University of Pretoria

COS 301 - Software Engineering

Team Fox

Software Requirements Specification and Technology Neutral Process Design

Author(s):	*Student number(s):*
Gian Paolo Buffo	14446619
Jita Hlengekile	14077893
Jacques van Wyk	12230091
Jedd Schneier	13133064
Jason Gordon	14405025
Josef Alberts	14395283
Sandile Khumalo	12031748
Kudzai Muranga	13278012

March 11, 2016

Contents

1 Introduction

This is the Software Requirements specification document for the system to keep record of research papers in the Computer Science Department of the University of Pretoria. This document includes sections such as:

1. Vision for the system
    - This includes the projected vision and outcomes of the system and what the Computer Science Department aims to achieve with this project.
2. Background to why the system is being developed
    - This includes a discussion of the problem which the Computer Science Department is faced with in regard to keeping track of their research papers and how this system will find a solution to the problem being faced.
3. Architecture Requirements
    - This includes how the system will be accessed by the users of the system and through which platforms this will be done. Additionally the quality requirements of the system will be discussed as well as what external systems will be used to put this system together. Lastly it will include any constraints that the client has specified with regard to what must be used in the system.
4. Functional requirements and application design
    - This will include the functionality of the system that is required by the users of the system.
5. Open Issues
    - This section will discuss any requirements of the system that may be unclear or has not been specified. It will also include any inconsistencies that have been discovered in the requirements.

2 Vision

With the implementation of this system, the client is trying to create a central piece of software that can be used by all staff members of the Computer Science department of the University of Pretoria to maintain their academic publications.

However, the software will provide much greater functionality than merely listing all publications. One of the main features of the system will be the ability to add and edit publications, as well as specify a multitude of metadata items such as the title, (co)-authors, deadlines, progress towards completion, status (published, accepted, submitted etc.) and intended venue (conference, journal).

Additionally, the system will be used to provide users (who are authors of publications) with all manner of information regarding their Research Output Units and funding. This includes information such as expected units and funding, obtained units and funding, and shortfall of units.

The system will provide outputs in the form of an Excel spreadsheet which will illustrate the aforementioned author details both in a tabular fashion and graphically, most likely in the form of bar and/or line graphs. A sheet will be generated for each individual author, and a master sheet will be generated for the Research Leaders and Head of Department respectively.

A typical usage scenario of the system will be as follows:

- A user, who in this case is an author as well as the Head of Department (giving him/her administrator rights), logs in to the system.
- The user then adds a new publication on his/her profile page, filling in the relevant metadata items. The user will by default be added as an author.
- The user realises that he/she made an error while filling in the title, and edits the title appropriately.
- The units corresponding to the publication's venue (which have been assigned by the system) are automatically added to the user, and all unit-related calculations are made
- The user's profile page is updated with the new publication and all related information.
- The user then chooses to generate the Excel document which contains not only information about his/her Research Output Units and funding in a tabular and graphical fashion, but also similar information for all

other members of the department, who are users. This is because the user has Head of Department rights.

3 Background

Our group, Team Fox, had a meeting with the client, Ms Vreda Pieterse, on Wednesday, 17 February 2016. The client discussed that she requires the implementation of a system which will allow researchers to keep track of their publications, as well as view statistical information regarding their Research Output Units.

We have been given this topic for our mini project for the module COS 301. Furthermore, the mini project is to train us and help assist us in readying ourselves for the main project of this module.

This will give our group, Team Fox, the opportunity to gain experience on how to put together an SRS (Software Requirements Specification) that will help us and the client to better understand what is needed from the system and its functionality and how they would like to interact with the system. Consequently, if we better understand what our client wants the system to be, it will give us a better opportunity to implement a system that the client will be pleased with and which the client will want to use.

In addition, the system will be used by the Computer Science department at the University of Pretoria. Hopefully the system will improve on the system(s) currently in use and will provide a basic structure of good quality which will enable it to be updated and improved in the future of its software development life cycle.

Furthermore, the project might serve as an example to other students on how to implement such a system or it can be used as a basic structure and starting point for other institutes.

4 Architecture Requirements

The software architecture requirements include the access and integration requirements, quality requirements and architectural constraints. We will return to the software architecture again toward the end of the paper.

4.1 Architectural Scope

The system will allow the staff of the Computer Science department to keep a record of their research publications. It will allow the staff to add and remove papers on the system as well as add and edit the metadata about each publication. The system will allow users to view other user's profiles for when they need their details in order to add them as authors to their publications.

The system will send notifications to the users when their publications are reaching their deadlines. It will send these notifications via email.

The information that the system uses, such as the papers and their metadata and authors will be stored using a MongoDB database.

An Excel spreadsheet will be used for reporting purposes. It will list the staff members, their publications, the number of units allocated to each staff member, as well as budget information.

4.2 Access Channel Requirements

When considering the access channels of a target system, there are two main possibilities to consider: the human users of the system and whether any other systems need to be able to access the system. For both of these possibilities one then needs to consider through which ways they will access the system, what functionality the system requires these access channels to have and how these access channels will provide said functionality.

The system required by our client (which will from herewith be referred to as the system or the target system) is a system through which researchers can keep track of their publications. The users of the system, the researchers, need a channel through which they can use the system. The client does not require any other systems to be able to access our system or use the functionality of our system for a higher processing scheme, at this stage.

The client would like the creation of two platforms through which the users of the system can access the system, namely:

- A Web Interface
- A Mobile Application (On the Android platform)

Both of these will have to access our system through the Internet - for the Android Application, this will require requesting permissions. The aforementioned platforms will need an interface through which to access our system, known as an Application Programming Interface (API). This interface will have to enable the passing of data from the system to the channel through which the user accesses the system. Considering the general functionality our system provides, and that communication would take place through the Internet, this API would have to be RESTful.

REST stands for Representational State Transfer, and it is a software architectural style that deals with resources and what resources are accessed through using the HTTP user-oriented network protocol. This RESTful API would enable the transfer of data between our system and the platforms our client has chosen for users to access the system through - in our case over a network, namely the Internet.

Before we consider this API further and what data it shall pass and how our access channels will use the API, let us revisit the functionality the access platforms will need to have to provide the users with the target system's maximum functionality.

- User log in
- Search functionality
- Ability to open excel spreadsheets
- Viewing of plain text
- Viewing of lists, tables etc.
- Viewing of links
- Ability to view information
- Ability to add information
- Ability to change information
- Ability to remove information

For the viewing of text, lists, links and the like, this will have to do with the Graphical User Interface (GUI) that the access channel provides. In the case of the Web Interface this can be designed using HTML and its associated counterparts such as CSS, Bootstrap and more. In the case of the Android Application, a number of classes and functions will be used. For example in

the case of a list, one would use a ListView item on their user interface and this can be created using the android.widget.ListView class, and operations on said list will be done using the classes' associated functions.

The Search functionality would also need to be provided by the Graphical User Interface, but actual searching of the database used in our system will be done by the system itself, so our API would need to be able to request the system to do the search, when a user provides the required information for the search by interacting with the GUI.

For the ability to open Excel spreadsheets, in the Web Interface, this would imply the ability to download a file; while on an Android Application, this would imply the same thing as well as the improved ability to open the user devices' spreadsheet viewer.

As for the user log in functionality, users would provide user details through the GUI and the API would then need to pass user details for verification to the system and be able to receive a response about whether or not log in was successful or not. Another option that would be safer is the use of API authentication, this is a token based authentication in which the user would log in and if successful the system will respond with a unique token that can be used in future requests.

As we already know the information that will be displayed on these platforms GUIs will be provided through the RESTful API from our system. For this information, the API would need to get information from the system and be able to send changes including the addition and removal of information back to the system. Basically all the interaction between the system and these platforms will be done by the API with the following HTTP methods:

- GET – For reading information from the system
- PUT – For adding information to the system
- DELETE – For removing information from the system
- POST – For making changes/edits to information on the system
- OPTIONS – For getting operations that can be performed by the system

While these platforms vary, they will both access the system through this API and use the Internet to communicate with the system. However the ways they call the API, process and deal with the responses will vary slightly on the two platforms. The API would provide a standard way for accessing the system despite what channel may be in use. These access channels thus simply need to provide users with the functionality that the target system

requires on a GUI and use the API for providing users with the results and information from the system.

References

- `http://www.andrewhavens.com/posts/20/beginners-guide-to-creating-a-rest-api`
- `http://www.tutorialspoint.com/restful/restful_introduction.htm`

4.3 Quality Requirements

4.3.1 Performance

The system must work as efficiently as possible and not lag in delivering information to the user. Since this system will not be sending large amounts of data, it should not be difficult for the system to transfer the data speedily. In this implementation, we expect the network speed to dictate performance – not the system itself. Regardless, the system must be able to handle the full amount of users (approx. 100) concurrently without experiencing significant delays or errors. The success of this requirement can be measured in the time taken for a task to be completed under normal, and heavy-load scenarios

4.3.2 Reliability

It is imperative that the system not experience unnecessary downtime (period of in-operation of the system), thus preventing users from checking on or altering information on the system. An optimal uptime (period of normal operation of the system) within a month would be 99% and upwards. Additionally there should not be frequent errors within normal operating parameters of the system. This means that simple logins (sessions), alterations or additions to the database should not cause errors which affect the system as a whole. Additionally, minor human errors, such as omission of information, should not be allowed to propagate through the system. But the system should rather allow the user to fix the error immediately – which will also contribute towards the accuracy of information in the database.

4.3.3 Scalability

The system will initially be implemented to manage papers pertaining to one faculty, but it might be necessary in the future to expand it to manage more faculties. The maximum amount for one implementation to manage should not exceed 11, and as such it should be scalable to that amount of faculties

and the users associated with them. We make the assumption that the total users per faculty does not exceed 100 and, thus, the total on the system will never exceed 1100.

4.3.4 Usability

Users must be able to login, edit information and logout without hassle. The interface should not be cluttered and should not contain redundant information. As management of information on the system is critical, accessing said information should be as simple as possible.

4.3.5 Auditability

In order to track changes and/or errors on the system, all actions by users must be logged as they occur. This includes, but is not limited to, log-in attempts, editing of information (whether successful or not) and log-outs. All errors should also be logged in order to simplify error-tracking. The logging system must not interfere with the users' actions and must not significantly affect performance. Logs must be timestamped and must contain information on which user executed an action – or where an error occurred.

4.3.6 Security

The system will be protected by a login system which will require a user to enter an ID and a password. The ID and password will then be compared to users in the system's database and, if an appropriate user is found AND the passwords (entered and stored) match, may the user proceed to access the system. This system must not, under any circumstances, allow unauthorised users to enter the system. If a user enters incorrect information, the system must notify the user and request that they correct the input before allowing them to continue to the rest of the system.

4.4 Integration Requirements

As the system will be accessed remotely through the Internet, it needs to integrate with web services. The GUI will be integrated online with HTML, JavaScript, CSS and any frameworks or libraries the developer decides to use as long as the system itself remains compatible on all browsers and does not use browser-exclusive functionality. The system must be integrated with the server, and thus make use of appropriate server-side technology such s AJAX and PHP. In summary, the system needs to be accessed anywhere on any browser through any device regardless of screen resolution, computer Operating System or hardware specification. Additionally, the system must work with these technologies in a robust and consistent manner.

For mobile development, the system is required to integrate with the Android mobile Operating System. It must work correctly on all previous versions, and thus have no dependency on current feature sets. It must also be scalable to future Android releases and switching from account access through Android to web access and vice versa must be seamless and secure. On that note, the system must integrate well with security measures.

Externally, the system must be integrated with a database system such as LDAP, mongoDB or an appropriate alternative the developer decides on, provided that the system integrates with the database securely and it remains scalable. The system is also required to send emails to users, and thus it must have email integration either through external email servers or technologies, or an internally created one. Any libraries or packages needed to realise the functional requirements must be integrated with the other technologies being used. It was decided that the system does not need to integrate with Google Calender and thus it will not be using the Google API.

The system will generate reports in the form of spreadsheets and this must be integrated with the appropriate technology such as Microsoft Excel or LibreOffice Calc. The system will also regularly generate statistics and graphs so it is imperative the system integrates with statistical modelling technologies. As the system will be primarily an Internet-based application, the HTTP protocol suite will be primarily used for website data passing from client to server. The sending of emails should adhere to the SMTP suite and message passing through the system can be achieved through TCP.

All integration needs to be seamless and not interfere with system performance. Security is also a major concern, and thus technologies integrated must not threaten the system's security as a whole or other technologies. The system must remain portable to any Android device and to any browser. As the system may grow and shrink as users are added and deleted, the system must be scalable. The data stored in the system will also increase and thus

storage integration must be scalable as well. The system must remain reliable with each integrated technology added, and any errors must be able to be traced to the technology causing it.

4.5 Architecture Constraints

There are not too many architectural constraints to this system. However some constraints do exist, such as:

- The only technological constraint we have been given is the fact that we cannot link the system to any external resources besides for sending an email reminder, to the user, about the due date for their paper.
  - The only constraint with regard to this would be that we would not be able to ease the way in which the user could keep track of their due dates by *e.g. Adding the date directly to their google calendar*. Thus the user would manually need to add the date to their calendar or create a reminder for themselves (if they want to). Another problem could be that the email reminder that is sent by the system could be cluttered between all the other emails that have been received, by the user, and thus the user may forget about the reminder that was sent by the system.
- The role of the user in the system *(i.e. Author, Research Leader, Administrator, Head of Department)* determines their permissions on the system.
  - Author - May only edit metadata about a paper they are working on or have already published.
  - Research Leader - May see all papers in his or her research group.
  - Administrator/Head of Department - Should be able to view everything in the system (including all papers published, in progress or discontinued) as well as the number of units each user has earned. They may also add or remove users.
- The system must only cater for the Computer Science Department of the University of Pretoria. Users may not be anyone else outside the department.
- When viewing profiles of other users, one may not see the papers of the other users - despite the progress of these papers - or the number of units they have earned in total.

- However when viewing ones own profile, one may view their papers - regardless of progress - and also may view the number of units they have earned in total.

- Developing the system for the web can be very different from developing it for Android. This does, however, depend on the language(s) used to develop the Android application. If it is coded using web based languages it would be easier to port it to android. However if it is developed using Java the web-based system would not be very portable to Android.

- Additionally if this system ever needed to be ported to other mobile OS's such as iOS or Windows Mobile it would be difficult due to the difference in development language for each OS.

- Lastly no papers, once added, may be deleted. Thus the database could become very large and make system backup and restore more difficult. Also there is no automatic backup of the data in the system, thus it would need to be done manually.

We will return to the software architecture requirements again later on in the paper.

5 Functional requirements and application design

This section discusses the application functionality required by users (and other stakeholders).

