Swagger is integral towards the Maintainability requirement of the system. This documentation that is both machine and human readable will ensure that future machines and computers, as well as future developers, will be able to read the system documentation and maintain the system as a result.

**Cost** Swagger is an open source specification whose use is freely available.Hence it will not contribute towards the cost of the system as it can be freely used with commercial license provisions.

**Usability** A core, and vital feature, of Swagger is the ability to generate specifications that describe REST APIs in a format that can be read by both machines and humans. This ease of use, especially between human and computer is a vital service that needs to be in play as this system will be based on a REST API

### 7.1.3 Libraries

- Apache FOP

**Description** Apache FOP or Formatting Objects Processor is a Java application that converts XSL formatting objects to printable formats such as PDF. The system will likely generate a vast amount of documentation and a program that allows us to convert this documentation into a format optimised for printing would save us hassles from getting the physical system documentation stored properly.

**Flexibility** Apache FOP is an output independent formatter which means it is able to support a wide variety of output formats, or printable formats, such as PDF, PS,PCL and AFP. Furthermore, the input for FOP is a standard XSL-FO file which is useful as the XSL-FO is also an XML file and that can be created from a variety of sources, adding greatly to its flexibility as a program.

**Cost** Apache FOP is open source software and is freely available. Usability

#### 7.1.4 Database System

- PostgreSQL

**Description** PostgreSQL or Postgres is an object-relational database management system. It supports the SQL standard and serves to store data securely. As it has high utility and functionality, it will be ideal for the database management system to handle the database needs of our system.

**Scalability** PostgreSQL satistifies the Scalability requirement rather nicely. It can handle a variety of different workloads, from single machine applications to much larger web applications with many concurrent users. This means that the program can be easily used to ensure that our sysstem can scale to the workload that is required of it.

**Security** Security in postgreSQL is handled internally on a per-role basis, to individuals or groups. This means that roles, essentially users, can be applied and revoked dynamically and on any object, down to a column level. This provides it with a dynamic and flexible security service that allows us to make use of various levels of security for the various roles that exist in our system in terms of user base.

Furthermore, PostgreSQL supports a wide variety of external authentification mechanisms such as LDAP or SSPI. It gives us further control on which users can connect to which of our databases as well as where they can connect from, IP address/domain socket and whether the connection must use Transport Layer Security or TLS.

**Flexibility** PostgreSQL has several interfaces as well as wide programming language libraru support. It has both internal and external interfaces that allow it to be used in a variety of applications using a variety of languages.

Additionally, PostgreSQL supports a large variety of native data types from Boolean types all the way to JSON.

Adding to this is the use of the Foreign Data Wrapper that increases the flexibility of our system as using PostgreSQL, our system can use regular database queries to join external data sources like normal tables.

**Integrability** PostgreSQL supports Foreign Data Wrappers or FDWs. This means PostgreSWL can link to other systems and retrieve data on those systems via the FDWs This FDW can take a variety of forms in terms of data source such as file system, another Relational Database Management System or even a web service. This allows our RDBMS to integrate with a variety of platforms and applications.

**Cost** PostgreSQL is freely available, open soure software that cocntributes no cost towards the system.

- Couchbase Server

**Description** Couchbase Server is a distributed multi-model NoSQL database optimised for interactive applications. In contrast to PostgreSQL, Couchbase is is document orientated database that makes use of no SQL standards.

There are a number of reasons to want to consider a NoSQL database and specifically with Couchbase Server, it is optimised for concurrent users and interactive applications. If the system does develop into one where concurrent access and interactivity are the primary functionality points, Couchbase Server would be an ideal database management system.

**Performance** Couchbase Server makes use of a cluster manager to supervise the behaviour and configuration of servers that comprise a Couchbase cluster. One of the services offered by this is high-speed failover which improves the system performance as it will automatically and quickly transfer control to a duplicate server on the cluster to replicated items as per requests.

**Reliability** Couchbase Server makes use of a "tail-append" storage design that is proof against data corruption, sudden loss of power and OOM killers or out of Memory Killers. This claim has the caveat that a sudden lose of power can still lose data if the power loss occurred in the middle of the disk transfer.

However, this aside, Couchbase Server provides great assurance that the data that we store will be protected and stored safely.

**Scalability** As mentioned above, Couchbase Server has great utility in supported concurrent user access and this means that the service will scale well when under heavy workload.

**Cost** Couchbase is licenced under the Apache Licence (Open source edition) and can be used without incurring additional system costs.

- Couchbase Sync Gateway

**Description** Couchbase Sync Gateway enables Couchbase database to reflect changes in an application's data in both directions. Bu this, it is mean that Sync Gateway stores data from the applications' on-device data in storage unit stored on the existing and scalable Couchbase Server technology.

It provides a number of useful features that when coordinated with Couchbase Server, extends much functionality towards web-based applications such as logging and user authentication. Of note is the ability to determine which data records in the database are relevant for which user.

**Reliability** The sync function provided by Sync Gateway is the core of the API. It provides a number of services that greatly contribute to the Reliablity requirement such as document validation or change authorisation.

**Scalability** Sync Gate is scalable as it makes use of the existing and scalable Couchbase Server technology.

**Security** In terms of the Security requirement, Sync Gate has one issue of contention. The sync function is not only able to do validation of data but it also authorises read/write access. This means that it needs to be highly monitored to ensure it prevents unauthorised access and threats. This will increase system complexity in terms of ensuring that authorisation protocols are in place but the usability that comes from using Sync Gateway with Couchbase Server may be worth the added cost.

**Flexibility** Sync Gateway is designed to be used by a variety of platforms like Swift and Android and Java. This means that we can use it for the Android application as well as the underlying system.

**Cost** As Sync Gateway is provided under an open source licence and as such, does not contribute to system costs.

#### 7.1.5 Operating System

- Linux

#### 7.1.6 Dependency Management and Build Tools

- Maven

### 7.2 Web Interface

#### 7.2.1 Programming Languages

- HTML
- JavaScript
- CSS

#### 7.2.2 Frameworks

- AngularJS

#### 7.2.3 Libraries

- Modernizr

**Description** Modernizr is a JavaScript library which is designed to detect HTML5 and CSS, specifically CCS3 in various browers. The use of this so as to prevent JavaScript from using features that have not yet been implemented yet or, where available, use workarounds to emulate them. Modernizr is chosen here for its ability to determine if features/functions are supported in web browsers to save on time when designing or improving the web interface.

**Performance** In terms of performance, which is defined as work done in a time interval, a library like Modernizr will improve our system performance because the library will prevent our system from using features and functions that are not implemented yet. By knowing, and thus preventing that use, our system will benefit from improved efficiency and thus provide better performance.

**Reliability** Reliability is the big quality requirement that Modernizr aims to satisfy. The ability to know which features the underlying languages in browsers have and do not have, and the ability to potentially make use of workarounds, means that the development team will always know the limits of what their JavaScript based components can and cannot do and this will also mean that they can develop workarounds and fall back functions for the functionality they require but are not yet developed.

**Scalability** Scability in this context is the ability of the application to handle above average workloads for an extended period of time. Modernizr is again useful because the ability to know what can be relied upon, in terms of functions provided by browsers, will enable the development team, to design using known functions, applications that can scale properly without having to rely on unknown or unreliable functions or components.

**Security** Security is an important component of any application, especially an application that is meant to be used to access highly sensisitive material. Security is aided by Modernizer in this regard because it allows the development team to see whichc functions their JavaScript components will have to rely on in HTML5 and CSS3 and this will enable them to plan their security features accordingly as they will know which functions will be supported and which functions will not be supported.

**Flexibility** Modernizr is important for the Flexibility quality requirement. If the system is meant to be dynamically flexible, it is important to be able to determine, quickly and accurately, which of the functions in our web system are available and which are not. In this way, Modernizr allows us to increase our flexibility because we are then able to design our system to expand around what functions are already supported and what functions will require fallbacks or workarounds.

**Maintainability** Modernizr enables the Maintainability of the system. It does so because our system, as it being maintained, will have a utility to determine if new functions, components or services offered make use of other functions that are not offered, implemented or available. In this way, we can ensure that system maintenance does not introduce additional system instability.

**Auditablility/Monitorability** Modernizr is essentially a colletion of tests to determine what a browser can and cannot do. These tests can be logged and included alongside system logs so that future development teams can trace the development of the web interface. Additionally, as Modernizr is itself a tool for testing, it allows the developers to monitor their system in terms of functions they have and functions they do not have.

**Integrability** Since Modernizr is used to determine what HTML5 and CSS3 functions are supported by a browser, it can be used by the development team to design a web interface that can integrate into additional platforms by providing the team with the ability to test for the functions supported by that platform and how much integration needs to be done to bridge the gap between the new platform and the existing one.

**Cost** Modernizr does not contribute towards the cost of the system as it is an open source JavaScript library.

- Karma

**Description** Karma is an information integration tool. It is designed to enable rapid and easy data integration, from a multitude of sources like databases, XML, JSON and so on. This information is integrated by modelling it according to an ontology, selected by the user, and facilitated by a graphical user interface. This model can then be interacted with to transform the data into a needed format.

Karma was chosen here for its great utility as an integration tool. It can integrate data quickly and easily from a variety of sources and this will mean that we can develop a web interface that is not so tightly bound to any one specific data format.

**Flexibility** Karma helps the system achieve the requirement of Flexibility because it enables the system to make use of any data source as Karma can be used to transform that source of the data into one that can be used by the system. This would mean, since users can define data transformation scripts, the system can be expanded to make use of any kind of data source that can be transformed with Karma.

**Integrability** Karma achieves the requirement of Integrability because it allows the users to transform data from a variety of sources into one specific format. This makes the system independent on any one specific source of data and means that it can integrate with a variety of data storage types as well as transform existing data for the purposes of exporting it from the system.

**Cost** Karma is open source software and as such, does not contribute towards the cost of the system.

**Usability** Karma is achieves the requirement of usability very easily. The interface is an easily manipulatable GUI that can store the transformed data in a database or in a RDF. This ease of use certainly means that should data transformation be needed for data capturing purposes, it can be done quickly and easily.

- Jasmine

**Description** Jasmine is an open source testing framework for JavaScript. Specifically, it is testing frame that is designed for performing unit tests. For reference, unit tests are when individual units of source code are tested.

**Flexibility** Jasmine achieves the requirement of Flexibility because, as indicated by one of its core development aims, it will run on any JavaScript-enabled platform. This capacity to run on any platform means that Jasmine, as a testing platform, it will enable programmers to perform unit tests free from specific testing environments or operating systems as it is cross-platform.

**Cost** As Jasmine is an open source framework, it will have no effect on the cost of the system.

**Usability** One of the core design focuses of Jasmine is that it must not intrude on the application or IDE, if present. From a Usability standpoint, Jasmine provides a simple and logical syntax for creating test suites, making it a boon for developers to use for testing.

- Gatling

**Description** Gatling is an open source load testing framework based on Scala, Akka and Netty. The software serves as a tool that can be use for loading testing for the purposes of analysis and measurement of performance of a variety of services, specifically in the case of web applications.

Gatling was chosen because of its utility as a performance analysis tool. This tool will enable the developers to have a quick and easy way to measure the optimality of their implementation of the web interface.

**Performance** Gatling supports, officially, the HTTP, WebSockets, Server-Sent events and JMS protocols. Gatling was built with high performance in mind and its ability to help determine the performance of our web application will make great strides towards ensuring that the application meets the Performance quality requirement in this regard.

**Auditablility/Monitorability** Gatling provides the functionality of creating "Ready-to-Present" HTML reports meaning that once it has performed its work of analysing the performance of the web application, it will then generate an HTML report containing its results. As this document is generated in HTML and also "Ready-to-Present" this enables us to have a high degree of auditability on our performance tests as these reports can be archived and reviewed like other system logs.

**Cost** Gatling does not contribute to the cost of the system as it is an open source load testing framework based on the scala, Akka and Netty languages.

#### 7.2.4 Database System

- HTML5 Local Storage
- Cookies

#### 7.2.5 Operating System

- Device Operating System Independant
- Web Browser
  - Google Chrome
  - Mozilla Firefox
  - Microsoft Internet Explorer 8,9 & 10
  - Microsoft Edge
  - Apple Safari

#### 7.2.6 Dependency Management and Build Tools

- Bower
- Grunt

### 7.3 Android Client

#### 7.3.1 Programming Languages

- Java
- eXtensible Markup Language (XML)

#### 7.3.2 Frameworks

- Spring for Android

#### 7.3.3 Libraries

- Android Butterknife

#### 7.3.4 Database System

- Couchbase Mobile which consists of:
  - Couchbase Lite
  - Couchbase Sync Gateway
  - Couchbase Server

#### 7.3.5 Operating System

- Android 4.2 Devices

#### 7.3.6 Dependency Management and Build Tools

- Gradle

# 8 Functional requirements and application design

## 8.1 Use case prioritization

### 8.1.1 Critical

- Create User
- View/Edit User
- User Authentication[Login/Logout/Permissions]
- Create Paper
- View/Edit Paper
- Web Interface
- Create Authors/Researchers
- View/Edit Authors/Researchers
- Add/Remove Authors/Researchers to Paper
- Create Research Groups
- View/Edit Research Groups
- Add/Remove Users to Research Groups
- Add Venue
- View/Edit Venue

### 8.1.2 Important

- Import Historical Paper [Linked with Create Paper]
- Generate Bibliography
- Research Paper Units
- User Units [Earnings]
- System Logging

### 8.1.3 Nice to have

- Dates for paper linking to Google Calender
- Android Interface
- Paper Progress Tracking
- User Hierarchy
- Home View

## 8.2 Use case/Services contracts

### 8.2.1 User Login - User holds no tokens

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to initiate user login via the back-end service.
- Pre-Conditions
  1. The user is already registered with the back-end service
  2. User account is not locked
  3. User currently does not have required tokens
- Post-Conditions
  1. The user will be logged in
  2. The user will be issued with an access and refresh token
- Service Contract

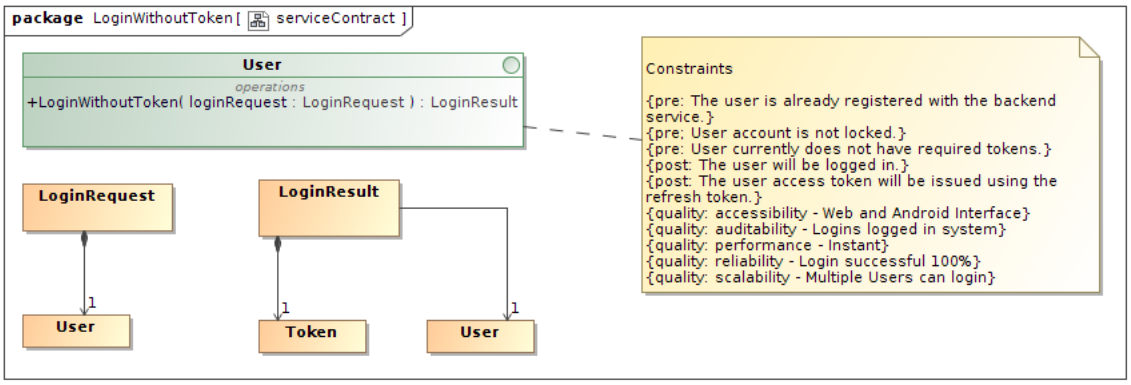


Figure 1: Login without Token Service Contract

### 8.2.2 User Login - User holds access and refresh tokens

- Description
  This use case will be used by the REST clients, specifically the Android app, to achieve safe long term login without storing user credentials on device.
- Pre-Conditions
  1. The user is already registered with the back-end service
  2. User account is not locked
  3. User currently has access and refresh tokens
- Post-Conditions
  1. The user will be logged in
  2. The user access token will be updated using the refresh token
- Service Contract

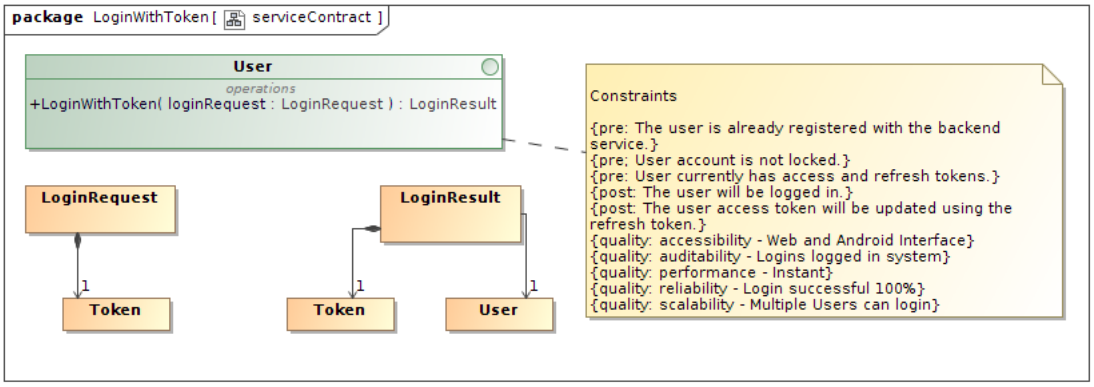


Figure 2: Login with Token Service Contract

### 8.2.3 Log Out User

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to log a user out of the system.
- Pre-Conditions
  1. The user is currently logged into the system
- Post-Conditions
  1. The user will be logged out of the system
- Service Contract

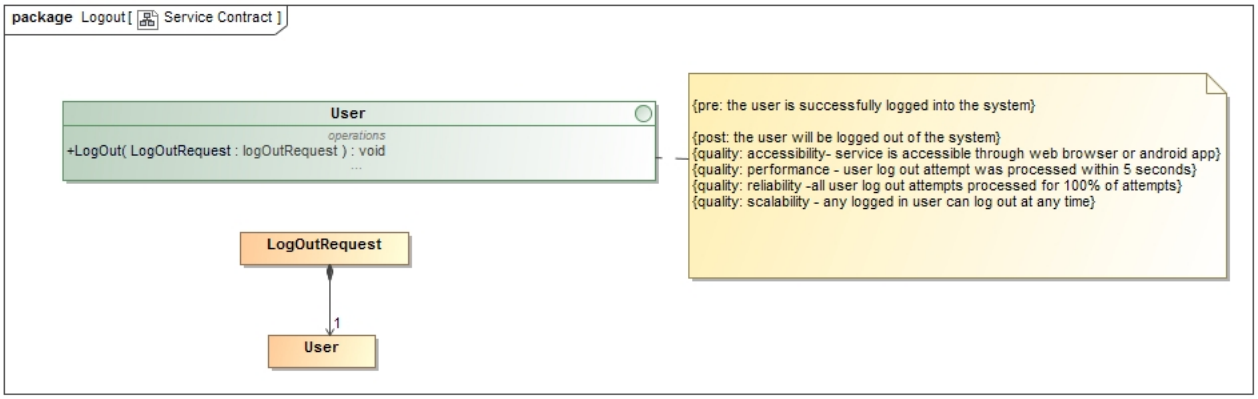


Figure 3: Log the User out of the System

### 8.2.4 Register User

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to Create a user of the system.
- Pre-Conditions
  1. The user does not exist in the system
  2. All required details have been provided
  3. All details are valid
- Post-Conditions

1. The user will be created in the database

- Service Contract

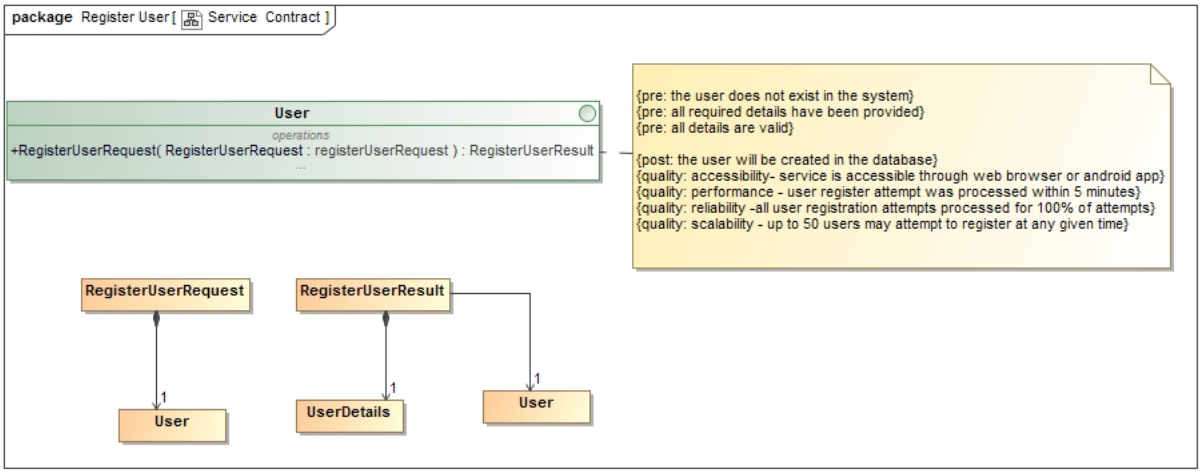


Figure 4: Register a User in the System

### 8.2.5 View/Edit User

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to View and Edit the currently logged in User.
- Pre-Conditions
  1. The user is successfully logged in to the system
  2. Edited details are valid
- Post-Conditions
  1. The user details will be edited
  2. All papers that the user is an author of should be displayed
  3. User page will be displayed
- Service Contract

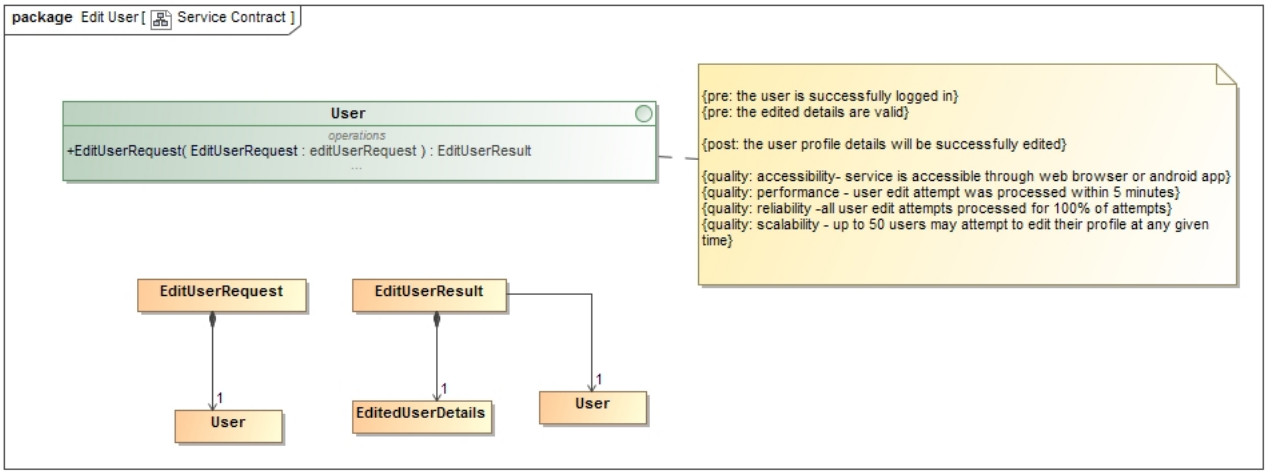


Figure 5: Edit a User Profile in the System

### 8.2.6 Create Paper

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to Create a Research Paper.
- Pre-Conditions
  1. The user is successfully logged in to the system
  2. Paper details are filled in and valid
  3. The user must belong to a research group
- Post-Conditions
  1. Paper successfully created
- Service Contract

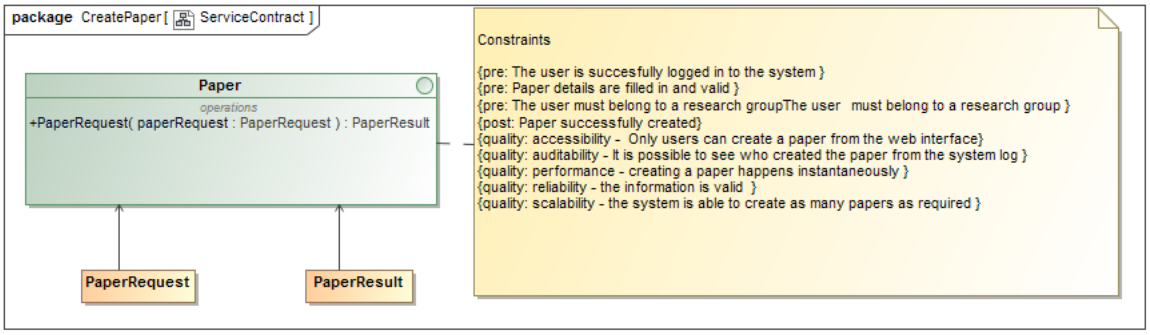


Figure 6: Create Paper Service Contract

### 8.2.7 Edit Paper

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to Edit a Research Paper.
- Pre-Conditions
  1. The user is successfully logged in to the system
  2. The user is an Author of the Paper
  3. The user is an HOD or a Research Group Leader for the Group of the Paper
  4. The new details are valid
- Post-Conditions
  1. The papers details will be edited
  2. The paper details will be displayed
- Service Contract

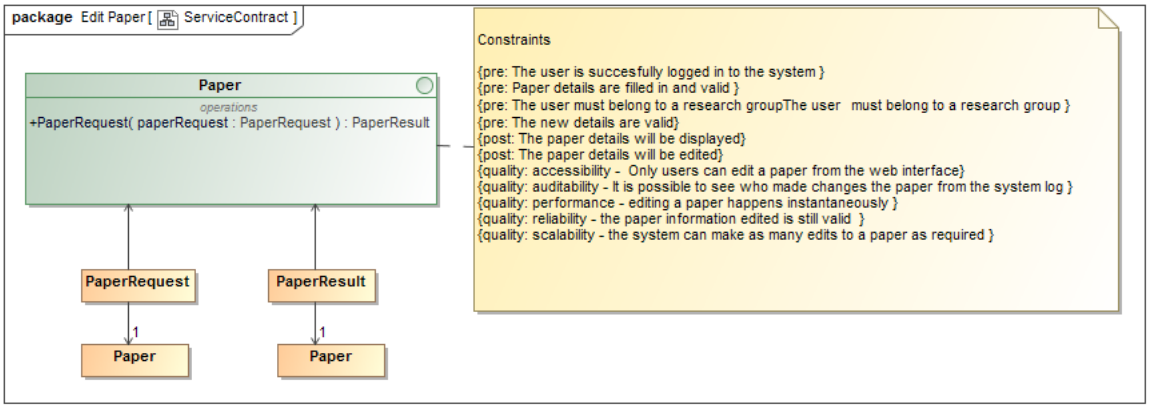


Figure 7: Edit Paper Service Contract

### 8.2.8 View Paper

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to View a Research Paper.
- Pre-Conditions
  1. The user is successfully logged in to the system
  2. The user is an Author of the Paper
  3. The user is an HOD or a Research Group Leader for the Group of the Paper
- Post-Conditions
  1. The paper details will be displayed
- Service Contract

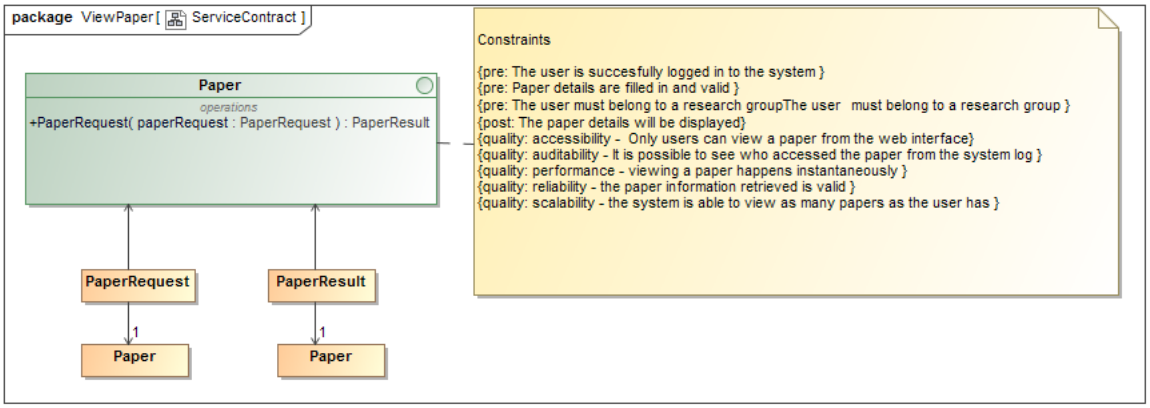


Figure 8: View Paper Service Contract

### 8.2.9 Create Author

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to Create Authors.
- Pre-Conditions
  1. The user is successfully logged in to the system
  2. The details for the Author are valid