5.1 Use case prioritization

Critical

- Adding a conference paper
- Adding an author to a conference paper
- User being able to see all papers they have added or co-authored
- Adding a researcher to a research group
- Editing publication metadata
- Research leader being able to view all papers and their progress
- The state of the paper (submitted, waiting, rejected, published)
- Functionality to back up information
- Add and remove authors any time of the paper
- Show history of papers
- Staff members being able to access the portal

Important

- Head Of Department being able to view all papers
- The sequence of authors(primary,second etc.)
- Log all activity
- Keep track of units, showing charts to see if authors meet the target
- Count units only when paper has been published
- U.P. is the default occupational address of all papers
- Show the intend venue of papers and the type of the paper

- Send a reminder of when the paper is due
- Units allocated to each venue appear by default once they has been stored
- Search for an author
- Head of department being able to view all units for all staff members
- An Administrator having complete access to the portal even on behalf of other users
- Head of Department having complete access to the portal even on behalf of other users

Nice to have

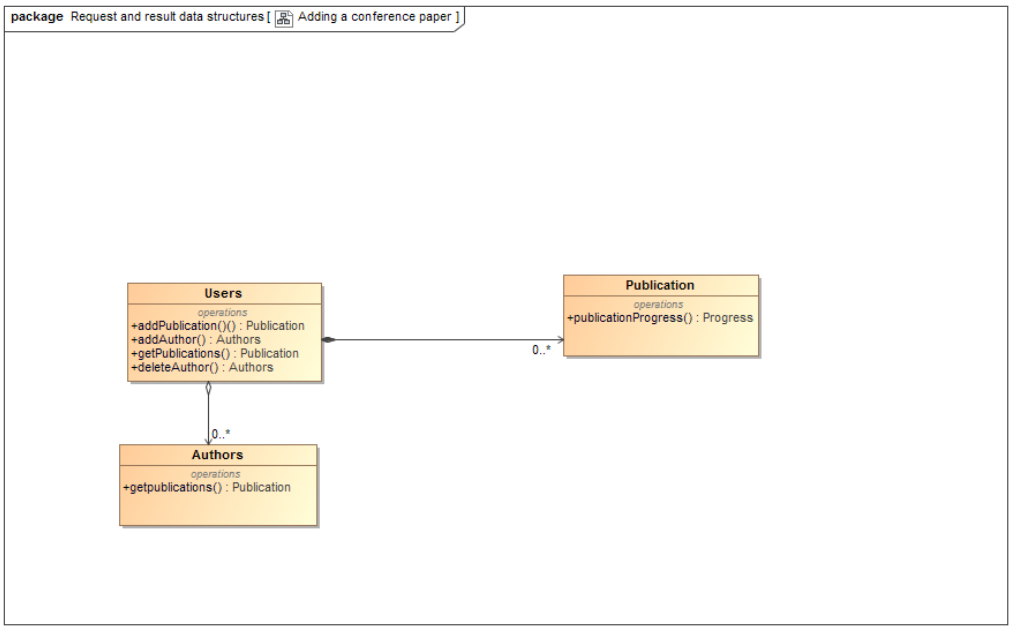
- Profile of the researcher
- Scalability to provide functionality for multiple departments
- A user who is not an author adding a paper for someone else

5.2 Use case/Services contracts

Adding a conference paper:

- Preconditions
  - User must be a staff member
  - User must be logged in
  - A paper must have at least one author
  - Primary author must be specified
  - User must enter the metadata about the paper
- Postconditions
  - Conference paper successfully added
- Exceptions
  - User can create a paper but does not have to be an author

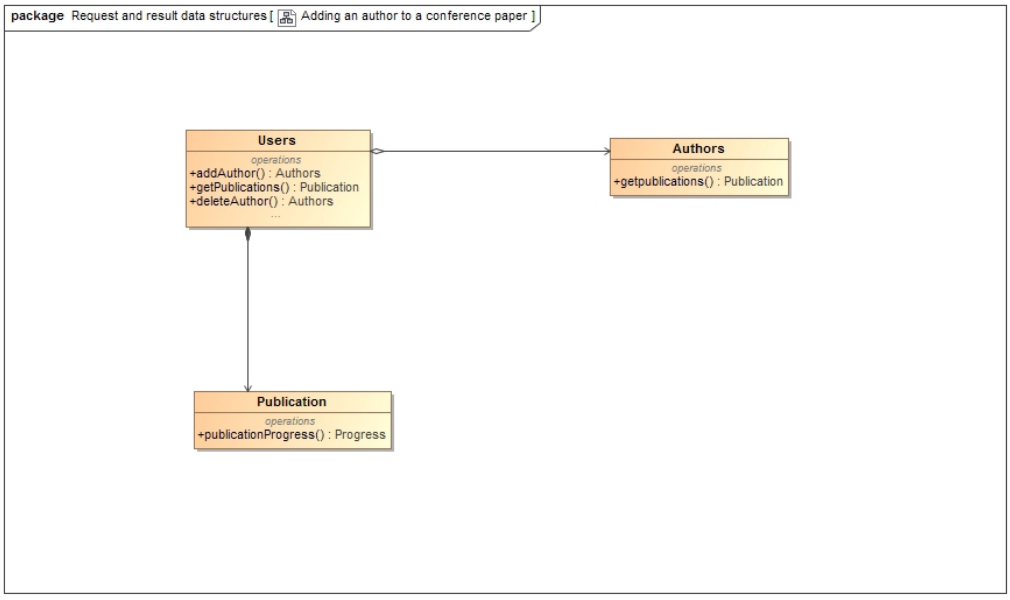
Figure 1: Adding a conference paper



Adding an author to a conference paper:

- Preconditions
  - User must be a staff member
  - User must be logged in
  - User can specify the number of co-authors
- Postconditions
  - Author would be added to a paper
- Exceptions
  - Author does not have to be a user

Figure 2: Adding an author to a conference paper

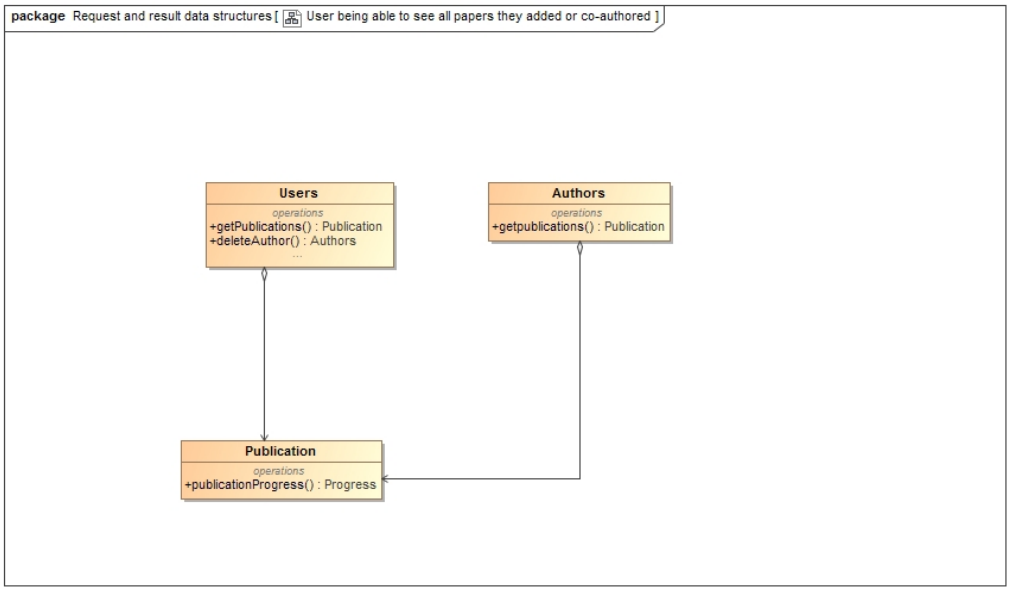


User being able to see all papers they have added or co-authored:

- Preconditions
  - User must be logged in
  - User must be an author or co-author to at least one paper
- Postconditions

– User will be able to view their papers

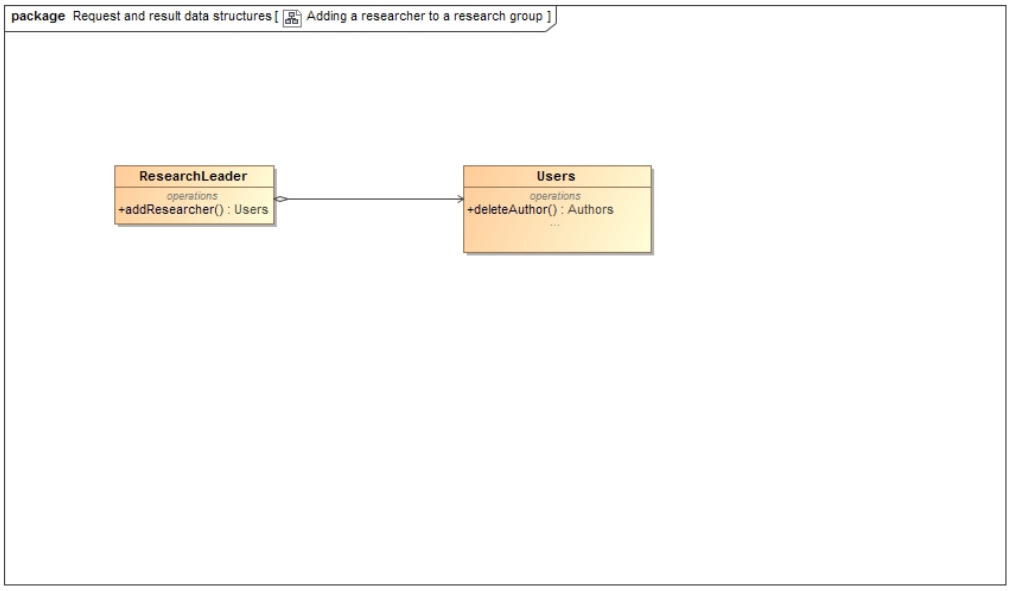
Figure 3: Users being able to view all authored/co-authored papers



Adding a researcher to a research group:

- Preconditions
  - User adding the researcher must be a researcher leader, Head of Department or an administrator
- Postconditions
  - Researcher added to research group

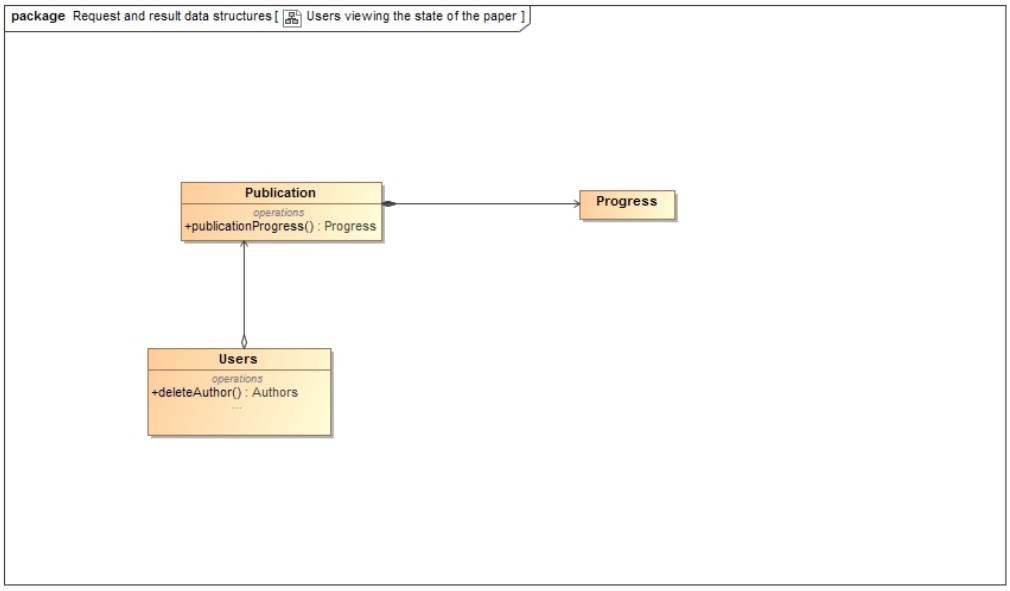
Figure 4: Adding a researcher to a research group



The state of the paper (submitted, waiting, rejected, published):

- Preconditions
  - The paper must have already been added to the system
- Postconditions
  - The user can view the status of the paper

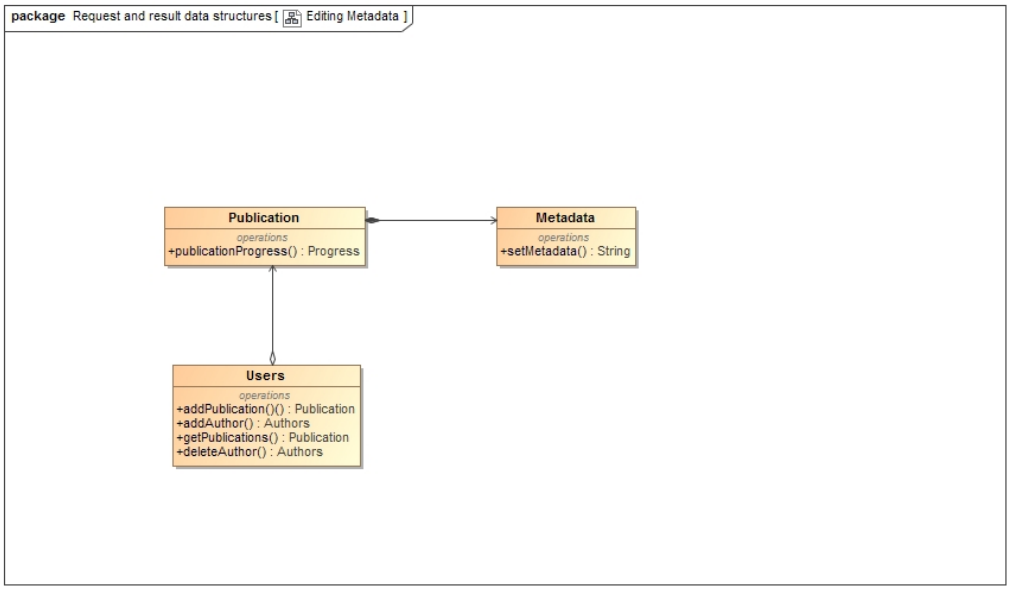
Figure 5: Viewing the current paper state



Editing metadata:

- Preconditions
  - The user must be the author or co-author of the paper
  - The paper must already be added in the system
- Postconditions
  - The user is successful in editing the meta data of the paper
- Exceptions
  - An administrator can edit the meta data of any paper in the system
  - The head of department can edit the meta data of any paper in the system
  - A research leader can edit the meta data of any paper in their research group

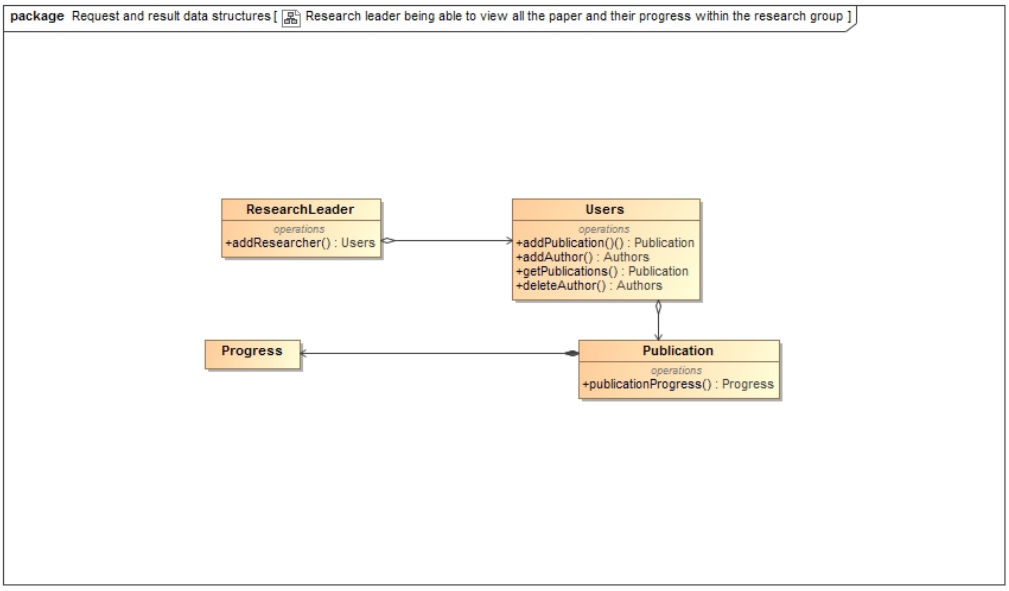
Figure 6: Editing metadata



Research leader being able to view all papers and their progress:

- Preconditions
  - The user must be the research leader of the research group
- Postconditions
  - The user will be able to view any paper in the research group
- Exceptions
  - The head of department and administrator can also view the papers in the research group

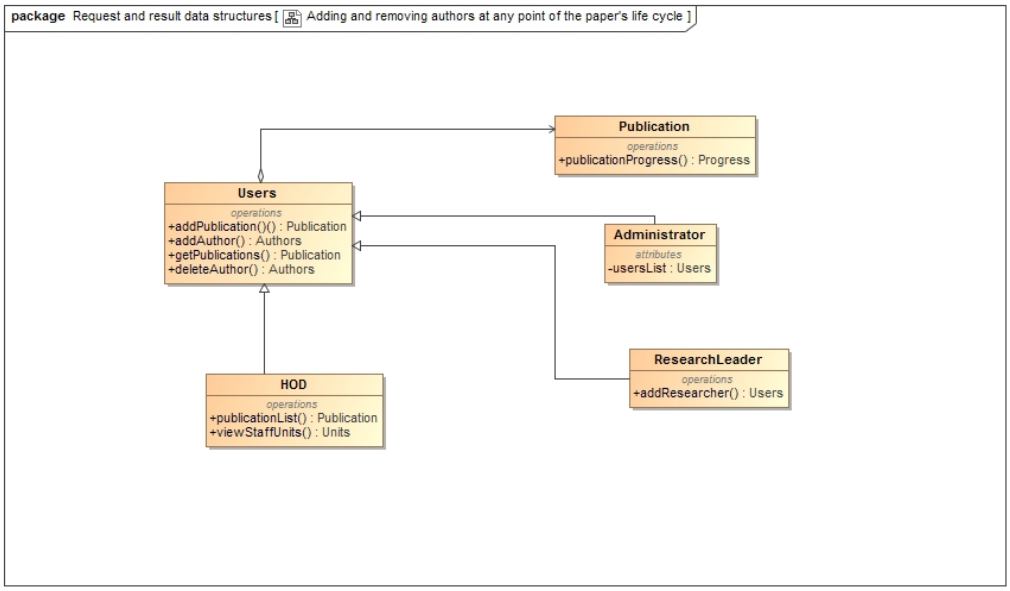
Figure 7: Research leader being able to view relevant paper progress



Add and remove authors at any point in the life cycle of the paper:

- Preconditions
  - The user must be the primary author of the paper
- Postconditions
  - The user would be successful in adding or removing authors to or from the paper

Figure 8: Adding and removing authors at any point in the life cycle of a paper



Show history of papers:

- Preconditions
  - The user can only view the history of their own papers
- Postconditions
  - The user will successfully view their papers' history
- Exceptions
  - The head of department can view the history of any paper
  - The administrator can view the history of any paper
  - The research leader can view the history of any paper in their research group

Staff members being able to access the portal:

- Preconditions
  - Staff members must have profiles on the system
- Postconditions

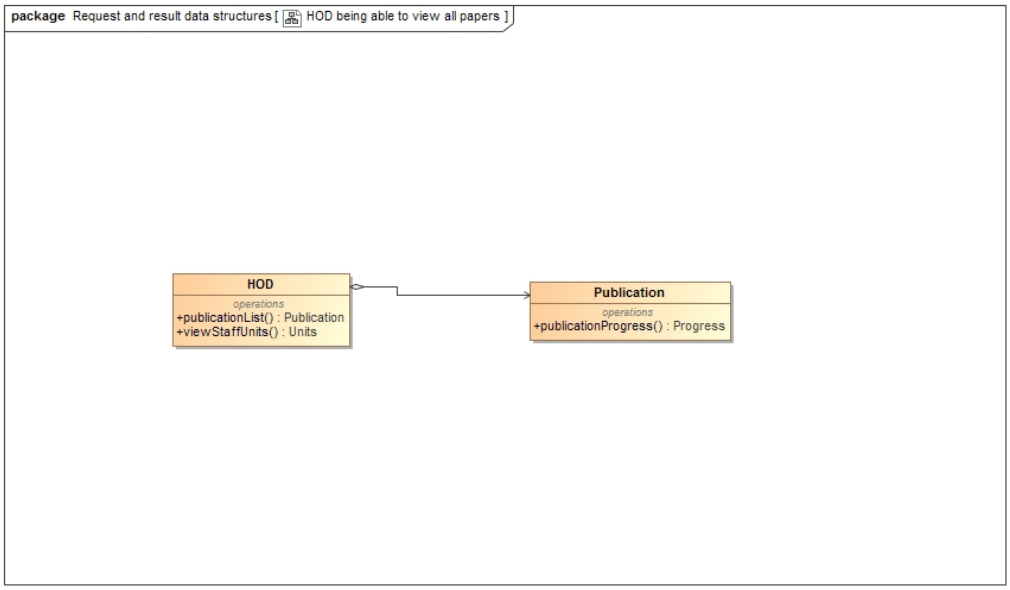
- Staff members successfully access the portal

- Exceptions

Head Of Department being able to view all papers:

- Preconditions
  - The user type must be the head of department
  - There can only be one head of department
- Postconditions
  - The head of department can successfully view all papers

Figure 9: H.O.D. being able to view all papers



The sequence of authors(primary,second etc.):

- Preconditions
  - The sequence of authors must be specified by the user that created the paper
- Postconditions
  - The sequence of authors is specified

Count units only when paper has been published:

- Preconditions
  - Units must be assigned to the paper
  - Paper must already be published
- Postconditions
  - Units for the paper are counted

U.P. is the default occupational address of all papers:

- Preconditions
  - A user profile must be in the process of being created
- Postconditions
  - The default institution for every profile will be U.p.

Show the intended venue of paper and the type of the paper:

- Preconditions
  - Must be done by a user who is about to create a paper or edited by an author or co-author
- Postconditions
  - Intended venue and type of paper is shown

Send a reminder of when the paper is due:

- Preconditions
  - User to be sent reminder must be an author or co-author of the paper
- Postconditions
  - Reminder is sent to the user about when the paper is due

Venue units appear by default once they has been stored:

- Preconditions
  - User must specify the units allocated to the paper

- Postconditions
  - Units for the paper appear by default

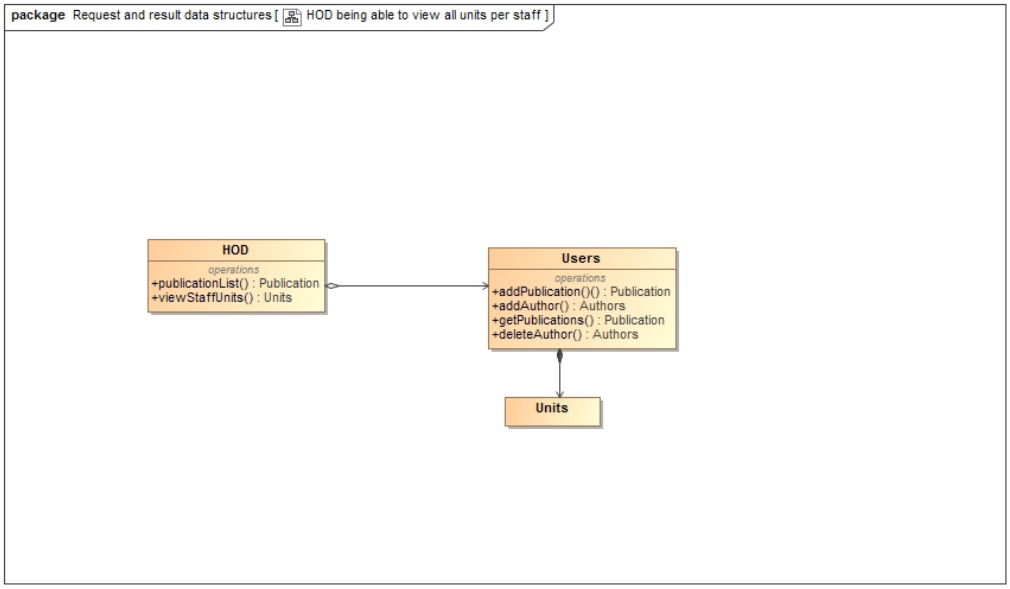
Search for an author:

- Preconditions
  - The author must be already added into the system
  - The user searching for the author must be logged in
- Postconditions
  - The author is found if they exist

Head of department being able to view all units per staff:

- Preconditions
  - User must be head of department
- Postconditions
  - head of department able to view the units allocated to each staff

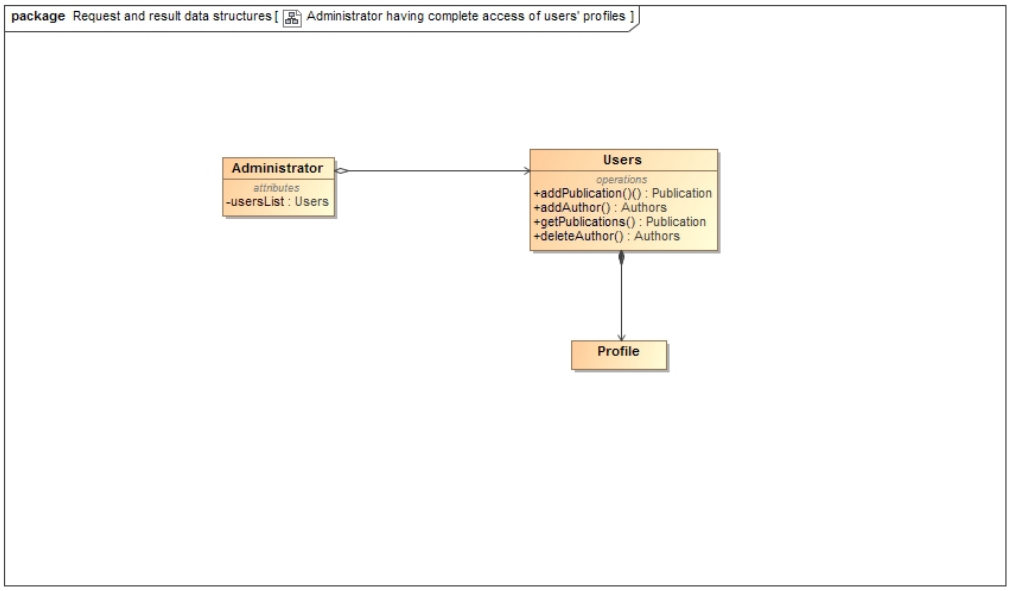
Figure 10: H.O.D. being able to view units per staff member



An Administrator having complete access to the portal even on behalf of other users:

- Preconditions
  - The user must be an administrator
  - The user the administrator is accessing must exist
- Postconditions
  - The administrator successfully have complete access to the portal

Figure 11: An administrator having access to user profiles



Profile of the researcher:

- Preconditions
  - The researcher must be logged in
  - The researcher must already have a profile on the system
- Postconditions
  - The researcher can successfully view their profile

A user who is not an author adding a paper for someone else:

- Preconditions
  - The user must be a staff member
- Postconditions
  - The user successfully creates a paper that another user is an author of

5.3 Required functionality

This section describes the functionality required of the system, and illustrates these requirements by means of use case diagrams.