3. The author must not previously exist in the system

- Post-Conditions
  1. The Author is successfully created
- Service Contract

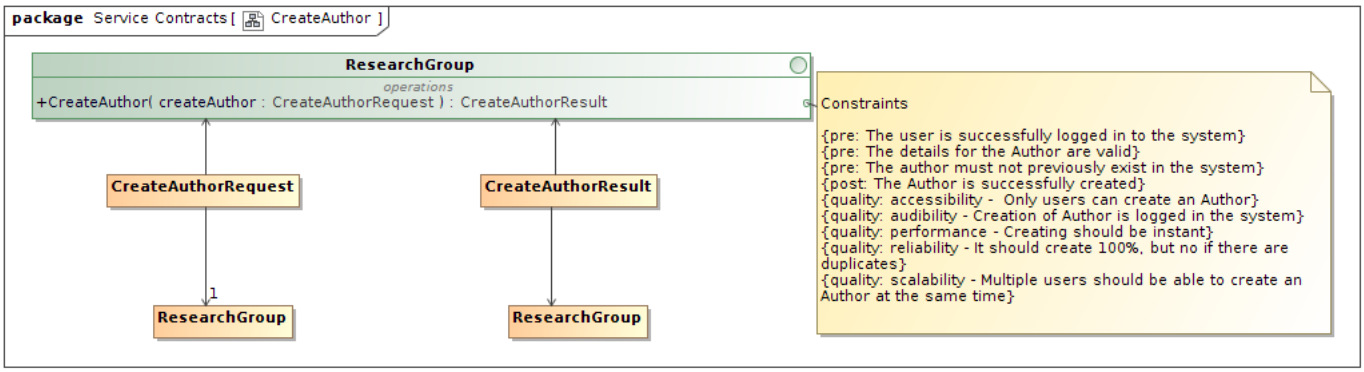


Figure 9: Creat Author

### 8.2.10 View Author

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to View an Author.
- Pre-Conditions
  1. The user is successfully logged in to the system
- Post-Conditions
  1. The Authors details will be displayed
- Service Contract

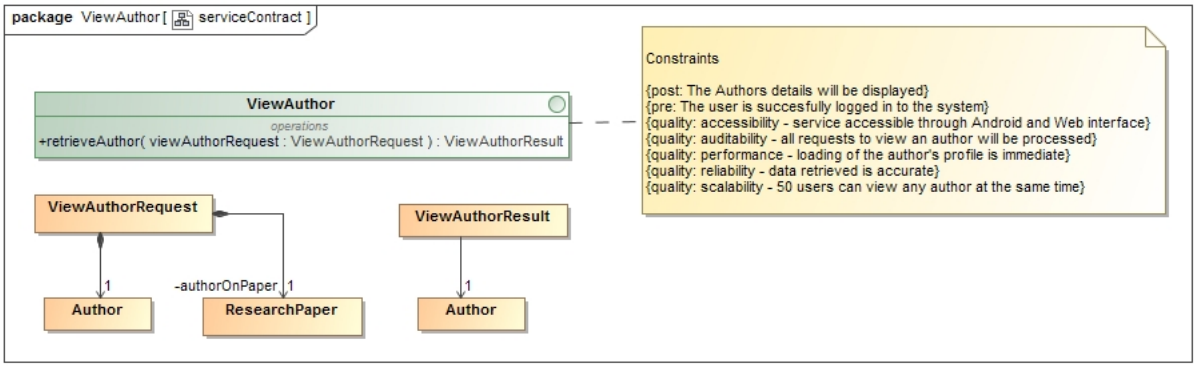


Figure 10: View author service contract

### 8.2.11 Edit Author

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to Edit an Author.
- Pre-Conditions

1. The user is successfully logged in to the system
2. The new details are valid

- Post-Conditions
  1. The Authors details will be edited
  2. The Authors details will be displayed
- Service Contract

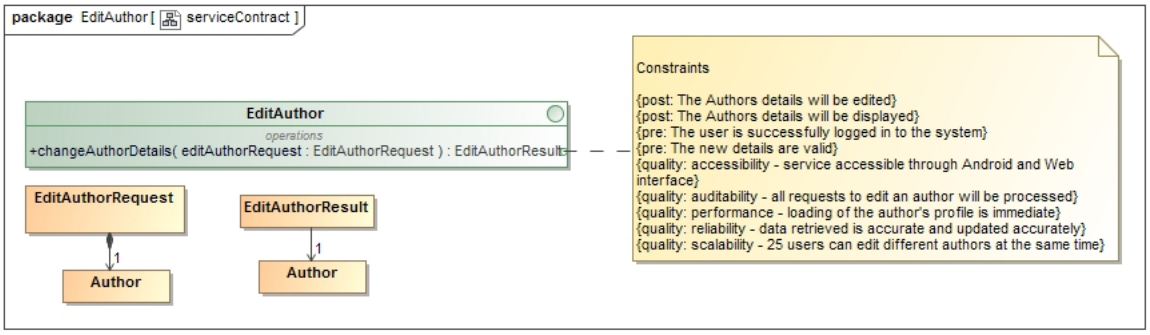


Figure 11: Edit author service contract

### 8.2.12 Add/Remove Author to Research Paper

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to Add and Remove Authors of a Research Paper.
- Pre-Conditions
  1. The user is successfully logged in to the system
  2. The user is an Author of the Paper
  3. The user is an HOD or a Research Group Leader for the Group of the Paper
  4. The new details are valid
- Post-Conditions
  1. The Papers Authors have changed successfully
- Service Contract

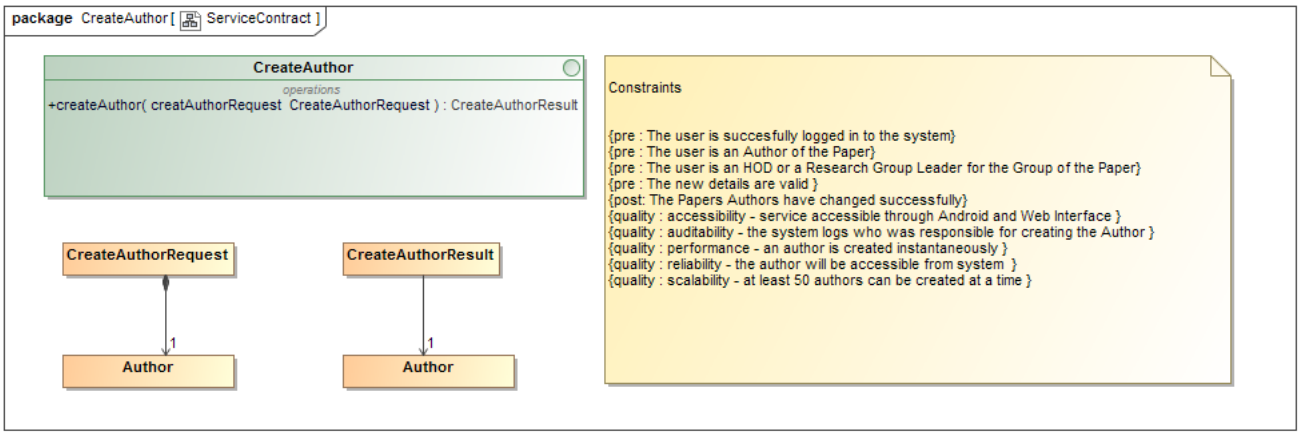


Figure 12: Create Author Service Contract

### 8.2.13 Create Research Group

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to Create a Research Group.
- Pre-Conditions
  1. The user is successfully logged in to the system
  2. The user is an HOD or a Research Group Leader for the Group of the Paper
- Post-Conditions
  1. The Research Group is successfully added
- Service Contract

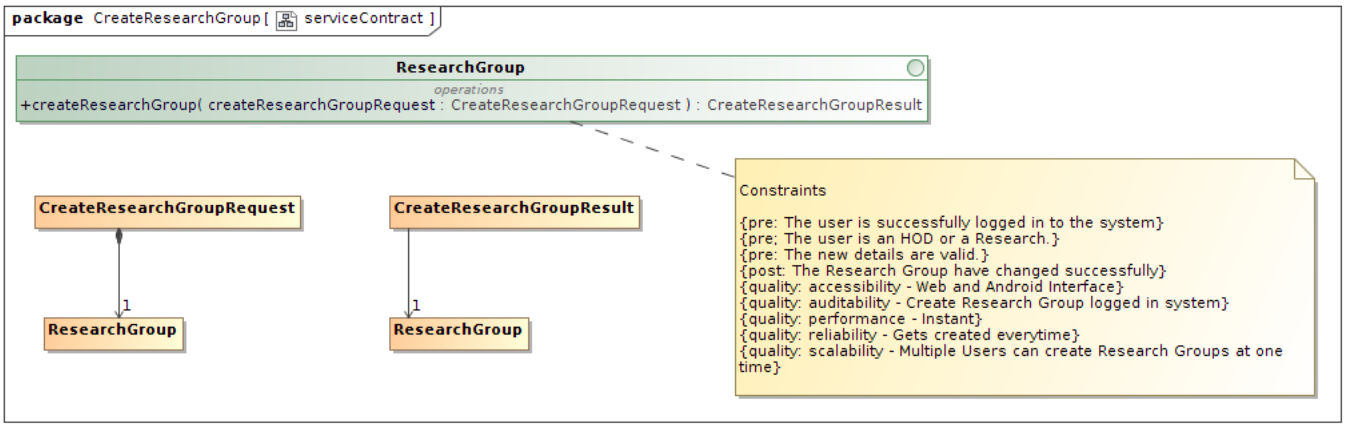


Figure 13: Create Research Group Service Contract

### 8.2.14 View Research Groups

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to View a Research Group.
- Pre-Conditions
  1. The user is successfully logged in to the system
  2. The user is an HOD or a Research Group Leader for the Group of the Paper
- Post-Conditions
  1. The Research Group has been displayed
- Service Contract

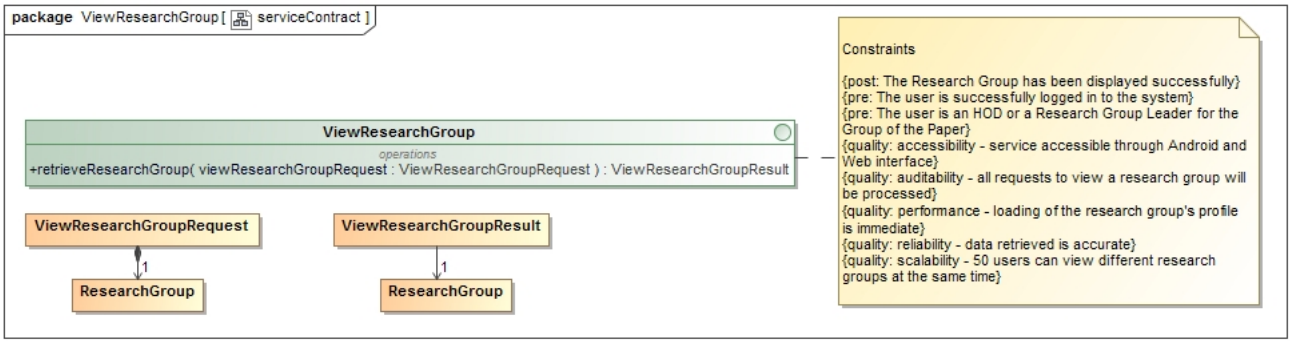


Figure 14: View research group service contract

### 8.2.15 Edit Research Groups

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to Edit a Research Group.
- Pre-Conditions
  1. The user is successfully logged in to the system
  2. The user is an HOD or a Research Group Leader for the Group of the Paper
  3. The new details are valid
- Post-Conditions
  1. The Research Group has been changed successfully
- Service Contract

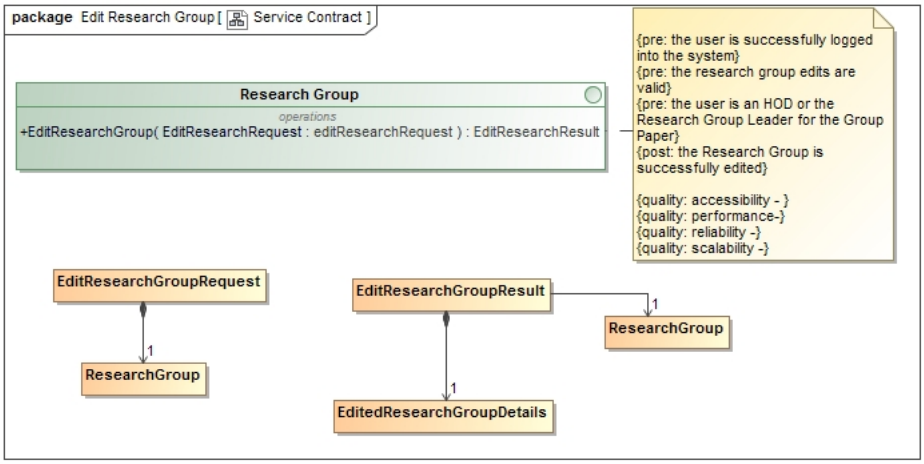


Figure 15: Edit a Research Group's data in the System

### 8.2.16 Add/Remove Users to Research Groups

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to Add/Remove Users to a Research Group.
- Pre-Conditions
  1. The user is successfully logged in to the system

2. The user is an HOD or a Research Group Leader for the Group of the Paper
3. The user is valid

- Post-Conditions
  1. The User is successfully added to a Research Group
- Service Contract