Figure 12: Add a Paper

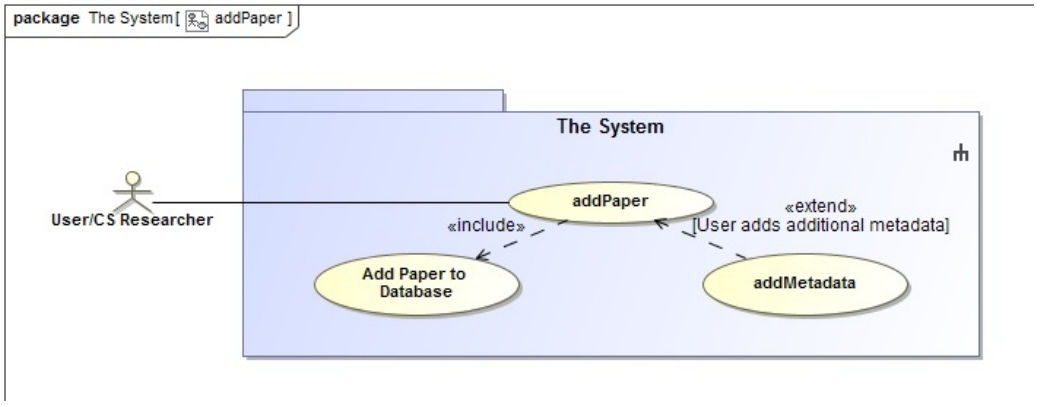


Figure 13: Add Metadata

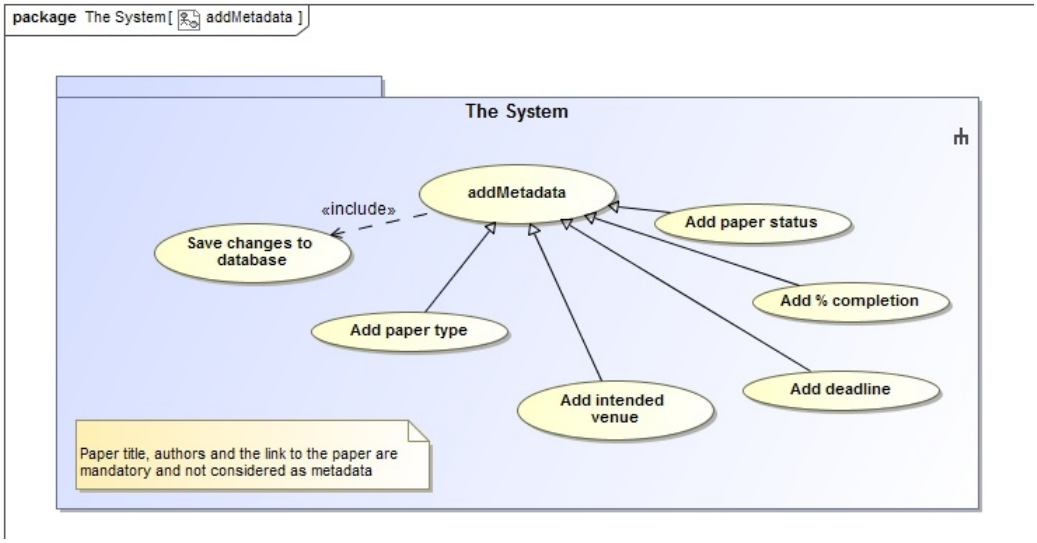


Figure 14: Edit Paper Data

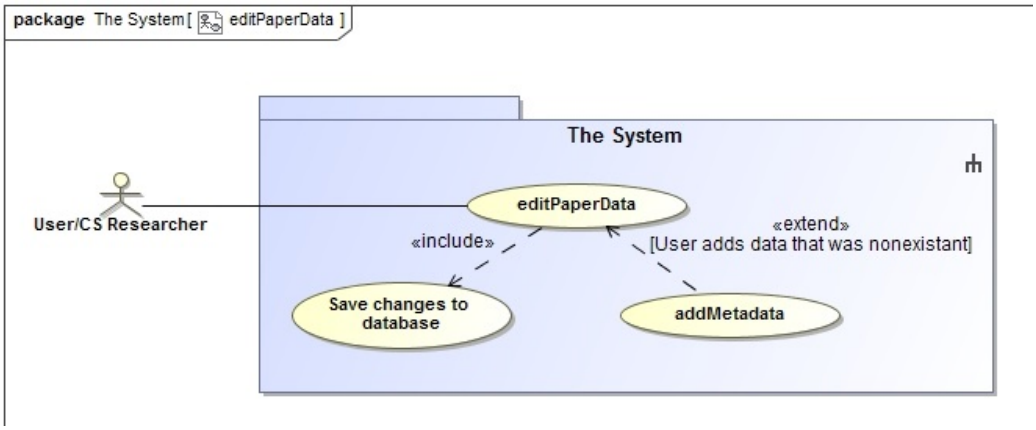


Figure 15: View Authored Papers

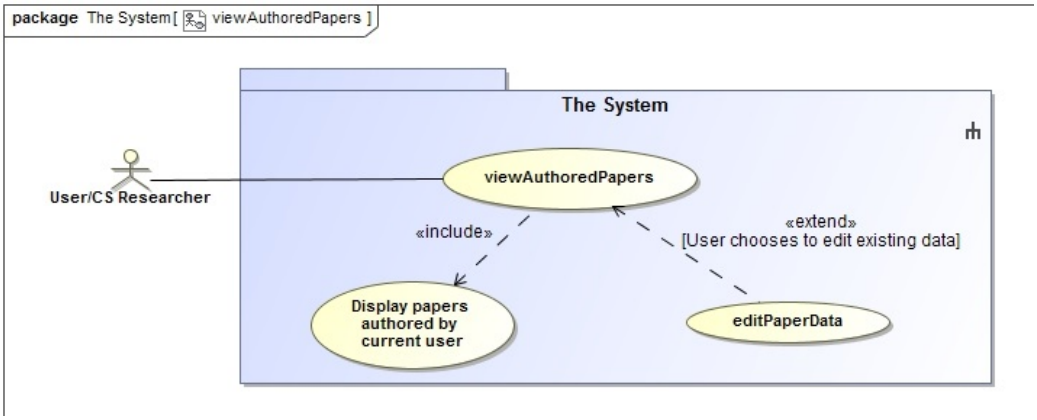


Figure 16: View Research Group Papers

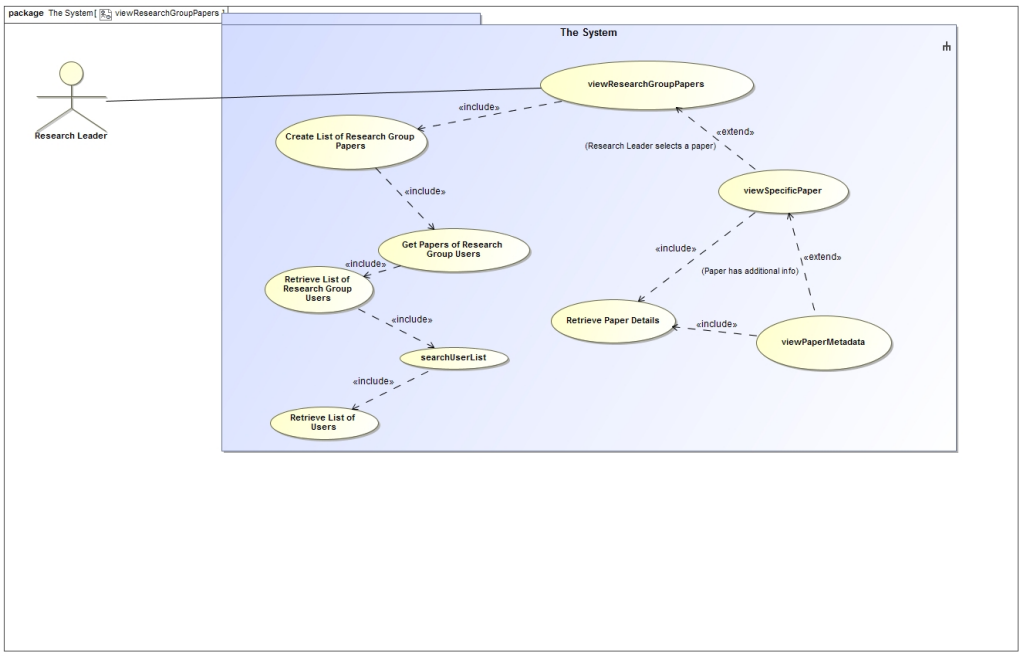


Figure 17: View All Papers

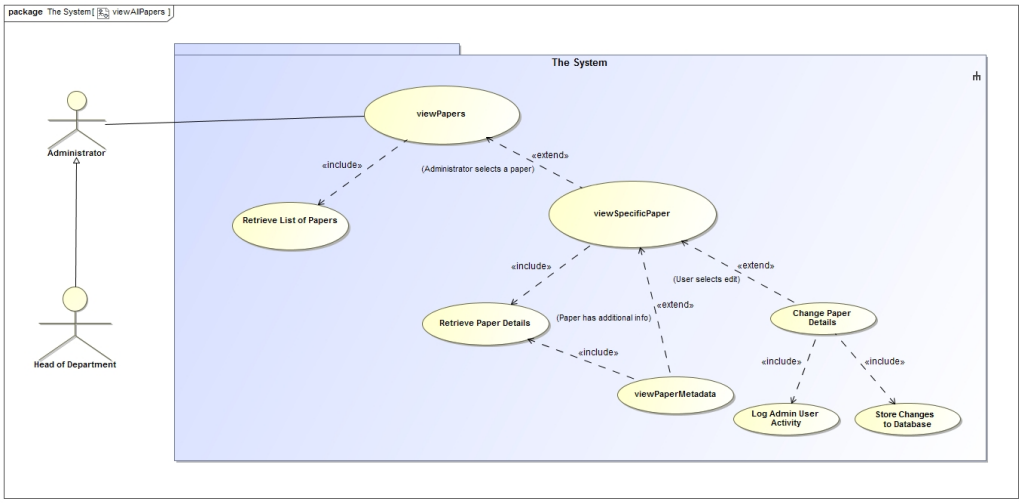


Figure 18: Add User

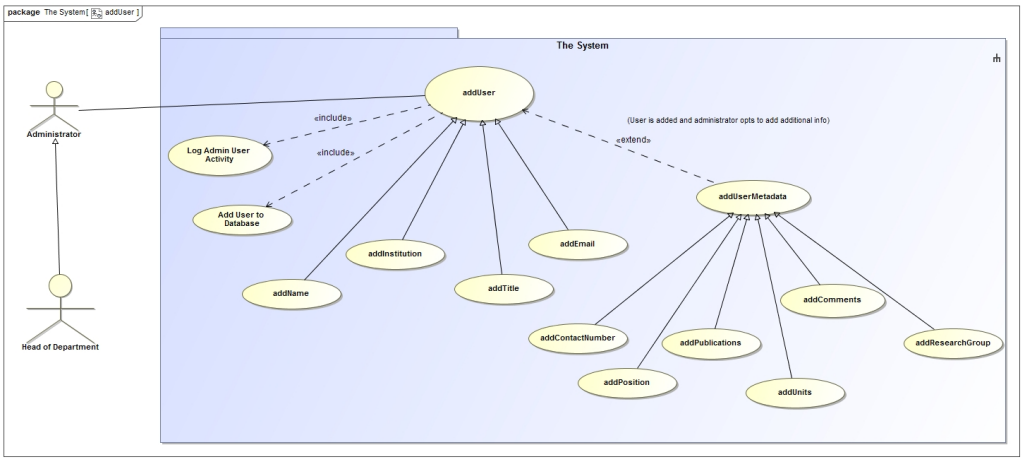


Figure 19: Remove User

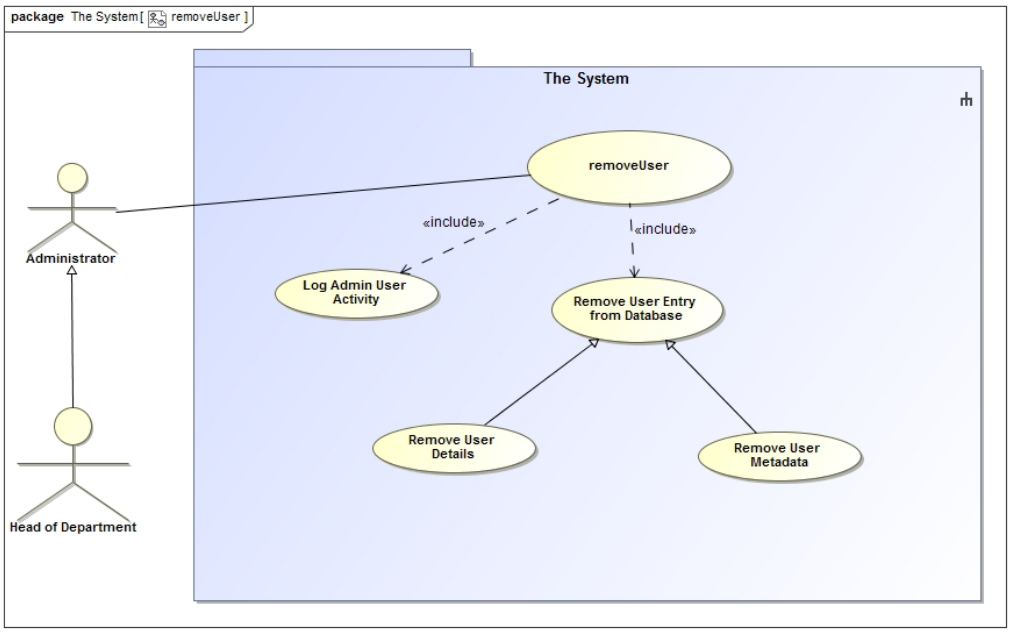


Figure 20: Add Research Group User

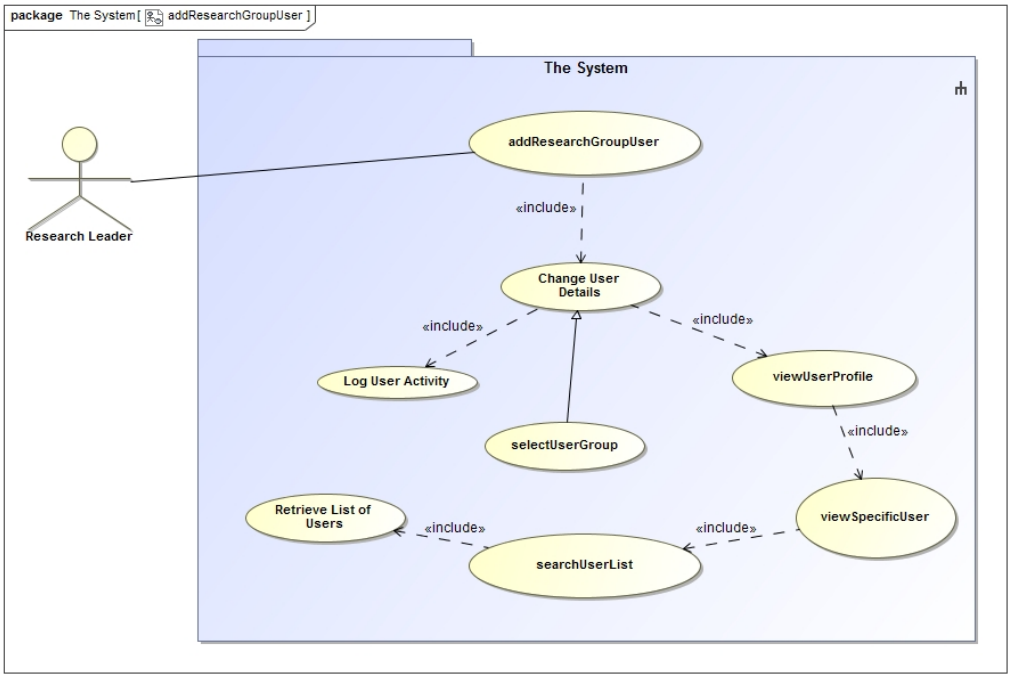


Figure 21: Remove Research Group User

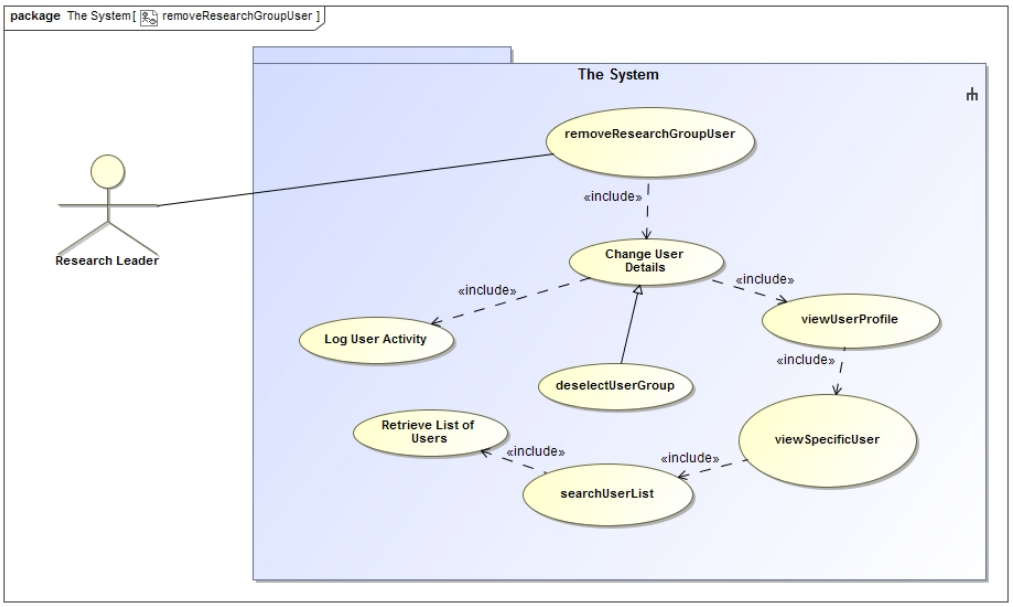


Figure 22: View Research Group Progress

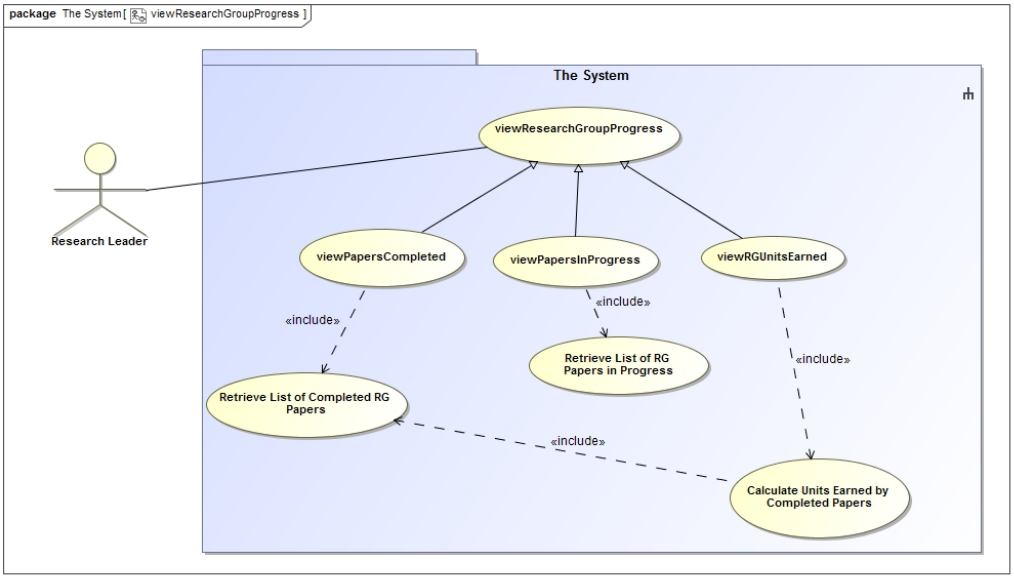