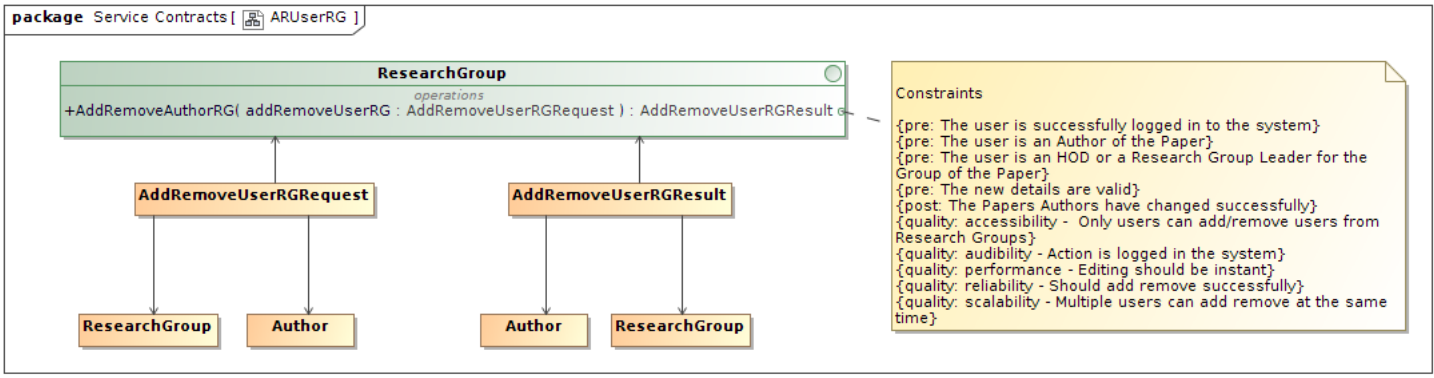


Figure 16: Add Users to Research Group

### 8.2.17 Import Historical Paper

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to Import a Historical Paper.
- Pre-Conditions
  1. The user is successfully logged in to the system
  2. The paper details are correct
- Post-Conditions
  1. The Historical Paper is successfully added
- Service Contract

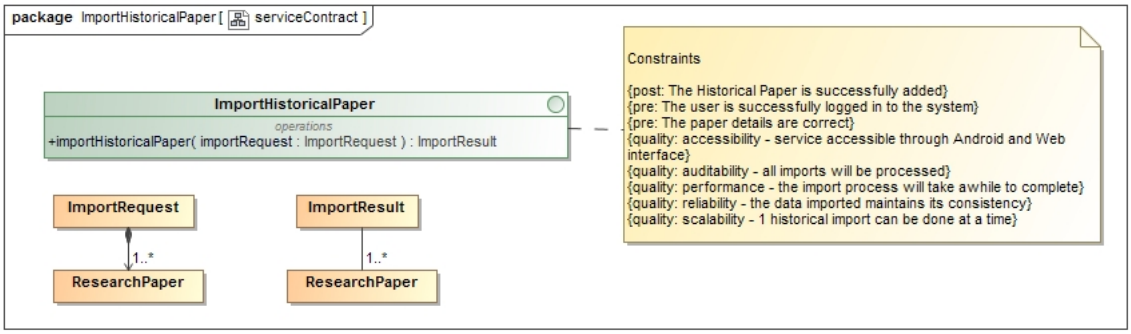


Figure 17: Import historical paper service contract

### 8.2.18 Generate Bibliography

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to Generate a Bibliography for a User.

- Pre-Conditions
  1. The user is successfully logged in to the system
- Post-Conditions
  1. Bibliography successfully generated
- Service Contract

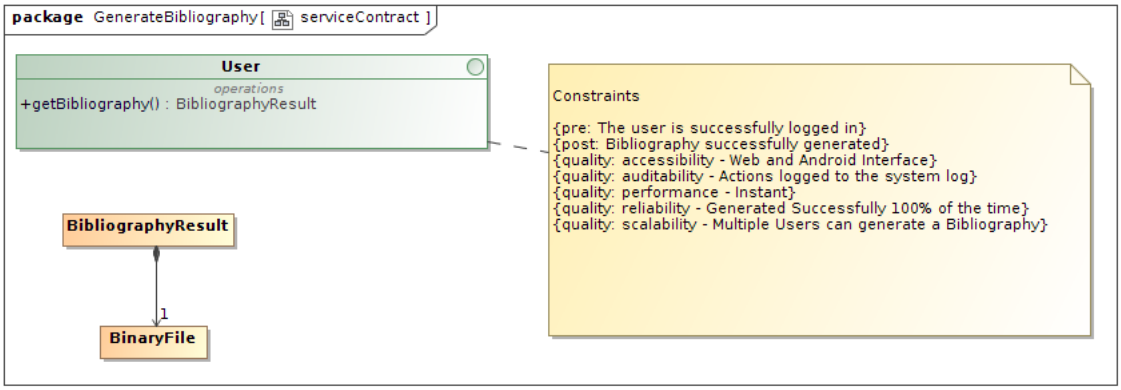


Figure 18: Generate Bibliography Service Contract

### 8.2.19 Create Venue

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to Create a Venue.
- Pre-Conditions
  1. The user is successfully logged in to the system
  2. Venue details are valid
- Post-Conditions
  1. The Venue is successfully created
- Service Contract

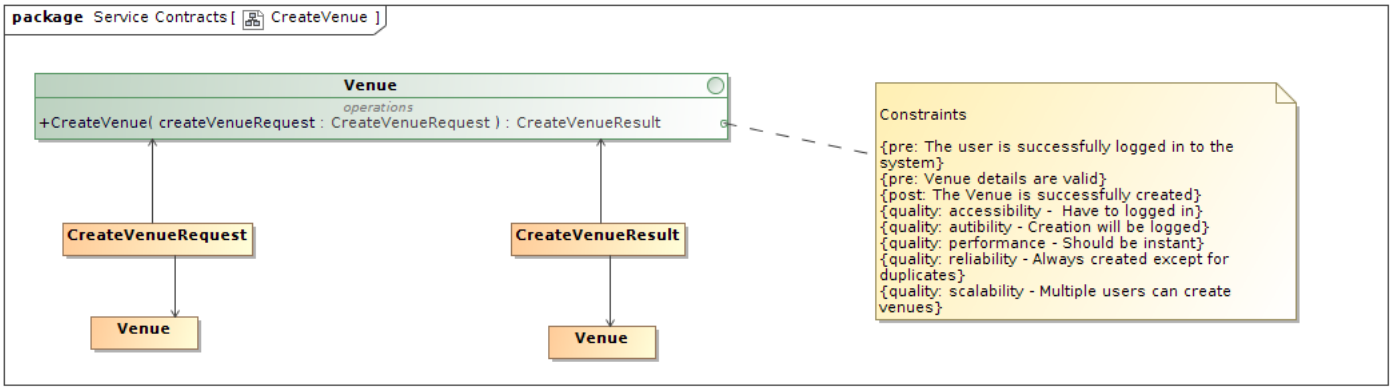


Figure 19: Create Venue

### 8.2.20 View/Edit Venue

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to View/Edit a Venue.
- Pre-Conditions
  1. The user is successfully logged in to the system
  2. The new details are valid
- Post-Conditions
  1. The Venue is successfully edited
- Service Contract

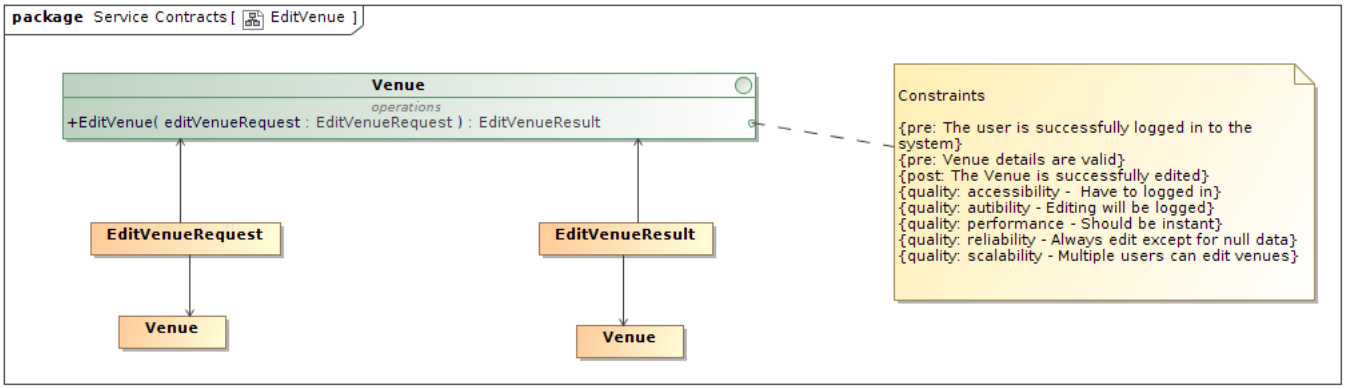


Figure 20: View/Edit Venue

### 8.2.21 View Log

- Description
  This use case will be used by the REST clients, specifically the web client and Android app, to view a system log.
- Pre-Conditions
  1. The user is successfully logged in to the system
  2. The user must be a Super User
- Post-Conditions
  1. The system log will be displayed