Figure 23: View Research Group Users

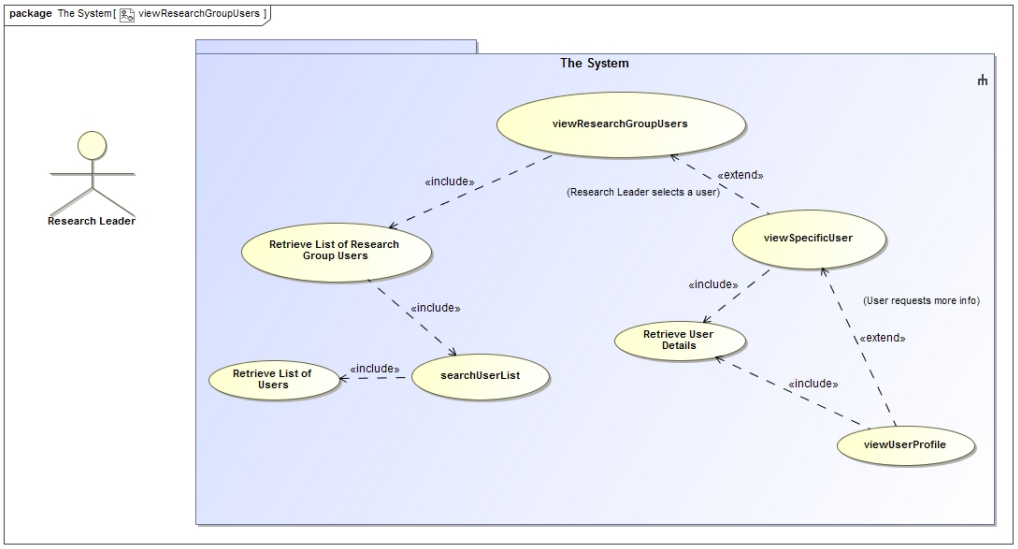


Figure 24: View Units Earned

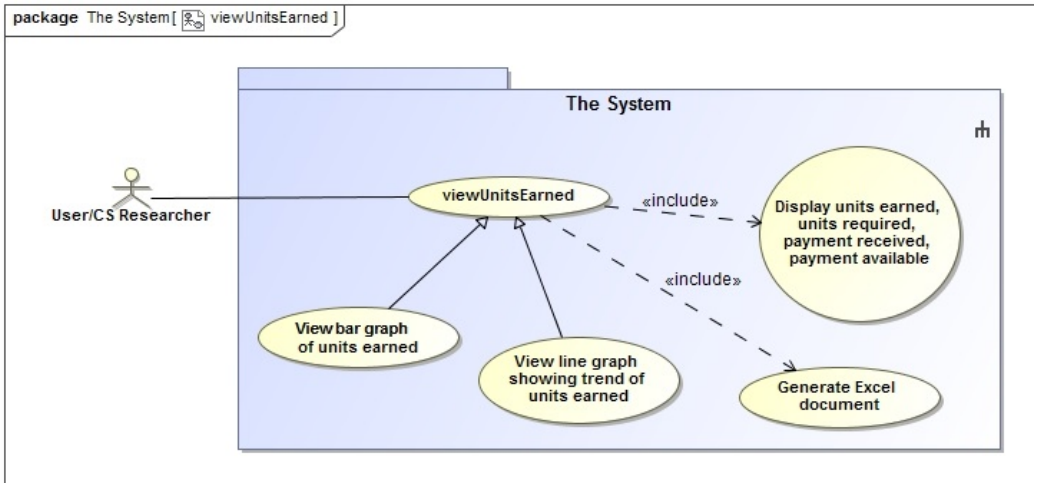


Figure 25: Edit Profile

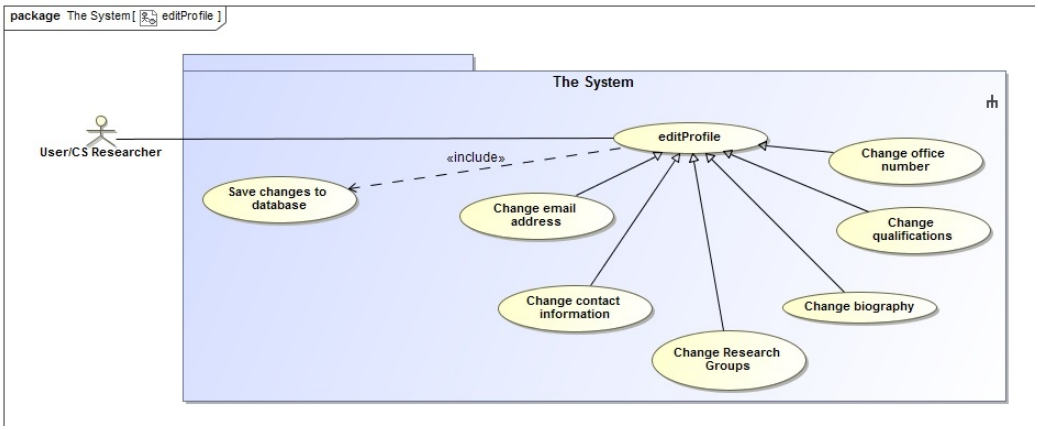


Figure 26: View Profile

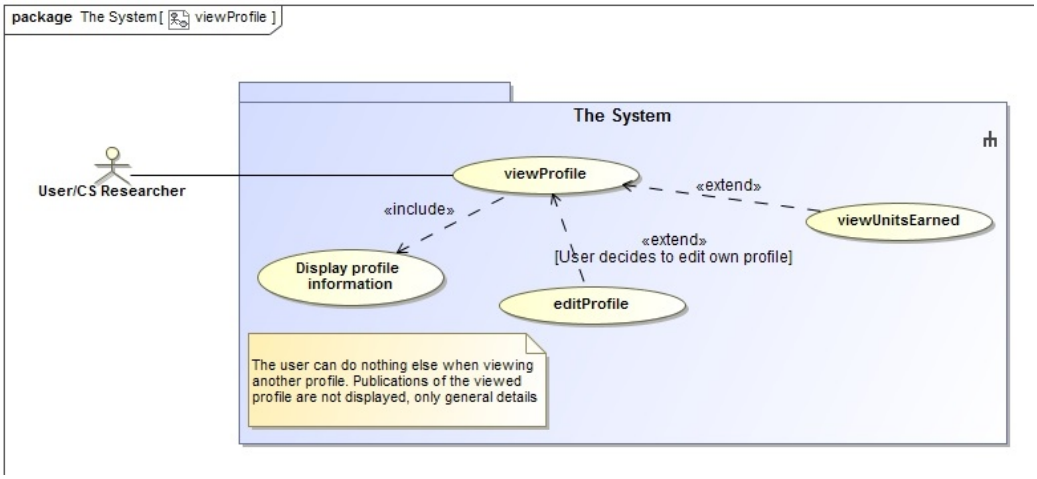


Figure 27: View All Users

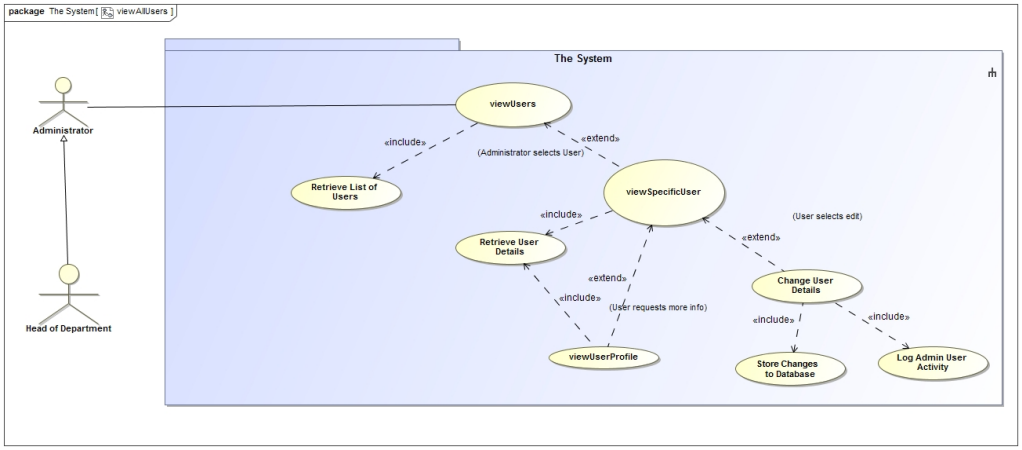


Figure 28: Log In

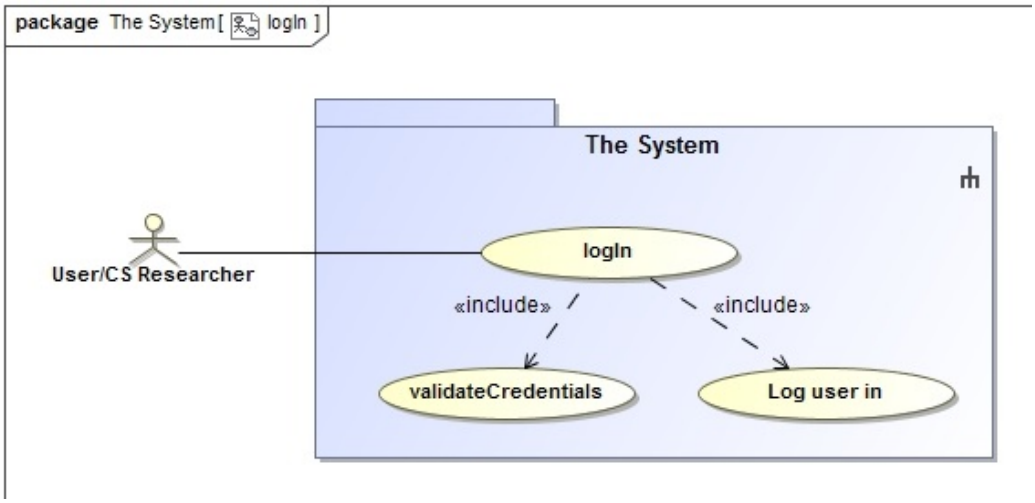


Figure 29: Log Out

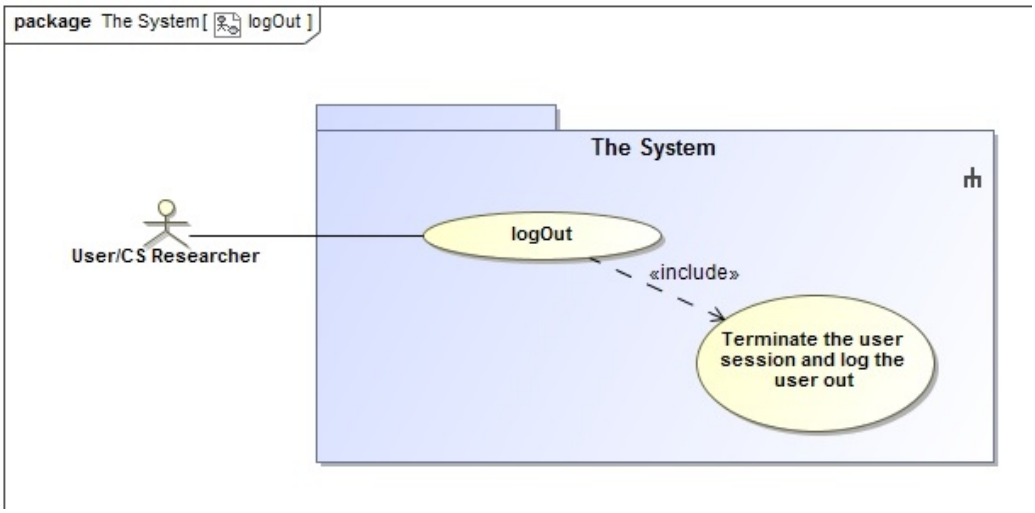


Figure 30: View Activity Log

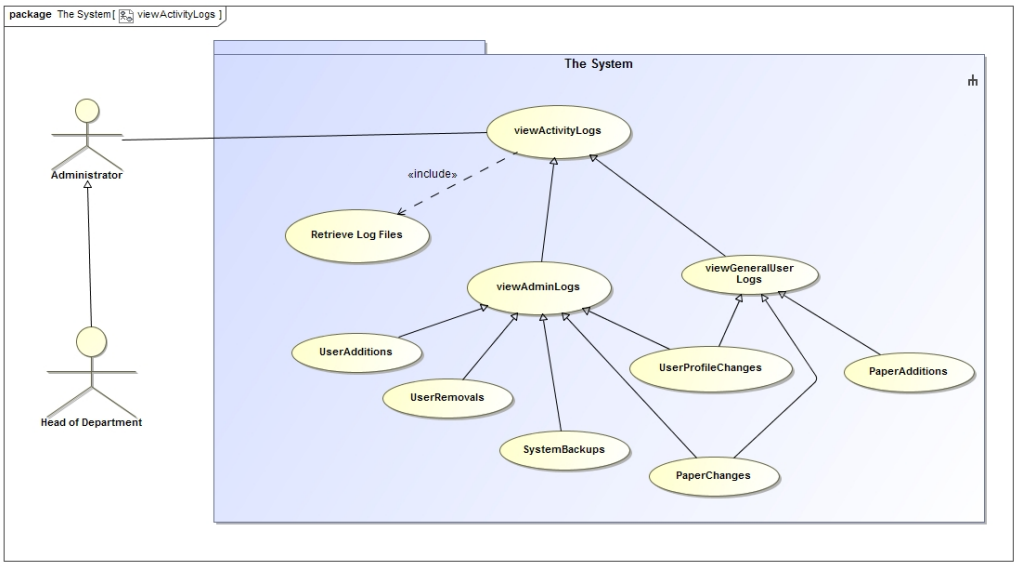
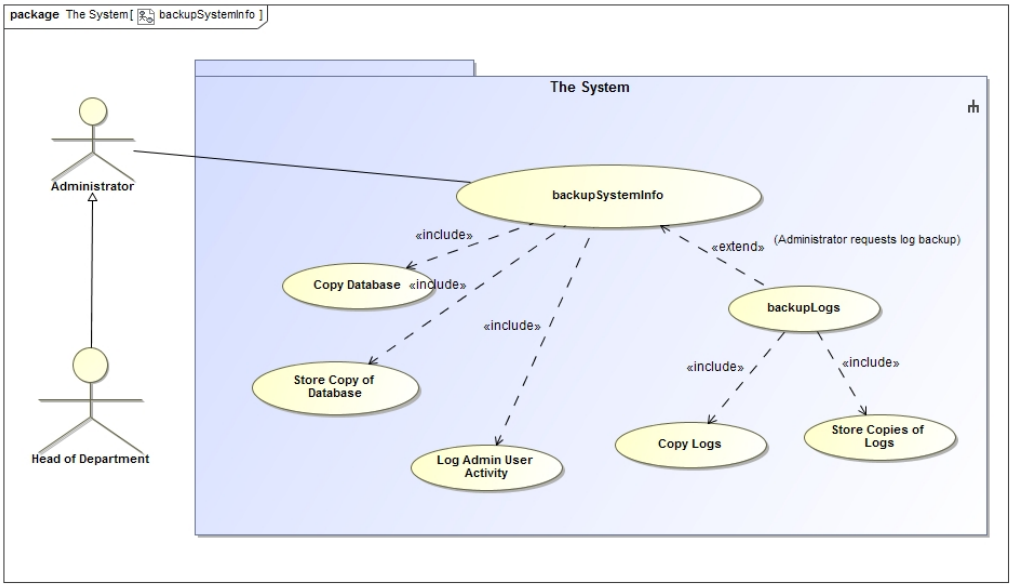


Figure 31: Back Up System Info



5.4 Process specifications

This section illustrates the requirements around processes in the system which need to be followed. Activity diagrams are used to illustrate these process specifications.