## 8.3 Required functionality

### 8.3.1 Login with Token

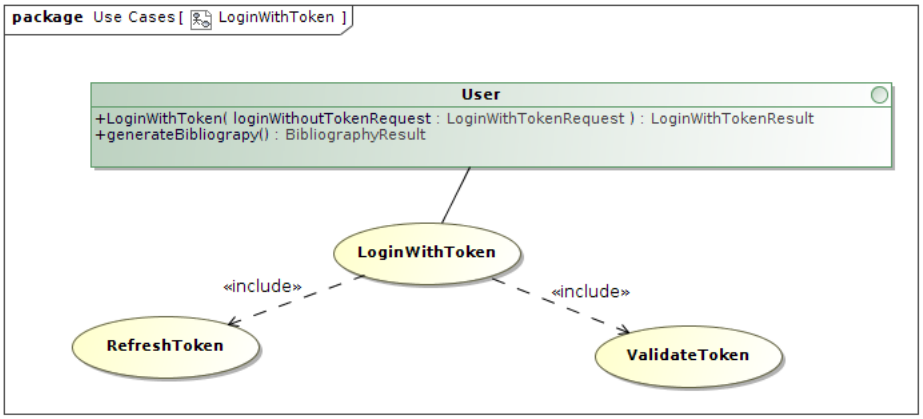


Figure 21: Use Case for Login with Token

### 8.3.2 Login without Token

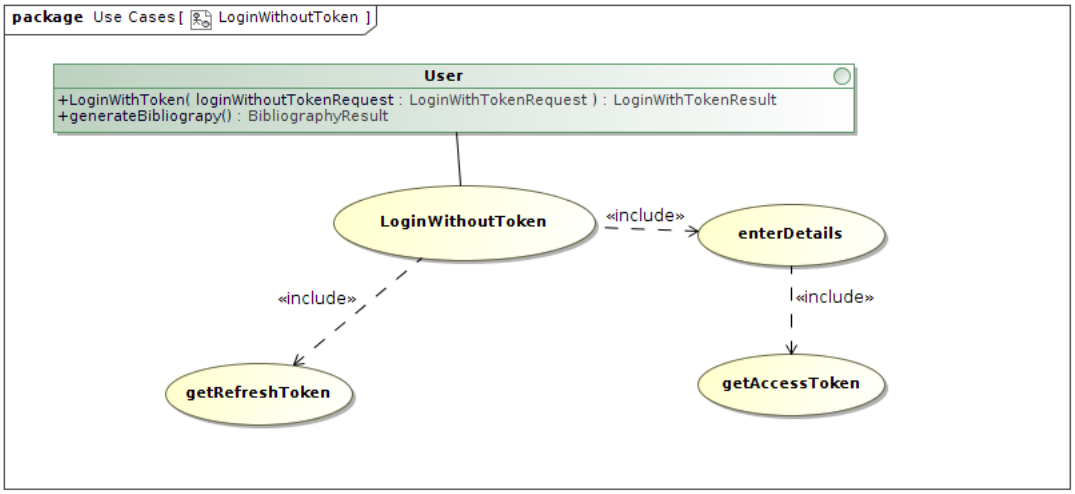


Figure 22: Use Case for Login without Token

### 8.3.3 Edit Research Group

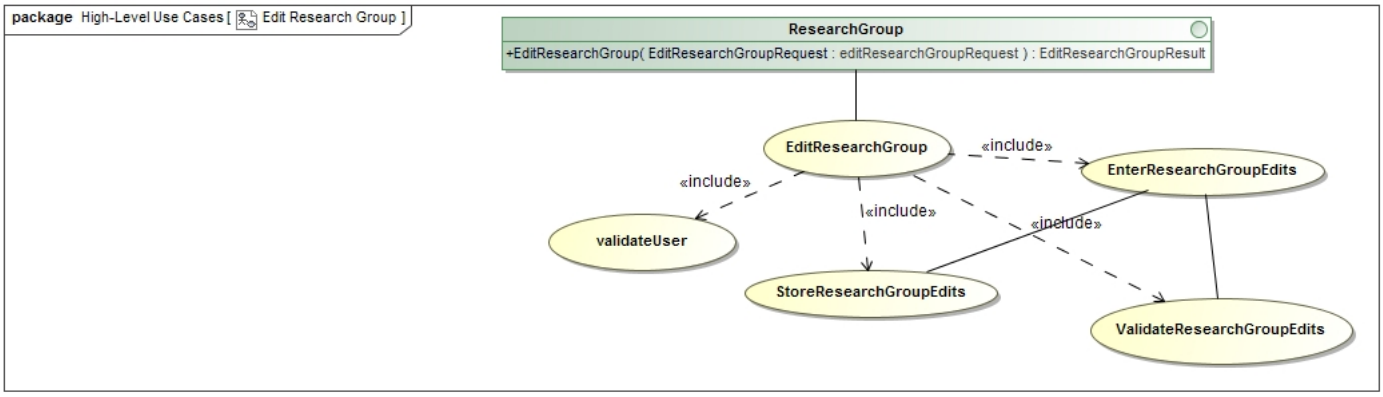


Figure 23: Use Case for Edit Research Group

### 8.3.4 Edit User

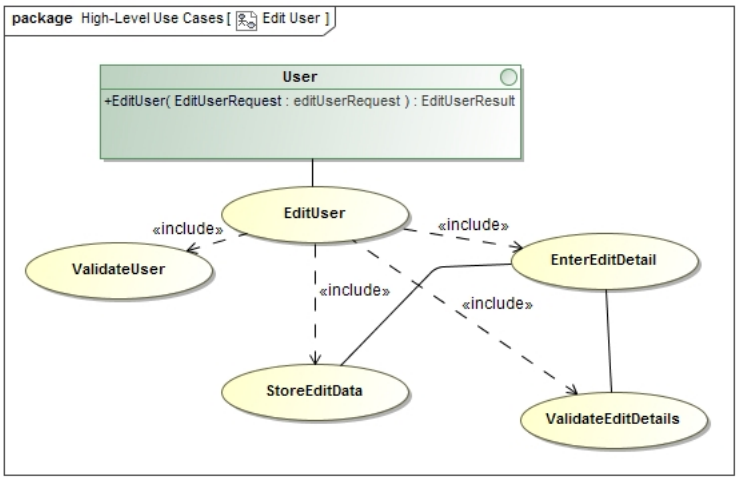


Figure 24: Use Case for Edit User

### 8.3.5 View User

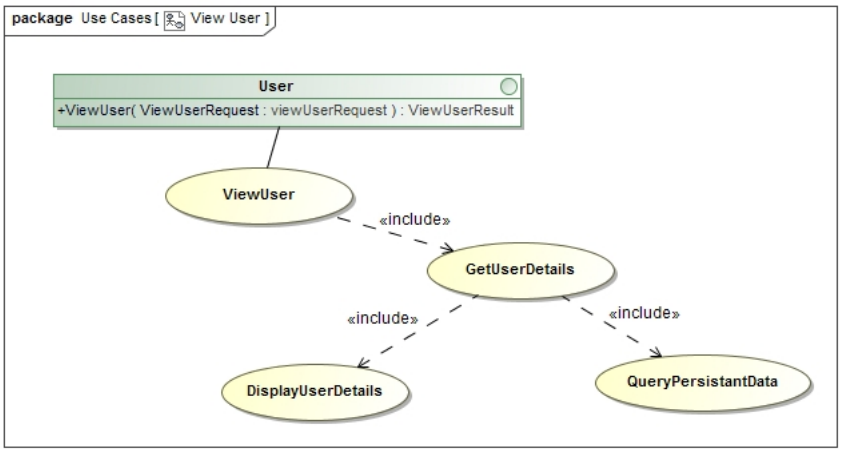


Figure 25: Use Case for View User

### 8.3.6 User Log Out

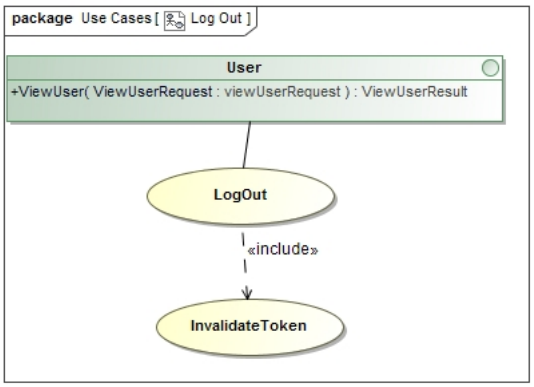


Figure 26: Use Case for User Log Out

### 8.3.7 Create Paper

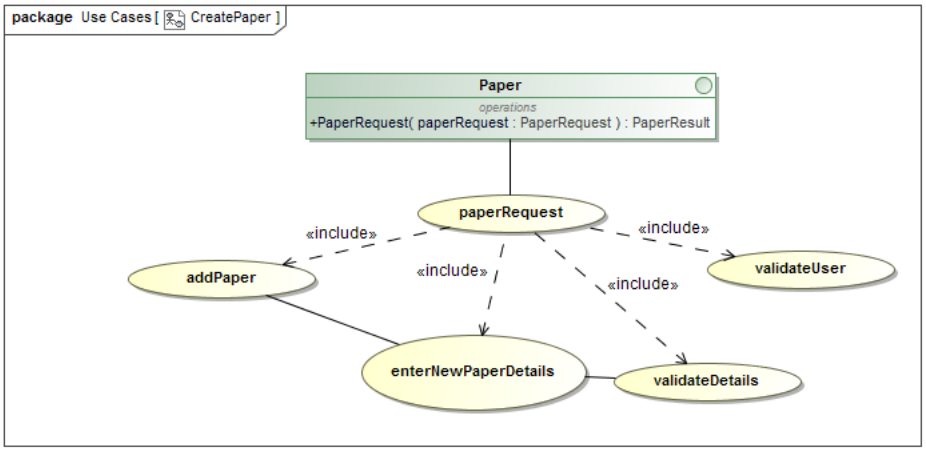


Figure 27: Use Case for Create Paper

### 8.3.8 View Paper

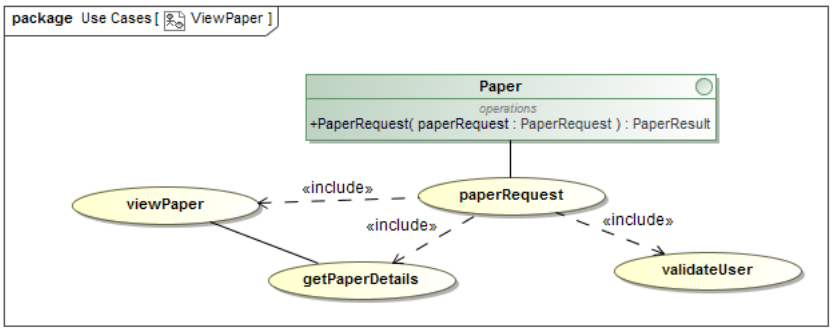


Figure 28: Use Case for View Paper

### 8.3.9 Edit Paper

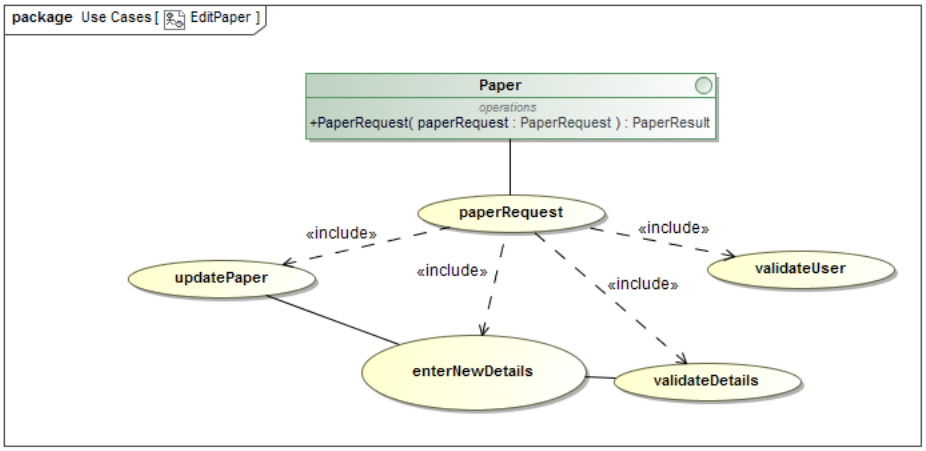


Figure 29: Use Case for Edit Paper

### 8.3.10 Register User

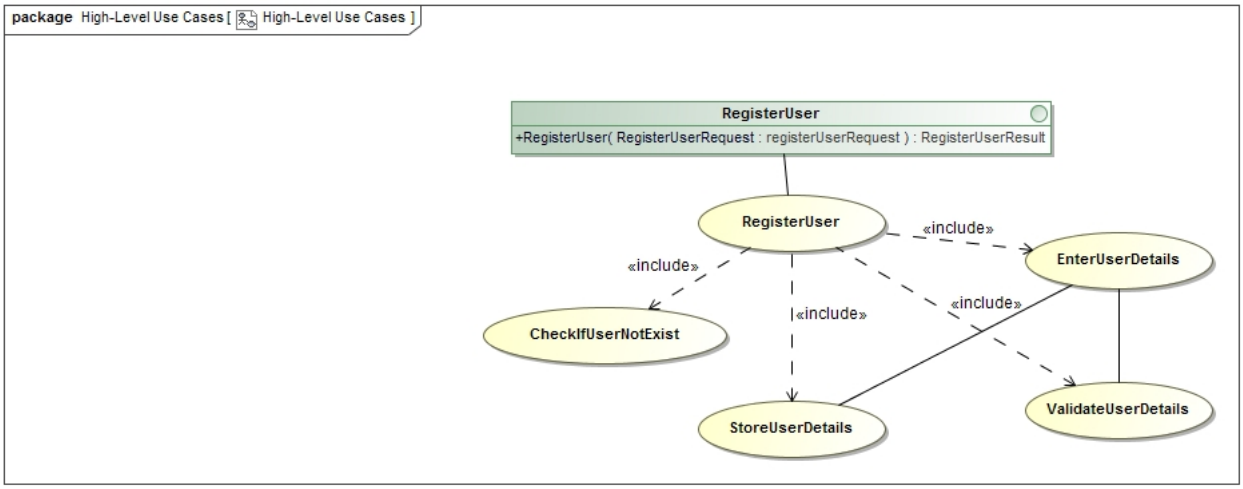


Figure 30: Use Case for Register User

### 8.3.11 Create Author

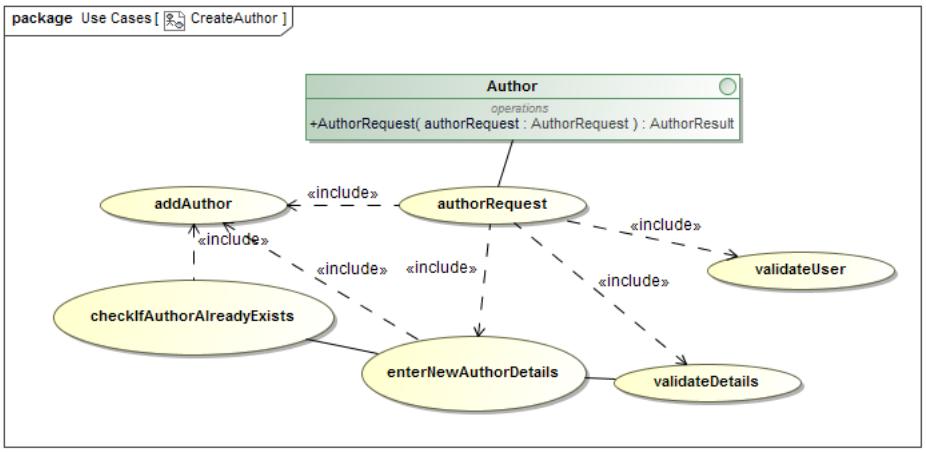


Figure 31: Use Case Diagram for Create Author

### 8.3.12 View Author

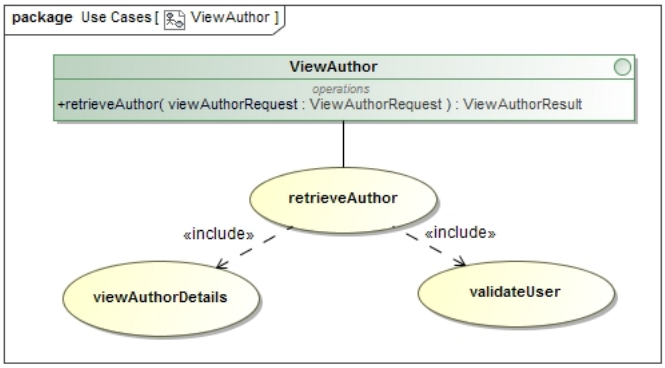


Figure 32: Use Case Diagram for View Author

### 8.3.13 Edit Author

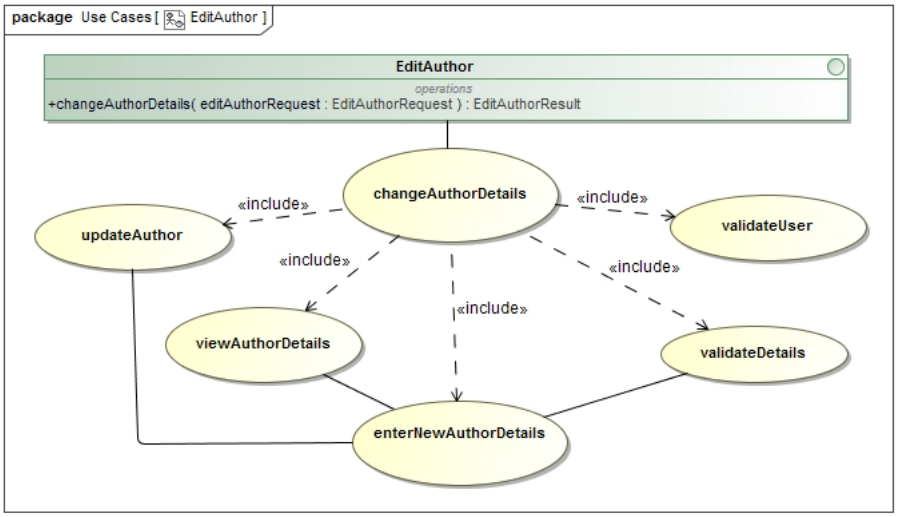


Figure 33: Use Case Diagram for Edit User

### 8.3.14 Create Research Group

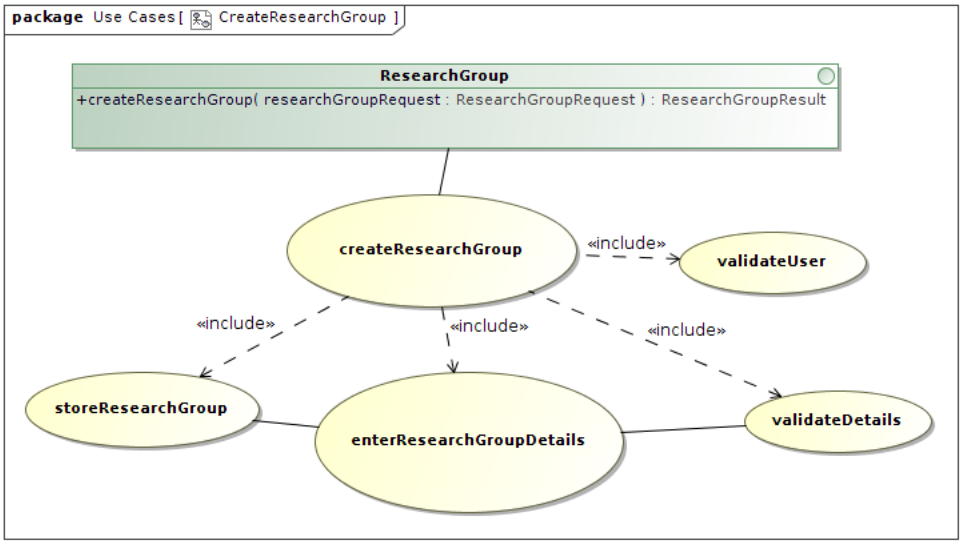


Figure 34: Use Case for Creating a Research Group

### 8.3.15 View Research Group

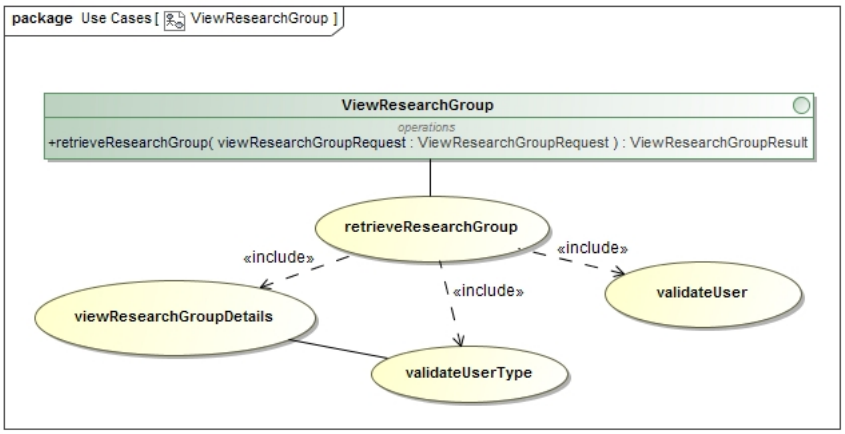


Figure 35: Use Case Diagram for View Research Group

### 8.3.16 Import Historical Paper

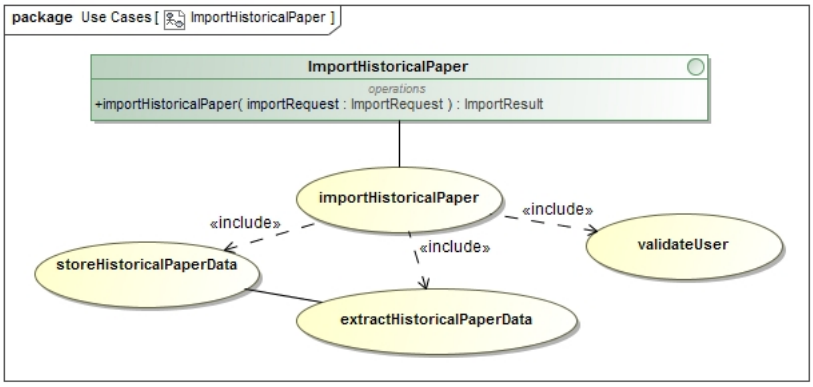


Figure 36: Use Case Diagram for Import Historical Paper

### 8.3.17 Generate Bibliography

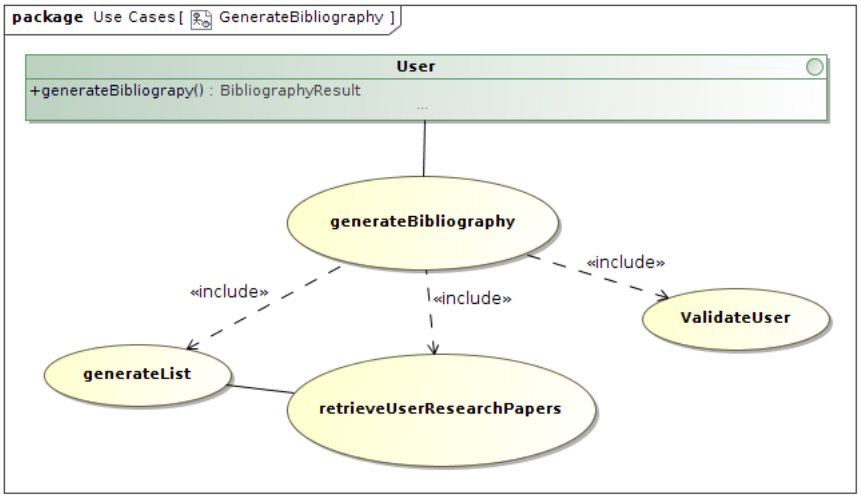


Figure 37: Use Case for Generating a Bibliography