Figure 32: Log In

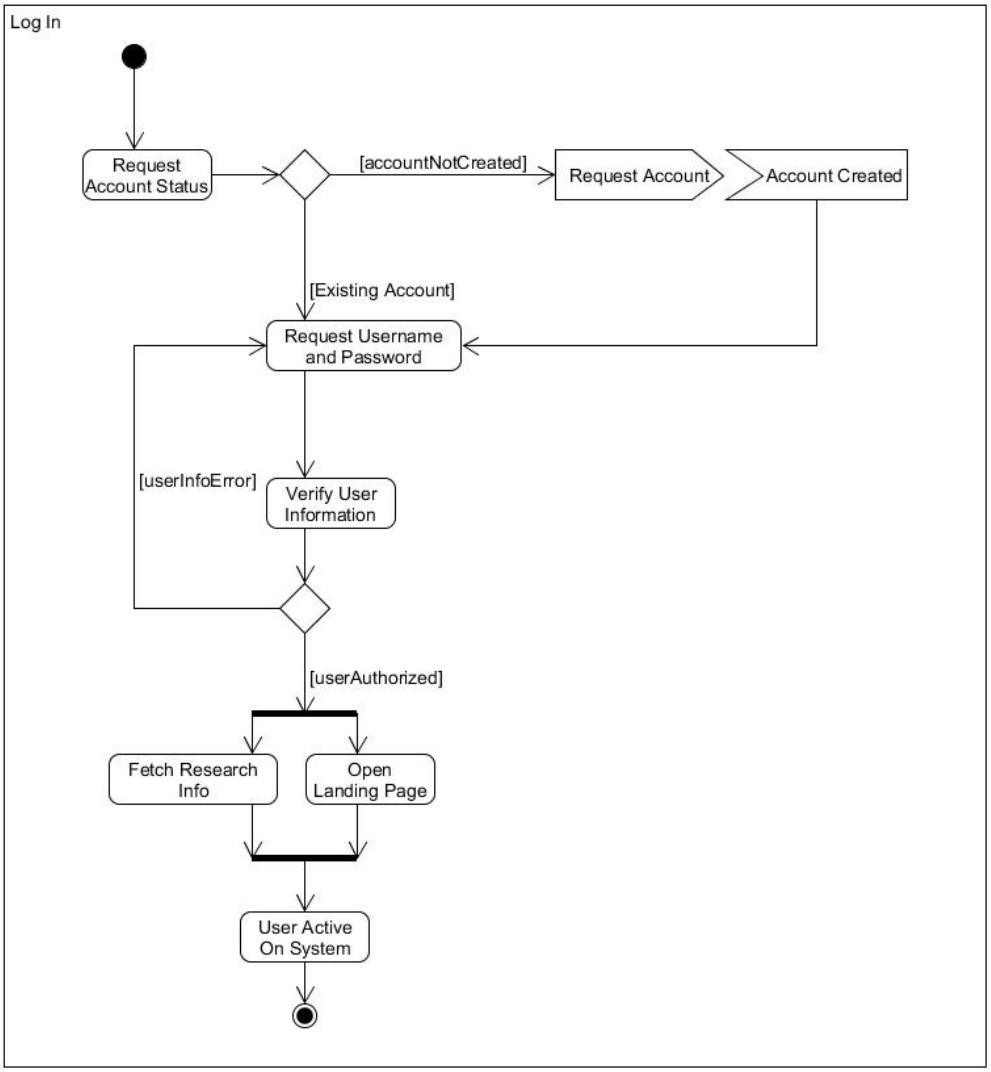


Figure 33: Add Paper

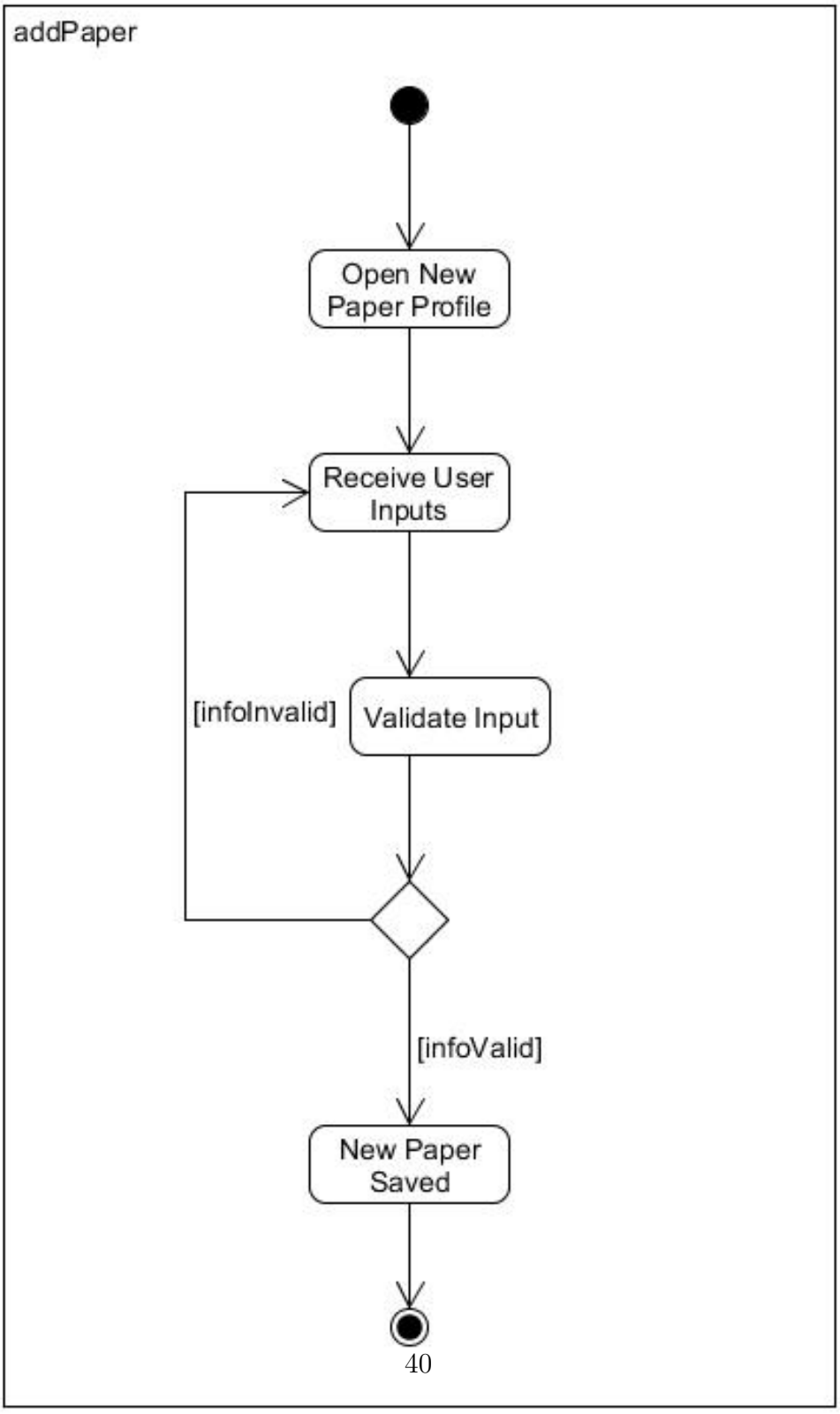


Figure 34: Edit Paper

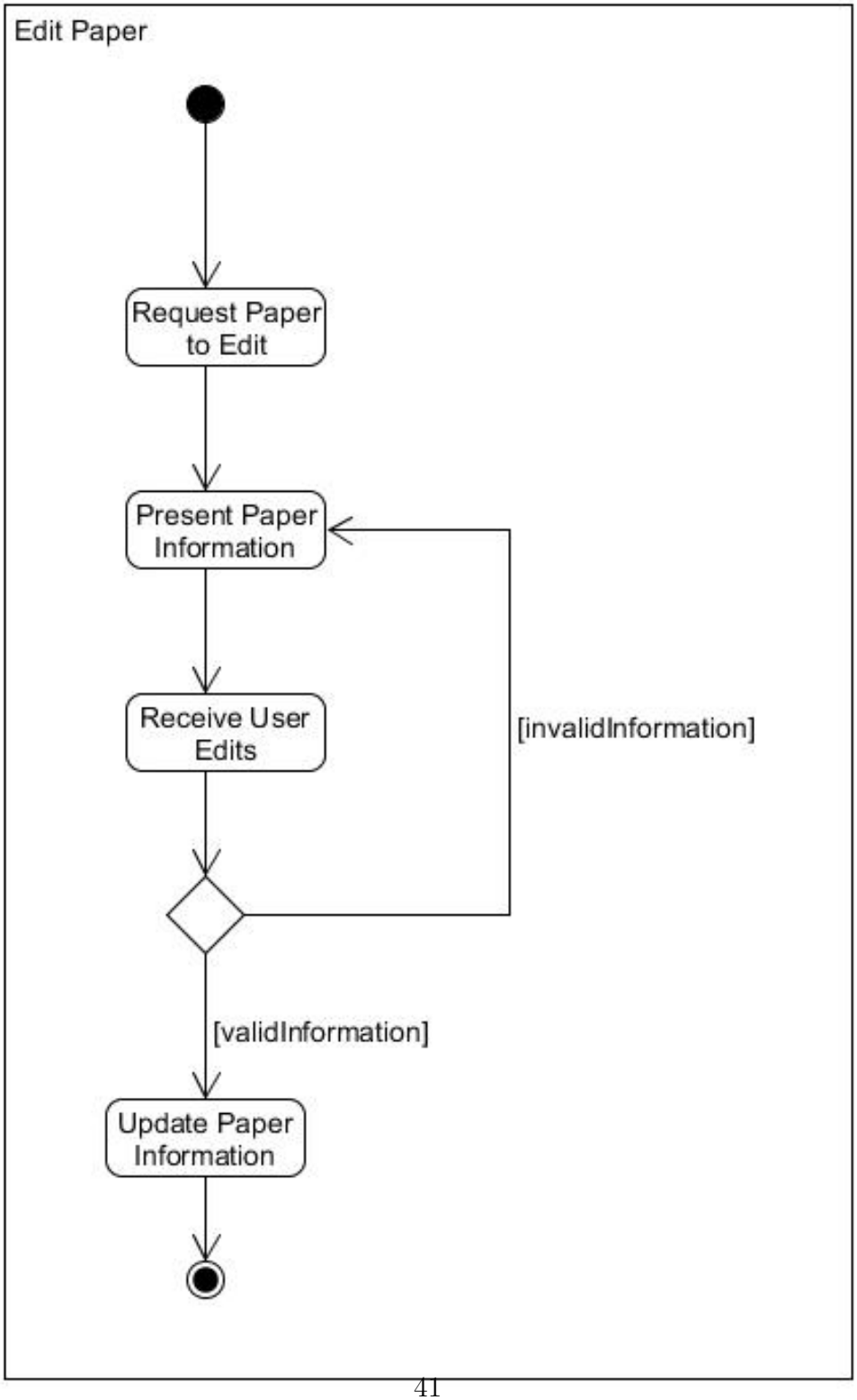


Figure 35: Generate Report

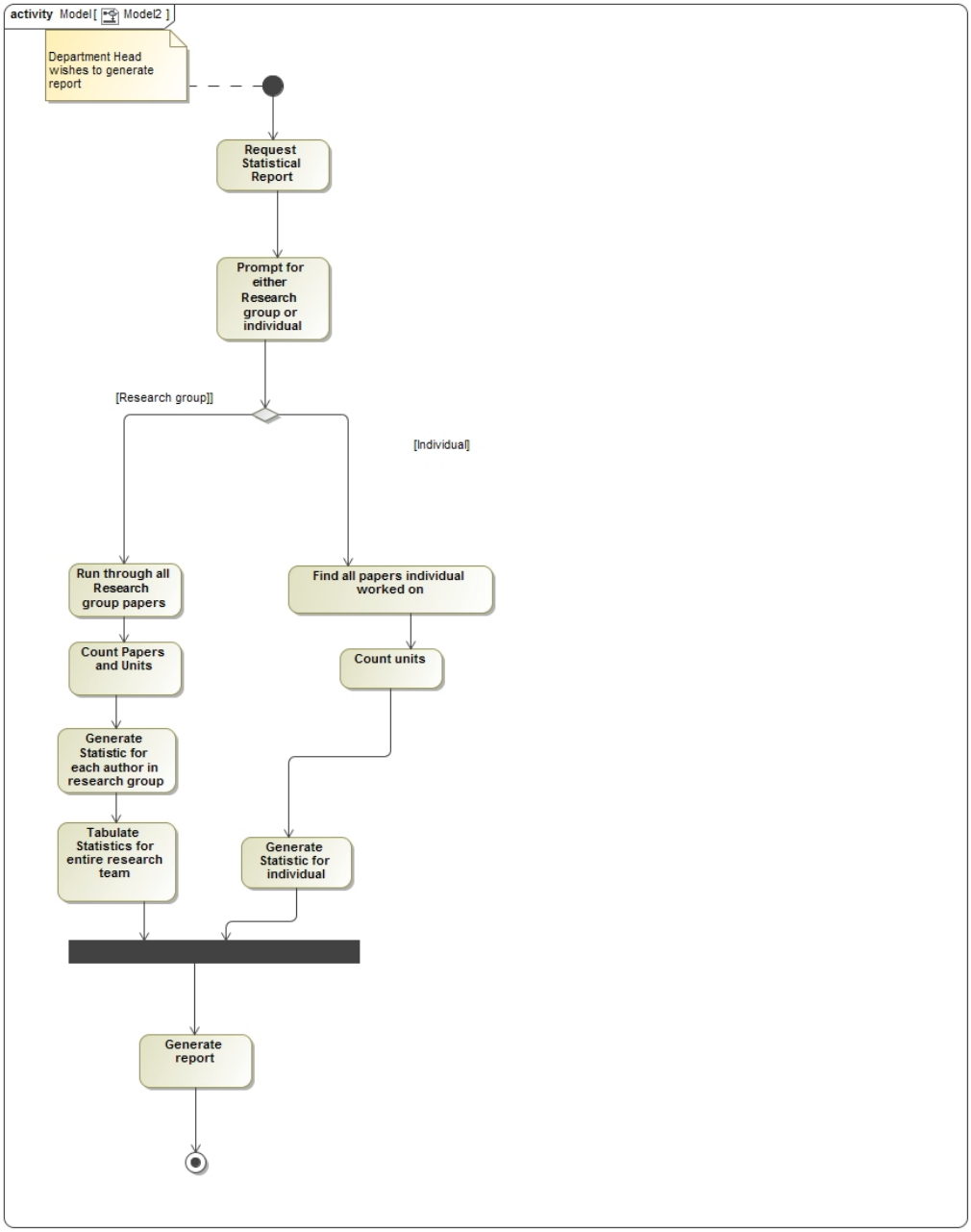
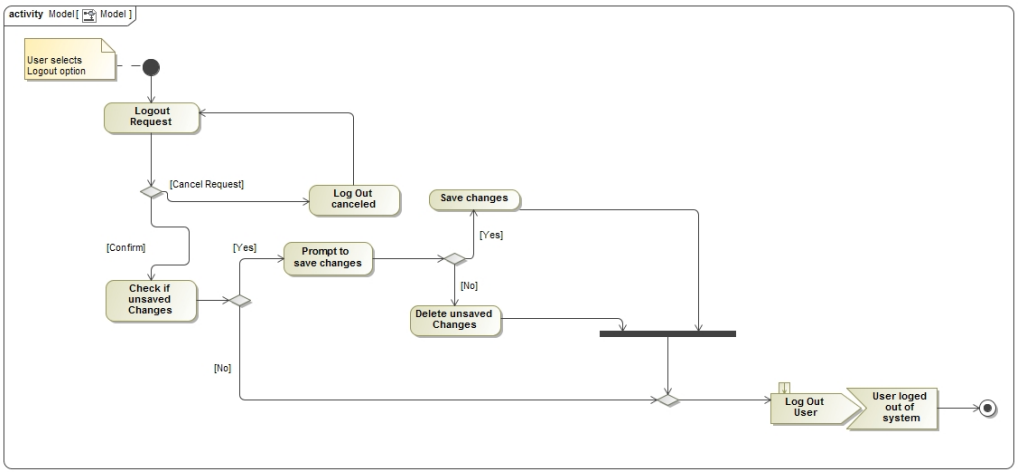


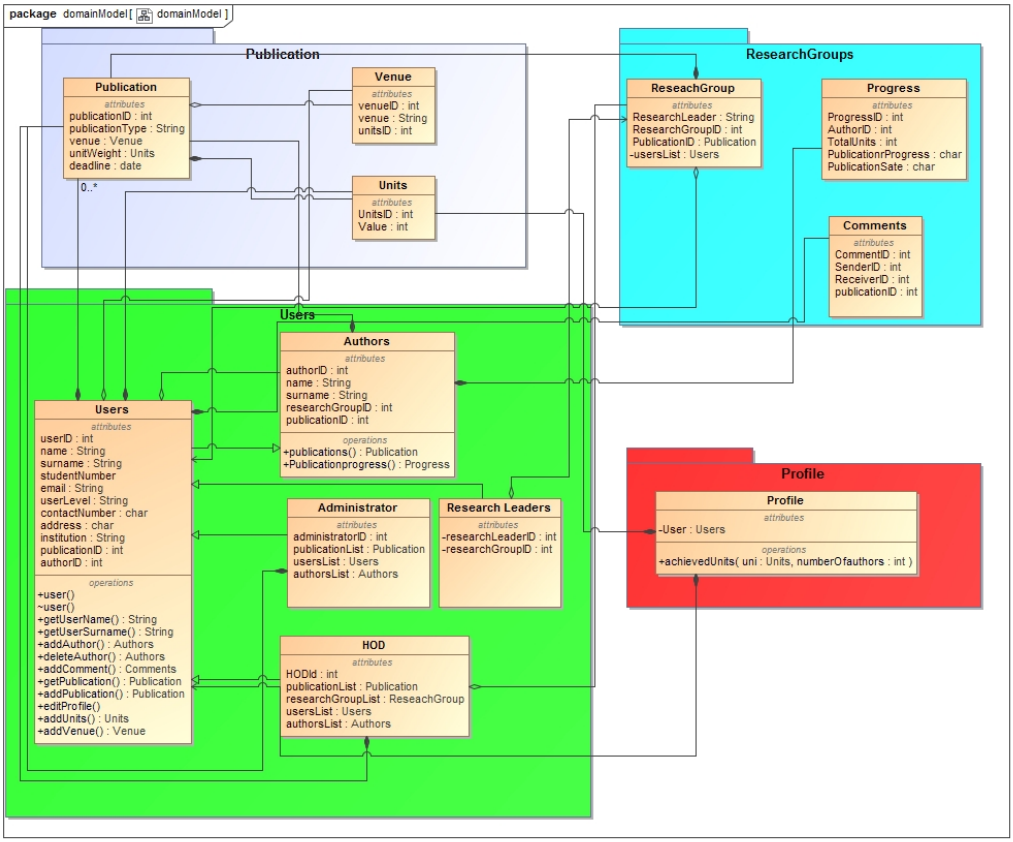
Figure 36: Log Out



5.5 Domain Model

In this section, the data structure requirements of the system are specified in a technology neutral way, by making use of UML class diagrams.