## 8.4 Process specifications

### 8.4.1 Login with Token

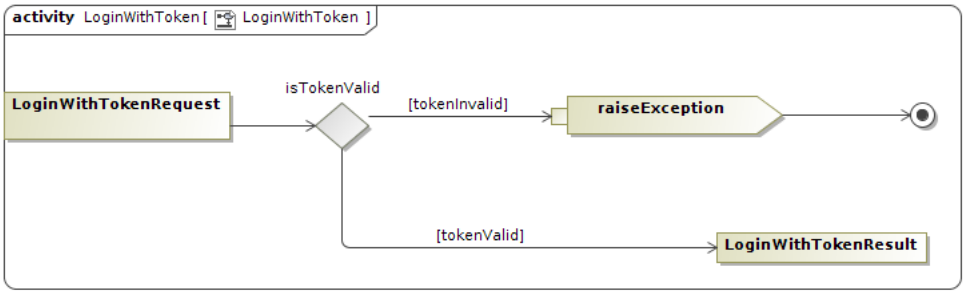


Figure 38: Activity Diagram for Login with Token

### 8.4.2 Login without Token

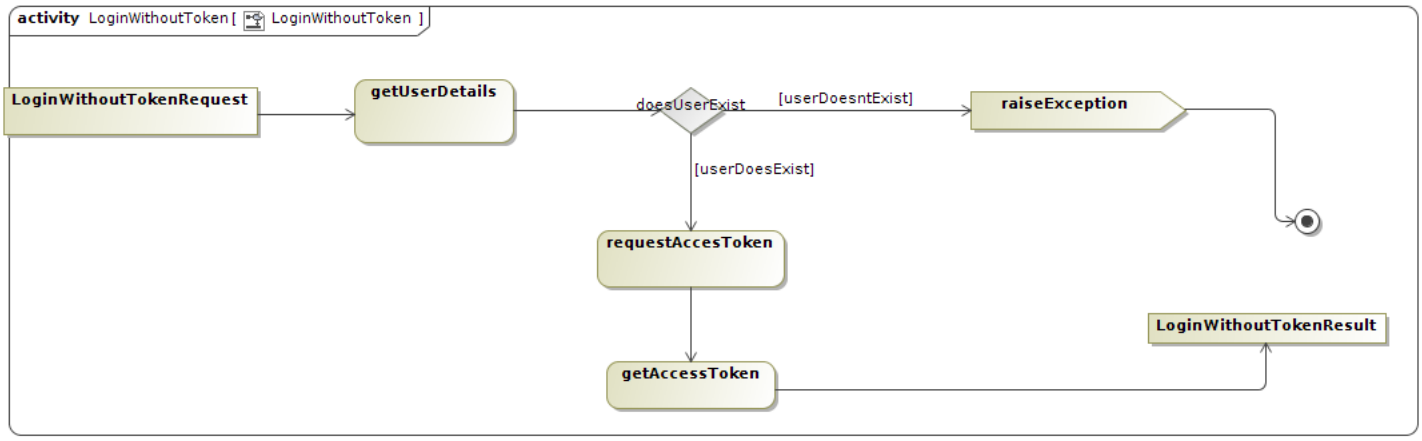


Figure 39: Activity Diagram for Login without Token

### 8.4.3 Edit Research

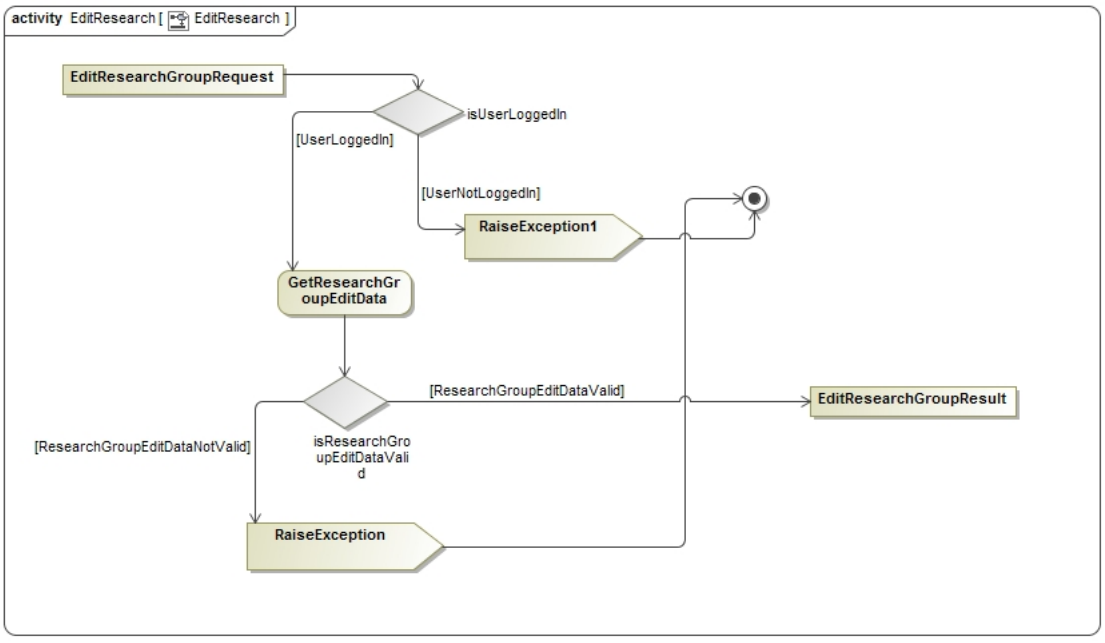


Figure 40: Process Specification for Edit Research

### 8.4.4 Edit User

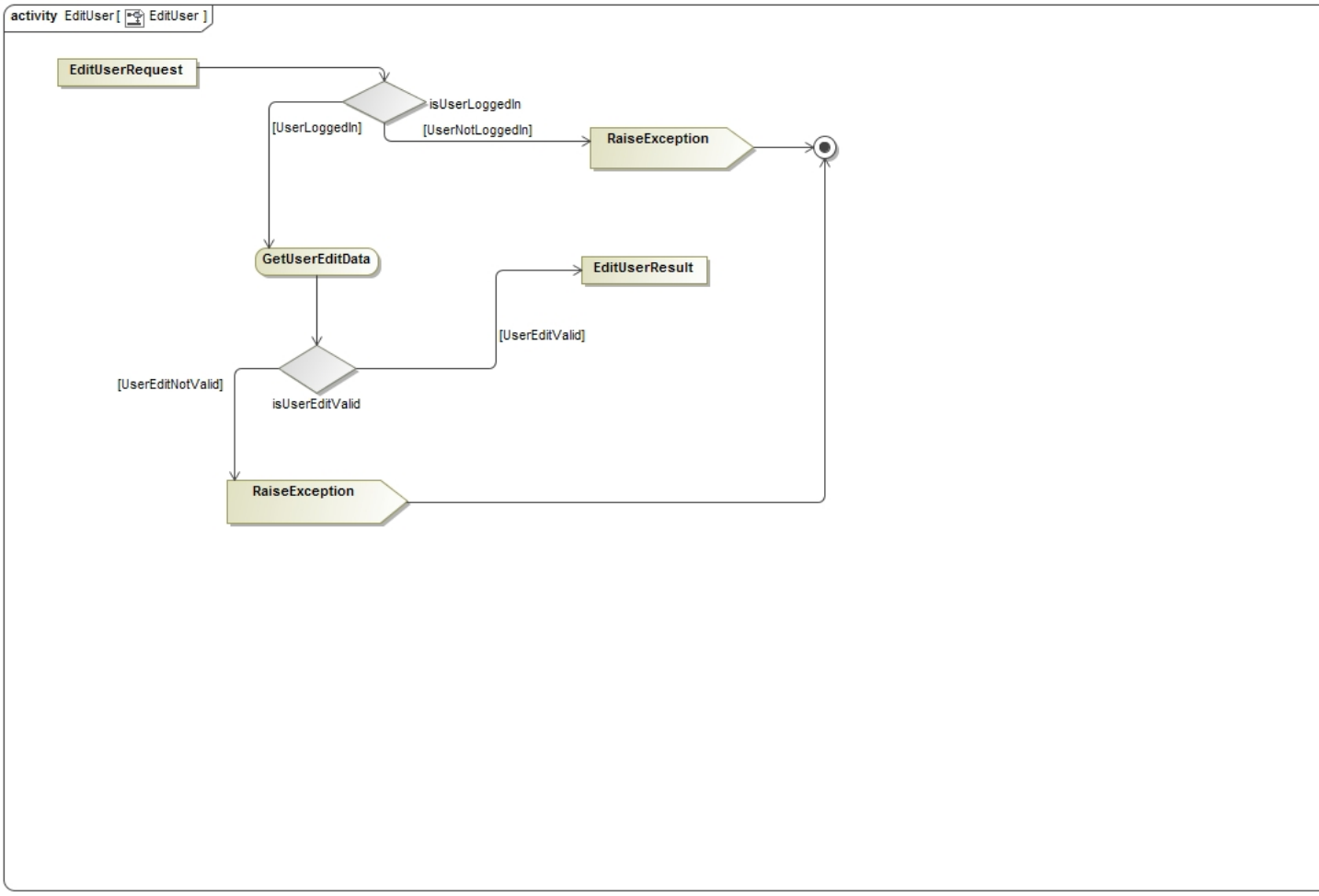


Figure 41: Process Specification for Edit User

### 8.4.5 Log Out

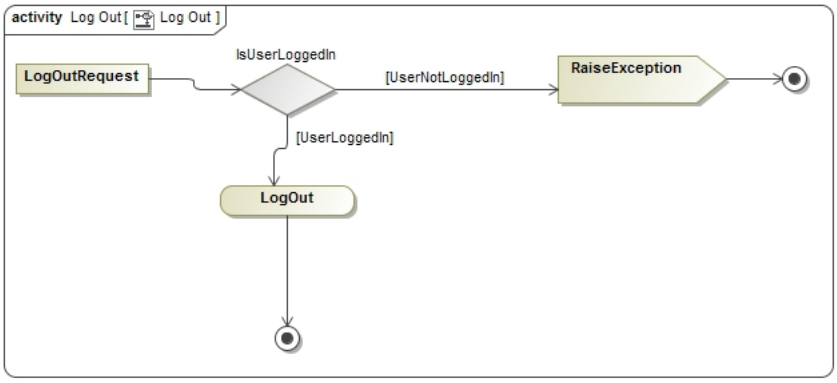


Figure 42: Process Specification for Log Out

### 8.4.6 Register User

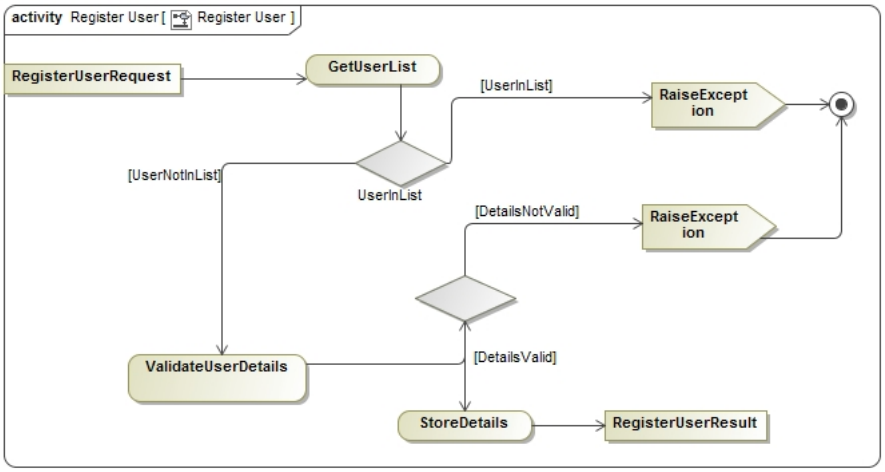


Figure 43: Process Specification for Register User

### 8.4.7 View User

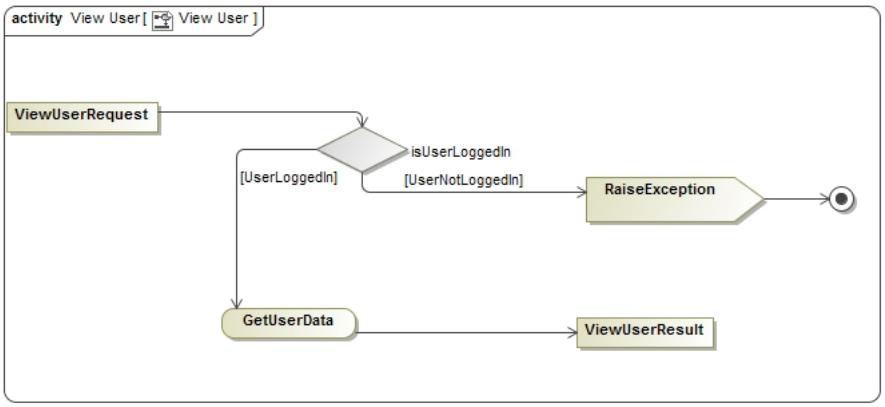


Figure 44: Process Specification for View User

### 8.4.8 View Author

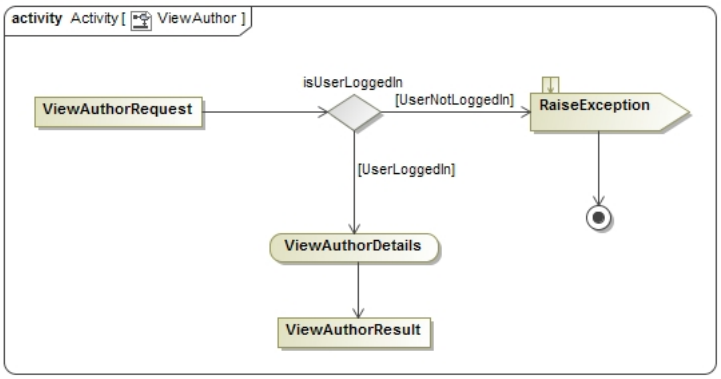


Figure 45: Activity Diagram for View Author

### 8.4.9 Create Author

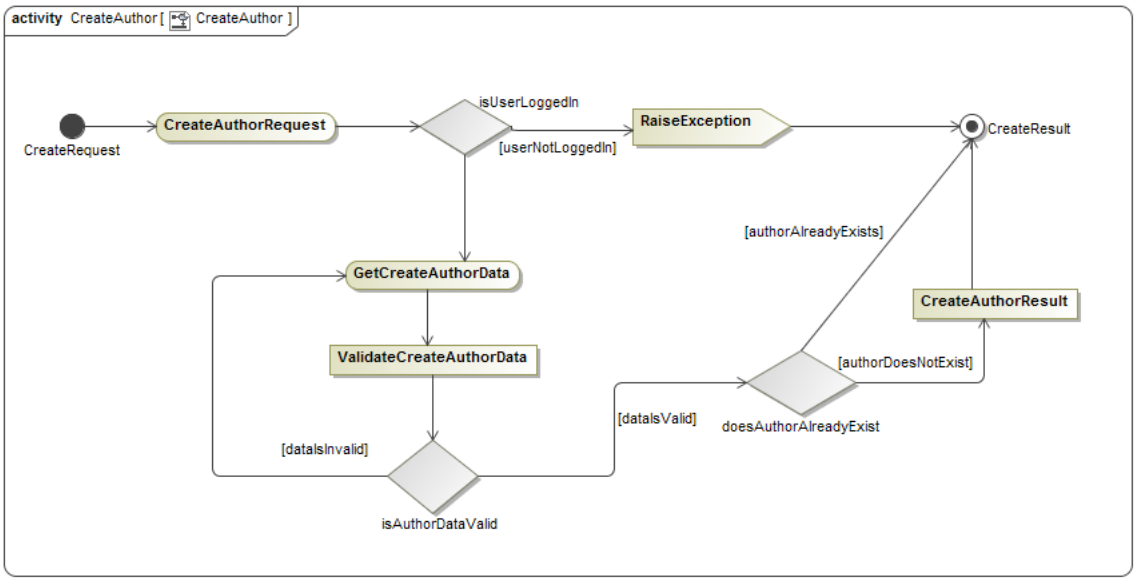


Figure 46: Activity Diagram for Create Author

### 8.4.10 Edit Author

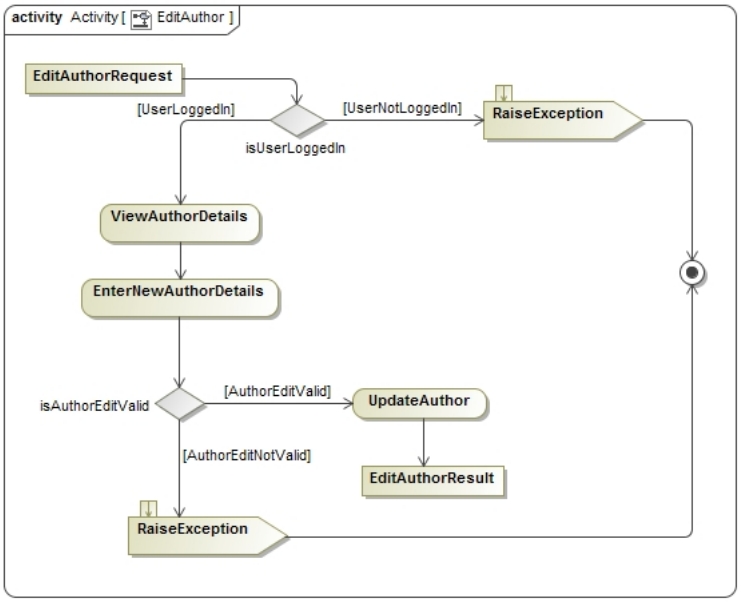


Figure 47: Activity Diagram for Edit User

### 8.4.11 Create Research Group

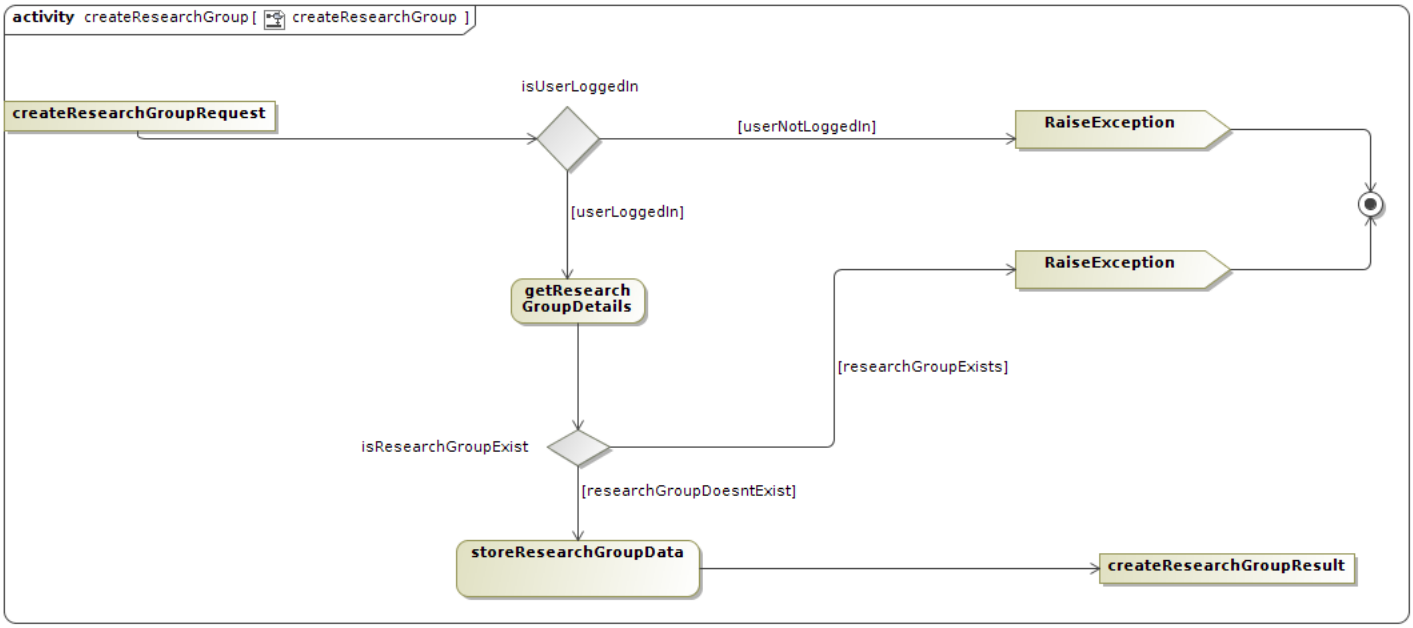


Figure 48: Activity Diagram for Creating a Research Group

### 8.4.12 View Research Group

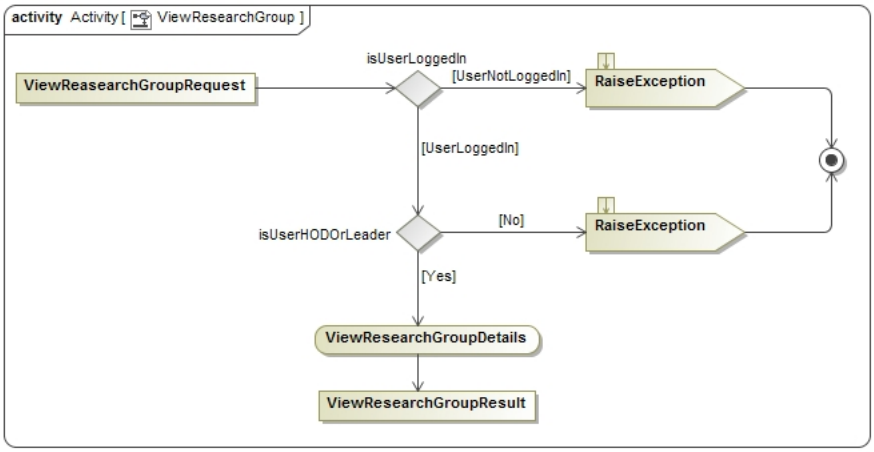


Figure 49: Activity Diagram for View Research Group

### 8.4.13 Import Historical Paper

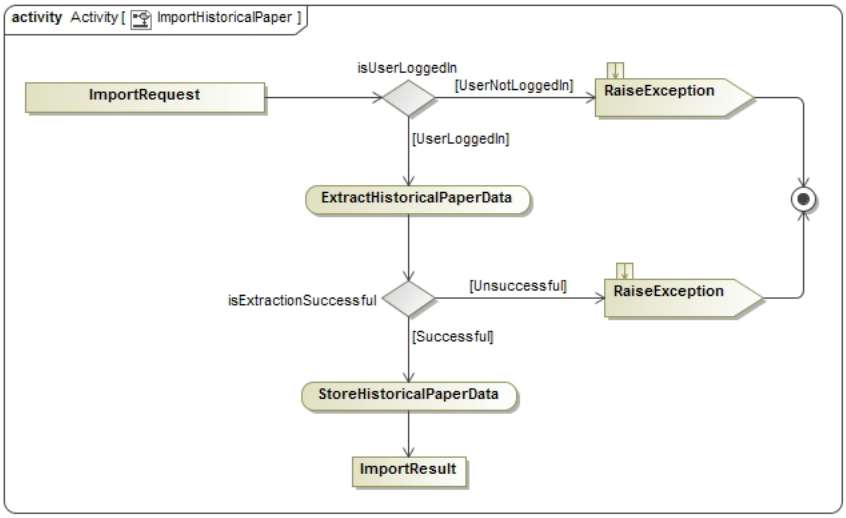


Figure 50: Activity Diagram for Import Historical Paper

### 8.4.14 Create Paper

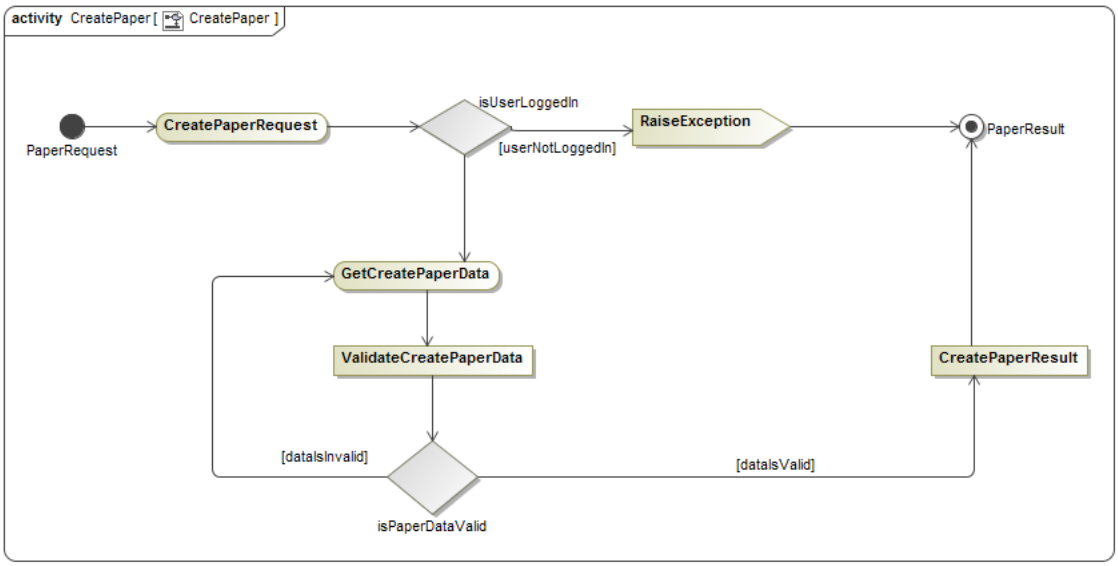


Figure 51: Activity Diagram for Create Paper

### 8.4.15 View Paper

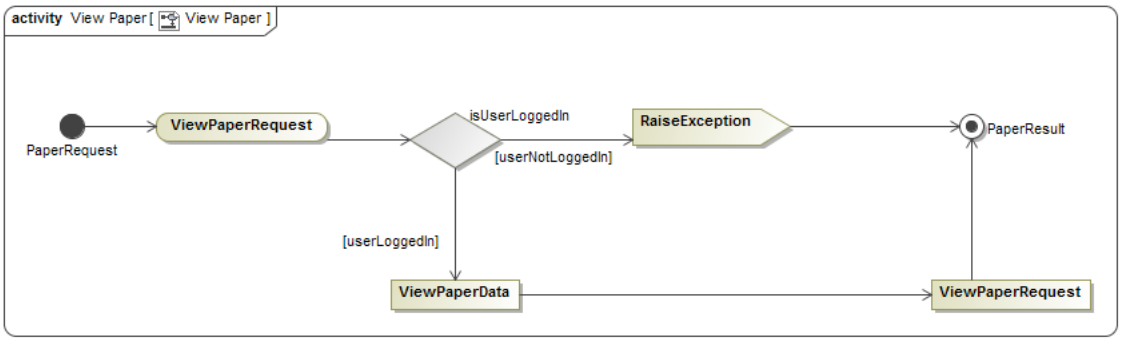


Figure 52: Activity Diagram for View Paper

### 8.4.16 Edit Paper

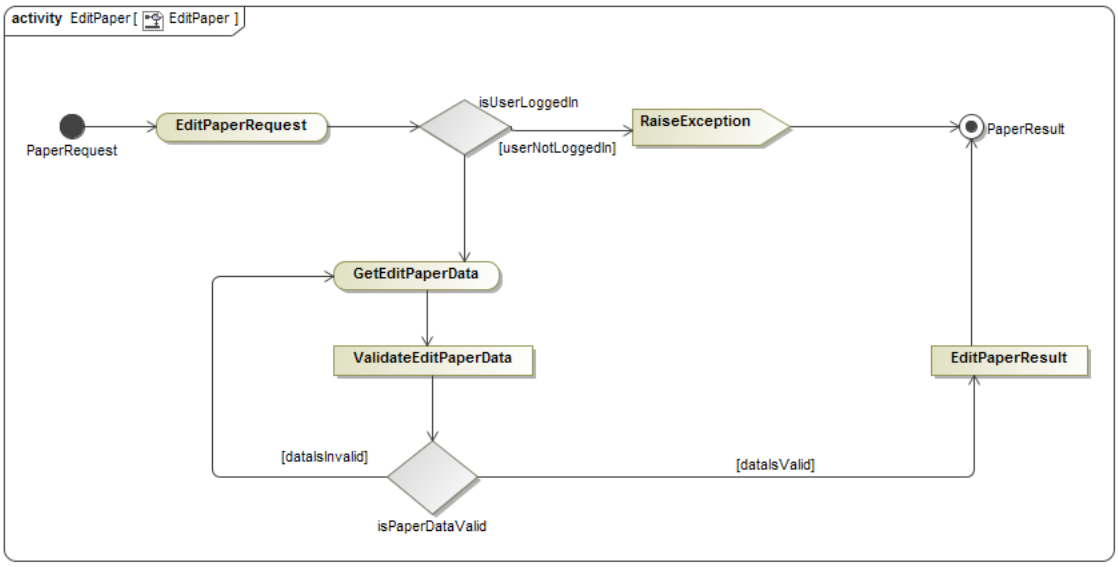


Figure 53: Activity Diagram for Edit Paper

### 8.4.17 Generate Bibliography

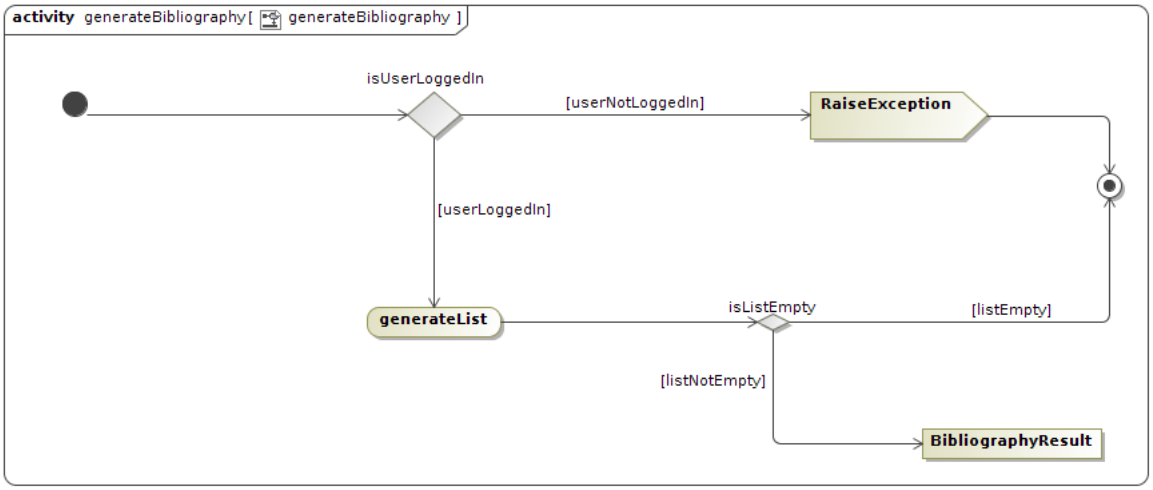


Figure 54: Activity Diagram for Generating a Bibliography

## 8.5 Domain Model

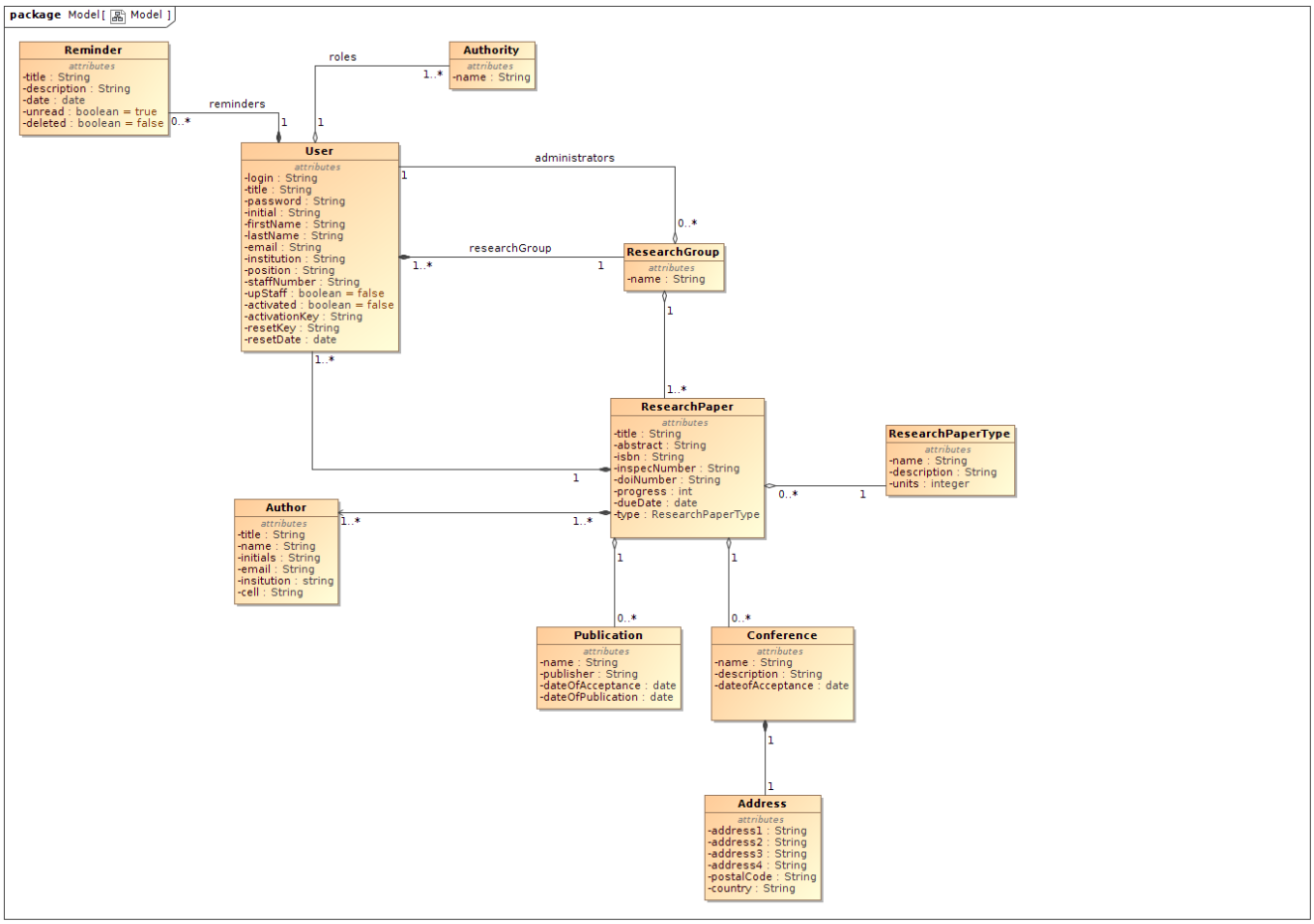


Figure 55: Domain Model

# 9 Open Issues

## 9.1 GitHub Repository

Team Echo Repository: `https://github.com/andrewbroekman/echo`
This repository contains:

1. All work done by team members.
2. CONTRIBUTORS.md file outlining which members where involved in which phases of the project.

# Appendix A Data Dictionary

| Term | Definition |
|---|---|
| Android | A Linux based operating system developed by Google Inc. and the Open Handset Alliance for smart phones, tablets, and other mobile computing devices. Android applications are developed in Java. |
| HTTP | HyperText Transfer Protocol, standardised by RFCs 7230-7237 |
| ISO | International Organization for Standardization |
| MVC | Model-View-Controller |
| OAuth | An open standard for authorization, using an access token approach, allowing resource owners to authorize third-party clients to access protected user resources on a resource server, without the client providing their credentials to the third-party in question. |
| RDBMS | Relational Database Management System |
| SPA | Single Page Application |
| SQL | Structured Query Language is a programming language, standardised by ISO for managing data in RDBMS. Actions allow for the creation, reading/retrieving, updating and deletion of data in the management system. |

# List of Figures


1 Login without Token Service Contract . . . 14
2 Login with Token Service Contract . . . 15
3 Log the User out of the System . . . 15
4 Register a User in the System . . . 16
5 Edit a User Profile in the System . . . 16
6 Create Paper Service Contract . . . 17
7 Edit Paper Service Contract . . . 18
8 View Paper Service Contract . . . 18
9 Creat Author . . . 19
10 View author service contract . . . 19
11 Edit author service contract . . . 20
12 Create Author Service Contract . . . 20
13 Create Research Group Service Contract . . . 21