Figure 37: Domain Model



6 Software Architecture

6.1 Architectural Patterns or Styles

Our research team has decided to use the MVC (Model-View-Controller)- and Layering architectural patterns for the research publication system for the mini project. Since the system should have a web-interface and a mobile Android interface, it is necessary to divide the system into parts where the two interfaces are independent of each other, but both can have concurrent access to a central database. In addition, this is necessary, because the interfaces will likely use different client-side technologies for the graphical user interface and client logic.

The structure of MVC in the system:

- Model

  The Model represents the database and how the user and research data is structured, organized and stored in the database on the server-side. Furthermore, it provides an interface through the written server software by which the model can be manipulated from user interaction through the controller.

- Controller

  The controller partly represents the logic and the user-functionality of the system and is downloaded or exists on the client's side of the system. The controller interfaces with the model and the view where the controller;for example, verifies user-input and determines the validity of it; consequently, the controller manipulates the model through the server-side software interface and the user's view is updated with the updated data stored in the model.

- View

  The view represents the GUI(Graphical User Interface) that is displayed to the user and how it looks to the user when he/she interacts with it. The view gets updated when the model has been manipulated by some user's interaction with the controller.

The MVC architectural pattern simplifies and modularizes the design of the research publication system in the following way:

- The GUI's of the website- and Android applications can be developed independent from each other.

- The client-side software and functionality of the website- and Android applications can be developed independent from each other.

- The database and server software can be developed independent from which client-side programming languages will be used and can be re-distributed to work with other controllers and views.

The Layering architectural pattern makes use of a set of programmed layers. Each layer has been implemented to provide a certain unique functionality and can be used or reused by certain other layers. By making use of the Layering architectural pattern it can make the development life cycle of the research publication system more efficient in the following ways:

- Layers can be tested and debugged individually.

- Layers can be reused in other layered systems.

- Layers can be updated and improved to replace the older ones.

- Layers can be used to improve the security of the system; for example, all user-input must go through a security layer before it can be transmitted to other layers.

- Layers can help the system to be formed as a united whole.

- Layers helps the development of the system to be less complex, because the functionality is integrated into their separate layers to be worked on separately.

Examples of layers that can be used in the research publication system:

- A client-handler layer to handle what happens with user-input. Two different client-handlers might be implemented for web- and Android applications.

- A display-handler layer to receive data from a data source and view it to the user in an appealing manner. Two different display-handlers might be implemented for web- and Android applications.

- A security-handler layer to verify user-input or file-uploads before it transfers it to the server.

- A server-handler to modify the database or to retrieve data from the database.

- etc.

6.2 Architectural Tactics or Strategies

For our system, which is to be used by researchers to track their publications, a number of quality requirements were described earlier. These are:

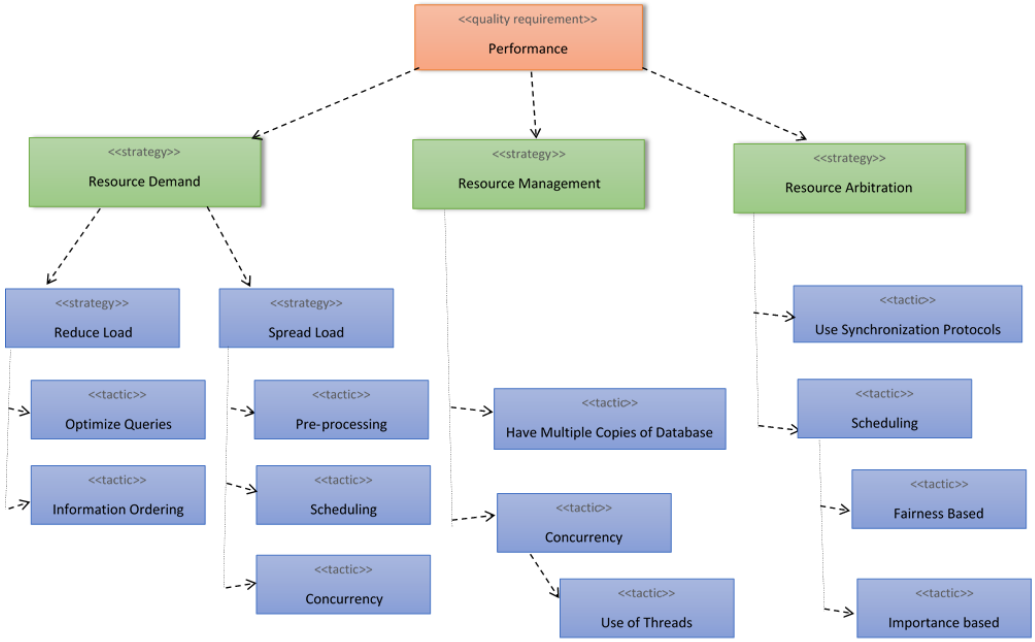
- Performance
- Reliability
- Scalability
- Usability
- Auditability
- Security

In this section, ways of achieving these quality requirements are discussed. This is done through architectural tactics and strategies. An architectural tactic is a design decision for realizing quality goals at the architectural level. Tactics influence the control of a quality attribute. On the other hand, an architectural strategy is a collection of tactics. For each quality requirement, the goals that need to be achieved will firstly be described, followed by a diagram that demonstrates the relationship between the quality requirement and the strategies and tactics chosen to achieve the described goals. Thereafter, the tactics and how they will be realized within our system are explained.

6.2.1 Performance

With performance, the goals that need to be achieved were described as having speedy data transfer that can only be slowed down by the speed of the network, concurrency so the system can handle all one hundred potential users at once and that the system should be efficient and not lag in information delivery.

Figure 38: Performance Tactics

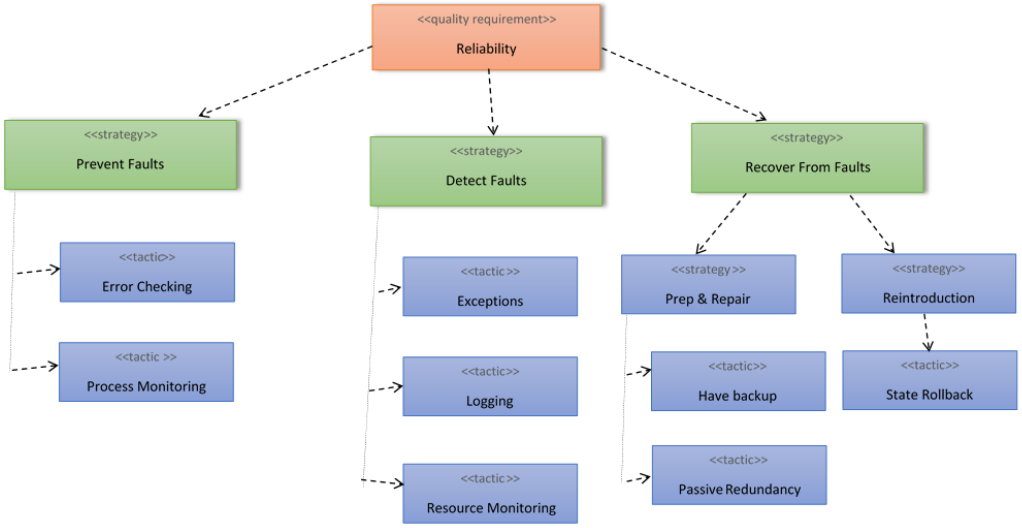


In order to address performance, our strategies focus on the management of resources, management of the demand on resources as well as dealing with resource arbitration. Our strategies achieve this through concurrency and scheduling tactics, mainly but also by reducing and spreading the load. The major resource in our system will be the databases which store information about users and publications. Through the tactics described we will able to control the use of these resources as well as avoid conflicting accesses. This is crucial as this could cause anomalies but, more importantly, slow down the system and give users a negative perception of the system if they have to wait to access resources.

6.2.2 Reliability

In terms of reliability, the goals that need to be achieved are an optimal time of ninety-nine percent and more up-time within a month as well as being able to ensure that system operations do not cause any system failures and errors. In addition, the system should avoid human errors.

Figure 39: Reliability Tactics

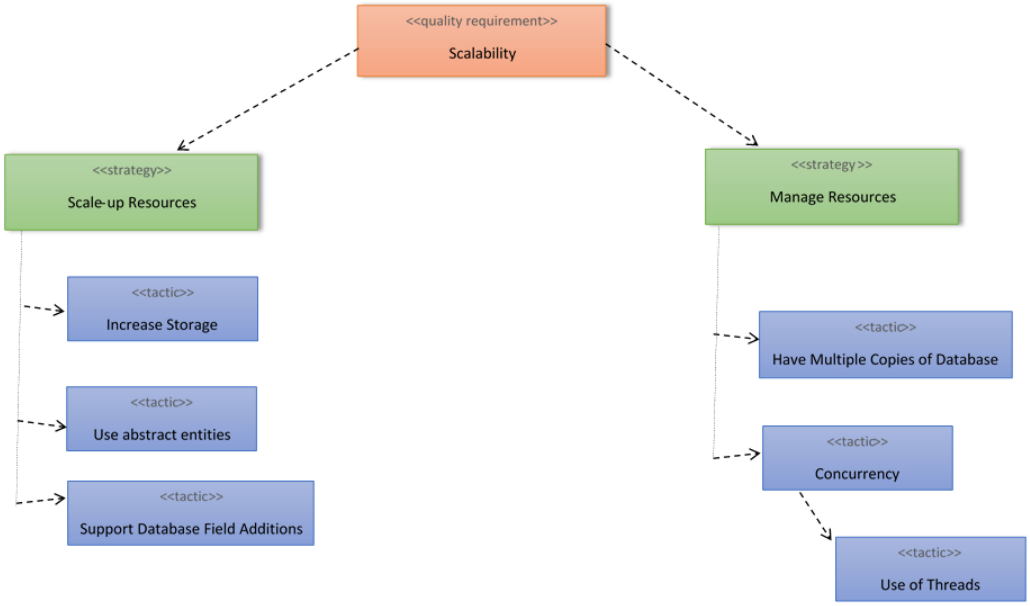


In order to maintain reliability of the system, the strategies we apply are there to prevent system failures, and in the rare case that they do occur, ensure they are detected and make appropriate preparations to recover and return to a working state of the system. In order to prevent faults in the system, tactics such as error checking and process monitoring are applied. This way we can avoid human errors and ensure all operations are running smoothly. To detect faults, we make use of logging and resource monitoring. This means making note of any changes or actions to the database and ensuring that it is always functional and that no potential faults exist in the databases. To recover from faults, one should always backup the system when it is in a working state, and use passive redundancy to inform system components when they need to make state updates. Thereafter, once a fault occurs, one can always roll back to the most recent working state.

6.2.3 Scalability

For scalability, the goals that need to be achieved include the system being expandable to other faculties at the University. With there being a total of 11 faculties, that means the system should be able to handle 1100 users concurrently without hassle and still meet the same performance standards as if there were only a 100 users.

Figure 40: Scalability Tactics

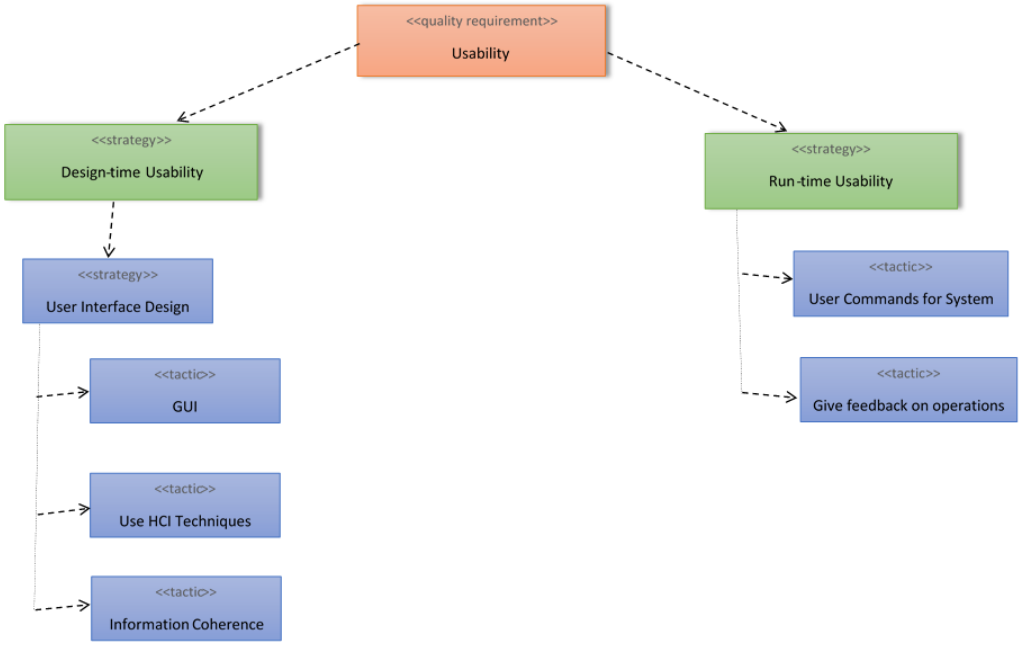


The strategies we use to address the scalability of our system to achieve the goal of it being expandable include the management of resources. This is necessary so that the resources can be managed to deal with the greater demand once the system is expanded to more faculties. In addition, the resources will need to be scaled up so that that the system can be applied in the different contexts of various faculties - hence tactics such as using abstract entities and the support of database field additions.

6.2.4 Usability

Goals that need to be achieved for the usability quality requirement are that the user should be able to perform system operations without hassle, the user interface should not be cluttered and information access by users needs to be simple.

Figure 41: Usability Tactics

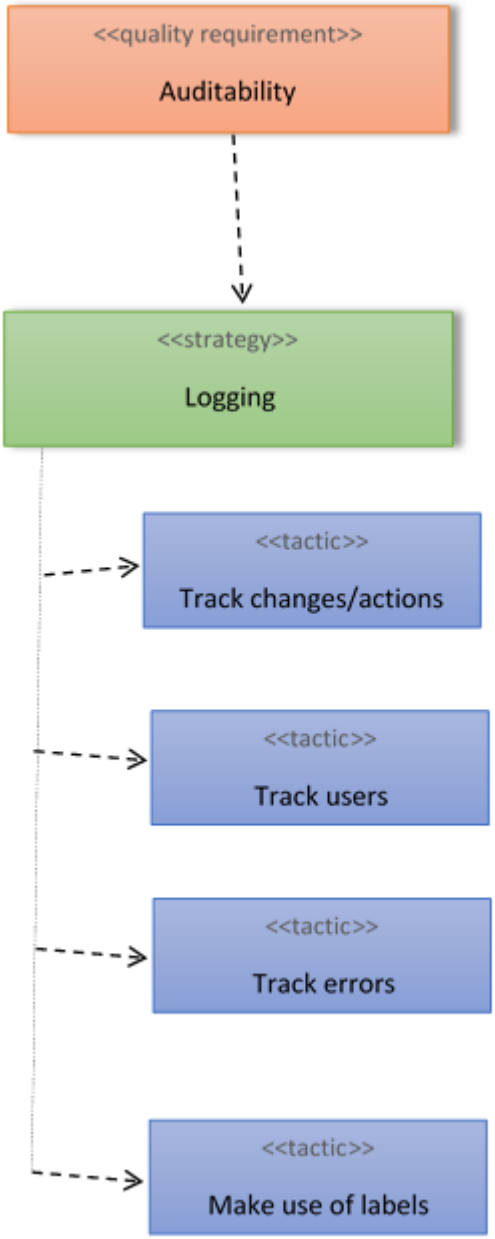


Our chosen usability tactics focus on making the system easier to use for users. The strategy of having a user interface is important and includes tactics such as using Human Computer Interaction (HCI) techniques as well as having a graphical user interface to make the user feel at ease with the system. In addition, information coherence as a tactic is important because it achieves the goal of information access and avoiding clutter and redundancy of information. Yet another strategy is having the system able to interact with the user at run-time with tactics like user commands and giving the user feedback on their operations. This makes the system more usable because it gives the user a semblance of control and keeps them informed.

6.2.5 Auditability

For auditability, goals that need to be achieved are the tracking of all information associated with the system including errors, changes made, when such occurred and who performed what actions - thus creating an audit trail for the system.