14 View research group service contract . . . 22
15 Edit a Research Group's data in the System . . . 22
16 Add Users to Research Group . . . 23
17 Import historical paper service contract . . . 23
18 Generate Bibliography Service Contract . . . 24
19 Create Venue . . . 24
20 View/Edit Venue . . . 25
21 Use Case for Login with Token . . . 26
22 Use Case for Login without Token . . . 26
23 Use Case for Edit Research Group . . . 27
24 Use Case for Edit User . . . 27
25 Use Case for View User . . . 27
26 Use Case for User Log Out . . . 28
27 Use Case for Create Paper . . . 28
28 Use Case for View Paper . . . 28
29 Use Case for Edit Paper . . . 29
30 Use Case for Register User . . . 29
31 Use Case Diagram for Create Author . . . 30
32 Use Case Diagram for View Author . . . 30
33 Use Case Diagram for Edit User . . . 30
34 Use Case for Creating a Research Group . . . 31
35 Use Case Diagram for View Research Group . . . 31
36 Use Case Diagram for Import Historical Paper . . . 32
37 Use Case for Generating a Bibliography . . . 32
38 Activity Diagram for Login with Token . . . 32
39 Activity Diagram for Login without Token . . . 33
40 Process Specification for Edit Research . . . 33
41 Process Specification for Edit User . . . 34
42 Process Specification for Log Out . . . 34
43 Process Specification for Register User . . . 35
44 Process Specification for View User . . . 35

45 Activity Diagram for View Author . . . . . 35
46 Activity Diagram for Create Author . . . . . 36
47 Activity Diagram for Edit User . . . . . 36
48 Activity Diagram for Creating a Research Group . . . . . 37
49 Activity Diagram for View Research Group . . . . . 37
50 Activity Diagram for Import Historical Paper . . . . . 38
51 Activity Diagram for Create Paper . . . . . 38
52 Activity Diagram for View Paper . . . . . 39
53 Activity Diagram for Edit Paper . . . . . 39
54 Activity Diagram for Generating a Bibliography . . . . . 40
55 Domain Model . . . . . 40