Figure 42: Auditability Tactics

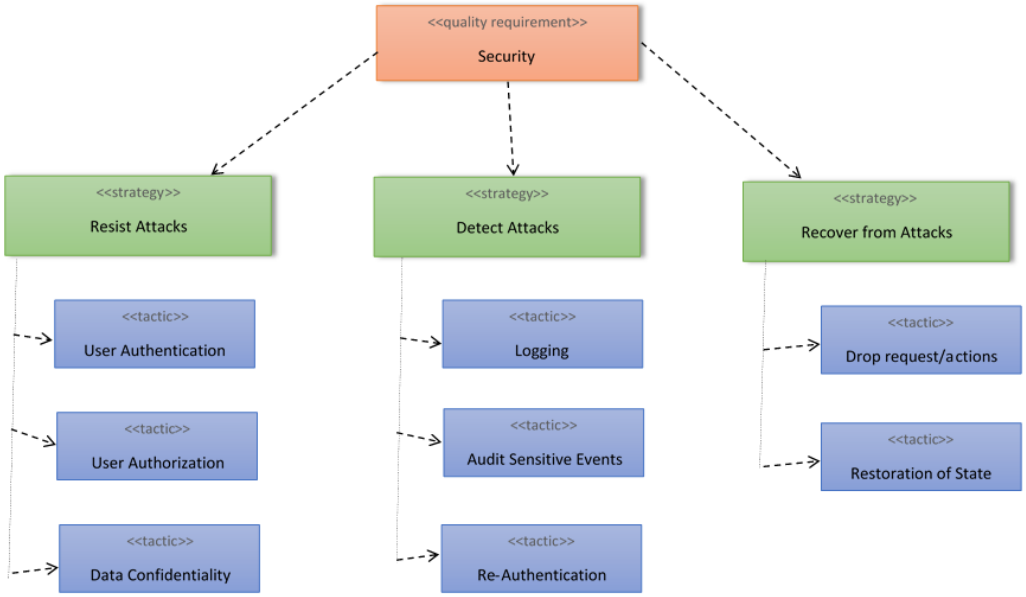


The strategy to deal with auditability is logging. We will make use of log files to implement tactics like tracking all actions/changes, tracking the users' activity and tracking errors as well making use of labels on logs that include time stamps, location of changes or errors as well as who made what changes and what caused errors. This all contributes to enabling system auditability, because everything is tracked and stored separate from the system databases.

6.2.6 Security

With Security, the goals that need to be achieved include a login system of which the user details will be verified and the prevention of unauthorized access.

Figure 43: Security Tactics



The strategies for addressing the security quality requirement include resisting attacks through tactics such as user authentication which addresses our goal of a login system, and user authorization which ensures users have the rights to perform certain actions and data confidentiality - which makes sure users only see the information they are allowed to see. Another strategy is the detection of attacks. This is done by requesting re-authentication if log in fails as well as the auditing of sensitive events such as making user removals and the like, and of course the logging of all actions so that inconsistencies can be traced. The final strategy - being recovery from attacks - includes actions such as dropping a request or action once it is found to be unsafe or to be a threat and once an attack is found, restoring the system to its state before an unsafe action was performed.

6.2.7 References

- Bachmann, F. Bass, L. Klein, M. 2003. *Deriving Architectural Tactics: A Step Toward Methodical Architectural Design.* `http://www.`

sei.cmu.edu/reports/03tr004.pdf [Accessed 8 March 2016]

- Kazman, R. Scott, J. 2009. *Realizing and Refining Architectural Tactics: Availability.* [Online]. Available: http://www.sei.cmu.edu/reports/09tr006.pdf [Accessed 8 March 2016]
- Petre, L. 2011. *Software Architectures, Lecture 2.* [Online]. Available: http://www.users.abo.fi/lpetre/SA11/SALecture2.pdf [Accessed 8 March 2016]
- Rehn, C. 2010. *Software Architectural Tactics and Patterns for Safety and Security.* [Online]. Available: http://www.christian-rehn.de/wp-content/uploads/downloads/2010/06/seminar_safe_sec.pdf [Accessed 8 March 2016]
- Solms, F. 2016. *Architectural Tactics.* [Online]. Available: http://www.cs.up.ac.za/courses/COS301 [Accessed 10 March 2016]

6.3 Use of Reference Architectures and Frameworks

The reason we are using reference architectures in this system is due to the fact that it helps with the development process of the system. What this section is meant to include is how the system should use frameworks to implement the architectural pattern(s) chosen and what the reasons for using these frameworks are.

6.3.1 Web 2.0 Reference Architecture

We have chosen to use and refer to the Web 2.0 reference architecture due to the fact that this is primarily a web-based system. The Web 2.0 Reference Architecture can be found here: http://archive.oreilly.com/pub/a/web2/excerpts/web2-architectures/chapter-5.html.

This Reference Architecture has 4 main tiers that are essential to our system, namely:

1. Resource Tier
2. Client Tier
3. Service Tierand
4. Design, Development and governance tools

Resource Tier :
The Resource tier includes 'backend systems that can support services that will be consumed over the internet'. Thus our system will need to use this tier to "consume" information which is input by the user for either: signing up, logging in adding a paper, changing the state of a paper, removing a user, adding a user etc. All this information will be updated in the database (also considered to be the model of the MVC pattern) which will store all the data of the user's and metadata of the paper's. This tier directly links with the Service tier due to the fact that for validation and storage of the input it needs to be sent to the server side and validated and stored from there.

Client Tier :
This component helps the user to use services and display graphical views. These are the web browsers used.

Server Tier :
This component connects the resource tier. Since they will be two versions of the application a web based and a android app, we will use the adapter architecture to read the data.

Design, Development and Governance Tools :
This tier refers to the tools that are used to actually build the web application. The reference architecture refers specifically to IDE's however I disagree with that in that one does not necessarily need an IDE to built a web application. This does not necessarily fit into this reference architecture directly. It is more just used to describe a tool for the developer to use and does not use the other tiers directly.

6.3.2 Django

Django is another framework will be used. The reason for this is that it follows the MVC architecture design pattern .Django's primary goal is to make it easy to create data driven websites that are complex. It comes with caching to make processing of data that is frequently uses faster. Django being a server side helps in keeping information safe. In addition Django is very fast.

6.3.3 Bootstrap

We will use bootstrap cs to style the web interface of System .Advantages of using bootstrap is that is customizable, you pick and choose the feature

you want then download the files. Another advantage of bootstrap is that is responsive. This will help other mobile devices such as IOS and blackberry which aren't running on the specified android app that will be designed. Bootstrap enhances working together as a group because it brings consistency regardless of who did the work and it stays the same across all browsers. Bootstrap is consistently updated so it stays in synced with new technologies. It also has a large support community in case you run into problems.

6.3.4 Android SDK

Android SDK will be used to develop the android app, this is specifically developed for android so t makes the functionality of the app faster than an app developed using hybrid frameworks.(such as ionic).

6.3.5 AngularJS

Another framework to be used is AngularJS. AngularJS enables massive parallel development. It also reduces code length, making code easier to maintain. Loops that could have been long are reduced to a single line. It allows a developer to deal with modules on the client side as they would deal with them on the server side.

6.3.6 jQuery Mobile Framework

jQuery Mobile is a multi-platform mobile 'touch-optimized web framework'. This can be used to develop the Android application for the system. This framework allows a developer to design one application that will be able to run on all smartphone and tablet OS's, allowing for easy portability, if needed at a later stage. To develop an application using this framework, developers need to use web-based languages such as: jQuery, HTML, CSS. It does allow for client-server communication (which is what is needed for this system). This framework is based on web development but was written specifically for smartphone and tablet applications and websites. jQuery mobile uses device-level optimizations to make sure that the layout is appropriate for the specific device it is running on. This allows for easier development of a device agnostic application. This framework is also extensible. It allows for many different third party plugins and is integrable with other frameworks such as AngularJS, Backbone.js and mobile frameworks (to name a few). This creates an opportunity to easily integrate frameworks which can implement the MVC architectural pattern, which is what is being used to develop this system.

6.3.7 Backbone.js

This framework would be used together with the Android application to provide the missing JavaScript architecture which jQuery Mobile cannot provide. jQuery allows the implementation of multiple patterns with a Model and View which is referred to MV*. Thus the implementation of the MVC architectural pattern is possible, which is the pattern that will be used in this system. Backbone helps to structure the application (which could be complex) while still keeping it easily maintainable and fast. Backbone is lightweight and performance-wise it is very good due to the fact that it prefers to integrate with other frameworks to create a User Interface rather than it having something built into it for this feature, thus reducing overhead. The reason that Backbone would be used with jQuery Mobile is the fact that Backbone is not used to create a UI (i.e. the View is basically just a template). Backbone may not necessarily implement a strict class system but it does use prototype-based inheritance as well as object extensions. This makes it possible to implement architectural and design patterns which use Object-Oriented programming.

Jackbone.js: Since the integration of jQuery Mobile and Backbone is common they decided to create a framework that integrates the two and makes it easier to develop lightweight, extensible and fast mobile applications. It also makes it easier to implement MV* patterns such as MVC. Where Backbone did not necessarily have a Controller, Jackbone has extended the Backbone framework to implement a controller in the MVC pattern. It relies mostly on Backbone.js and provides specialised classes for view and router.

6.4 Access and Integration Channels

In this section we mention and discuss the main access and integration channels of this system. Among them are different human users, each acting in different roles, and system users - such as remote clients or the database software.

6.4.1 Access Channels

- **Human Users:** Human users will have access to the platform through two separate channels. Namely, through a website interface and a native Android application.
  - **Administrator:** Two administrator users will have access to the system. Firstly the Head of Department will have an administra-

tor account through which he will have access to all features, and all data, in the system. He will, therefore, be able to access and update information on papers without having to be an author on the paper. He will also have access to the database on which all system data (including users, papers and venues) is stored. This will require that such access is available on both the web interface and the Android app. Additionally there will be another administrator account through which a technician must access the system in order to conduct routine maintenance and fix possible errors.

  - **User:** User accounts will be allowed to add papers to the system as well as edit papers which they are already assigned to (either as Authors or not). Users do not have administrative access to any data (apart from Authors and Venues) which they have not been assigned to. For instance, a User may not view or edit a paper which they have not been assigned to by another User, already assigned to said paper.
  - **Author:** Authors have no access to this system. Authors are simply data entries in the database which have been stored to simplify data entry in cases where an author needs to be assigned to several papers.

- **System Users:** The system will be sharing information to the two main portals - being web and Android. In order to accomplish this we will be implementing an API (Application Program Interface) which will act as a layer of abstraction. It will provide access to requested information (users, papers, venues, searches, etc.) and carry out tasks on behalf of the portal (be it web or Android). By doing this we will be advancing flexibility of the system, since the same API calls will be used regardless of the platform used to make them. The API will function by receiving a call, delegating the call to the server and then forwarding the response to the client (who made the initial request).

  Furthermore the database will be accessed via MongoDB's built-in API and called from the server - where each call from the client is handled. In other words, the server is accessed by a client interface and the server software, in turn, accesses the database.

- **Protocols:** Here we discuss some of the main protocols we plan to use in implementing this system. This includes protocols for transferring miscellaneous data (such as status codes or images) as well as data such as emails or hypertext files.

- **TCP/IP (Transmission Control Protocol/Internet Protocol):** TCP/IP will be used to establish connections between computers such as the server and the clients. TCP allows us to open a data stream to which we can read and write. This allows us to transfer data over networks, including the Internet. We chose TCP over alternatives such as UDP due to its inherent reliability.
- **HTTP (HyperText Transfer Protocol):** HTTP will be used to initiate transfers between components of the system. GET and POST requests will be used to transfer data to and from the server, database and the numerous client interfaces. Additionally, HTTP provides us with built-in error codes which will allow the system to track the success or failure of a request and react accordingly (retrying or requesting new input data from user).
- **SMTP (Simple Mail Transfer Protocols):** SMTP will be used to send emails to users, notifying them of oncoming deadlines for paper submission. SMTP provides a stable method to transfer email by letting us transfer the email to a SMTP server and then queueing the mail until the next server - either an intermediate SMTP or POP3/IMAP server - becomes available. By using SMTP we will minimize the chance of mail getting lost and a user not being notified.
- **FTP (File Transfer Protocol):** FTP will be used to transfer files (such as HTML, CSS or PHP files) to and from the server. It will specifically be used in transferring files to the webserver from remote locations.

6.4.2 Integration Channels

The system will integrate with an external database system (which has been chosen as MongoDB in this case) in order to store all relevant publication and user information.

In addition, the API will communicate with the client and server using AngularJS, as described in the Frameworks section.

6.5 Technologies

6.5.1 Programming languages

- Interface

  - HTML: The system will be mainly web based, and will need to be integrated with Android for mobile deployment. Thus the interface should be represented with HTML5, as it multiplatform and scalable with future technologies.
  - CSS and Bootstrap: Styling the web based interface should be handled through CSS3 and the Bootstrap framework. Bootstrap allows for a responsive, mobile interface with a professional appearance. Importantly its current popularity allows for future upgrades and scalable interface design.
  - Javascript, AngularJS and Node.js : The interface needs to remain robust, intuitive and responsive. To achieve a responsive environment, AngularJS will be used. Node.js is designed to build scalable network applications and using these frameworks, our system will remain modularised and will allow us to implement dependency injection.

- Server The server side processing needs to be handled by a suitable scripting language, however the debate between the flexible Python and the standard PHP can be extensive. Below is a solution.
  - PHP: Commonly installed environment, embedding in HTML, large user base.
  - Python : More secure than PHP, flexible with extensive add-on modules, tends to lead to much more scalable application
  - Solution: WPHP is a WSGI-¿PHP gateway that "allows you to run PHP processes inside of Python, using a WSGI gateway". In summary using both PHP and Python to achieve best of both options.

- Application platform
  - Java: Promotes modularisation, integrates with the other technologies.
  - Android SDK: The mobile application will be Android exclusive.

6.5.2 Framework

Besides the frameworks mentioned above the following will also be used.

- Apache Cordova: As the system will need to be mobile device deployable, the popular mobile application development framework PhoneGap or more specifically the open source version Apache Cordova will be used to wrap the web application into an Android application without having to remake the system in anything other than HTML, Javascript and CSS.
- ASP.NET MVC: The system will follow an MVC architecture and the free, fully supported framework for building web applications ASP.NET MVC will help facilitate this design.
- MEAN : A JavaScript software stack that makes use of MongoDB, Express.js, Angular.js, and Node.js to create robust, maintainable and efficient applications.

6.5.3 Database Systems

- MongoDB: A cross-platform document-oriented, NoSQL database. Using JSON-like documents with dynamic schemas, it will be able to integrate with the system easier and faster. More importantly, MongoDB allows scalability for our database system. Further simplifying the system it uses Node.js as a platform and is incorporated in the MEAN stack.

6.5.4 Operating Systems

- Desktop: Windows XP service pack 3, Vista,Windows 7, Windows 8 and Windows 10. Linux Ubuntu, ArchLinux, Fedora and all debian based systems. MAC OS X. All major web browsers will be supported.
- Mobile device: Android 2.2 Froyo to Android 6.0 Marshmallow and all future releases.

6.6 Server technologies

- Express.js: As an alternative to XAMMP, Express.js is the standard server framework for node.js which we are already using and is part of the MEAN stack. Express will be set up as our server to host the system.

6.7 Other technologies

– GitHub: Will be used as our version control system and be used to host our system source code. Thus concurrent work can take place, maintenance does not have to disrupt the system and mistakes can be undone.

7 Open Issues

- Should the admin be able to edit the meta-data of the papers or not?
  - Should the research leader be able to edit the meta-data of papers in their research group?
- What server will be used to host our site?
- Should we add any backup capabilities for the database?
  - If so, will the admin be the only one able to backup the database?
- Is there a specific database that we need to use to store the information (i.e. MySQL, MongoDB, Neo4j etc.)?
- Should the system be able to support more than 100 users if, for example, another employee was employed by the Computer Science Department? (i.e. should the addition of users past 100 be allowed?)
- Should any user be able to add papers on behalf of other users? There may be risks involved when allowing this.
- If a new user is added and they are already an author of a paper (when they were not a user of the system) should the papers, which the new user is part of, be displayed on their profile?
  - Will this new user receive units for the paper if it has already been published?
- If a paper has been "terminated" should the metadata still be editable while it is in a terminated state (besides for changing the state of the paper)?