Architectural Requirements Notes

P.A.P.E.R.S (Publication And Papers Electronic Repository System)

University of Pretoria - Team Charlie

Notes By:

Claudio Da Silva - 14205892
Dillon Heins - 14035538

GitHub Repository - Team Charlie

For more information, please click here

5 March 2016

Contents

1 Architecture requirements **1**
1.1 Architectural scope 1
1.2 Quality requirements 3
1.3 Integration and access channel requirements 5
1.4 Architectural constraints 6

1 Architecture requirements

1.1 Architectural scope

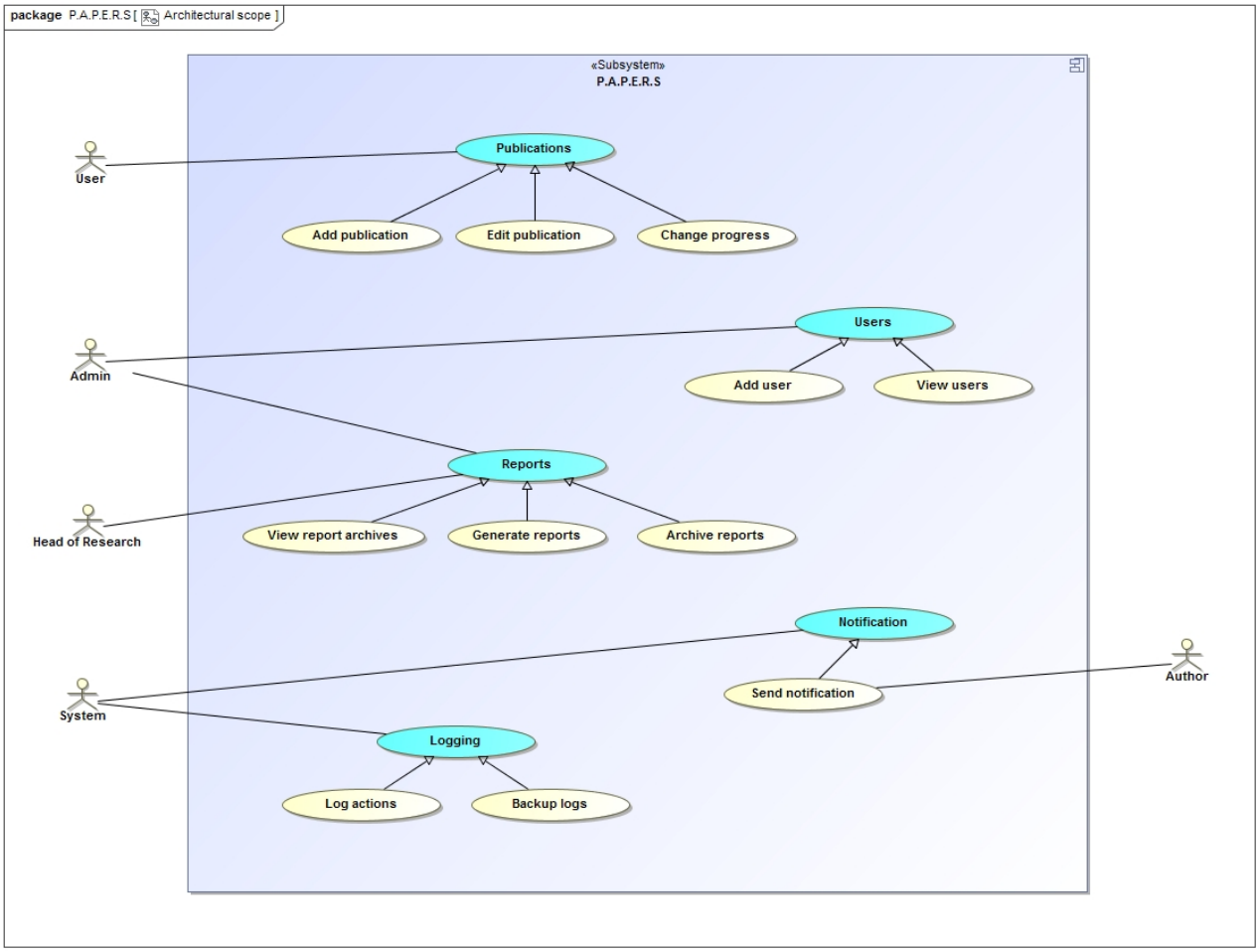

Figure 1: Architectural Scope - Use case for Architectural Scope (Further refined in use cases)

The system will be used to keep track of publications made for various research groups in the department, as well as keep track of the authors and the users in charge of things. From this we may later then generate reports on how various groups are doing as well as the department as a whole, and notify authors and users about upcoming deadlines and events.

In the event of dealing with queries on database items such as users and publications, for example insertion and deletion, as well as viewing, the system must first validate that those items do or do not exist in said database. Once that is verified, the queries should be run and the result returned. On success the user should be notified, on failure an appropriate exception should be thrown. Persistence must be maintained throughout the database with very little fault tolerance.

In dealing with reports, data being queried should be validated and the system should ensure that all queried data is recent and correct, halting any new additions during reporting. Reports should be properly archived for

later retrieval and properly backed up regularly.

In dealing with notifications, the system should ensure that all emails being used for every author does in fact exist and is a valid email. When sending this email, likely from the mail server at UP, the SMTP connection used must be ensured to be secured and the email should be sent through valid channels as to not appear as spam mail to other users. The system should also validate that it is using the latest deadline updates at all times.

1.2 Quality requirements

- Performance
  - Workload is a maximum of 100 users concurrently.
  - No implementation of concurrent editing of document entries - last saved edit is written to the database.
  - The system should be able to support 100 users updating information at the same time as updating is more intensive than reading.
  - The response time of the system should be fast enough that a user is able to complete their work without frustration. Due to the system being off-line it is reasonable to expect the system's response time to be only limited by the speed of the network.
- Reliability
  - The system should not fail whilst providing critical or important use cases.
  - The system should handle all requests and respond with appropriate result objects for each.
- Scalability
  - Ability for multiple external systems to connect to the system's API.
  - The system should be able to support a large amount of historical document entries being added to the database.
- Security
  - A hierarchical system will be used to determine the security privileges of users of the system.
  - Passwords are to be hashed using at least sha256 and should be stored as such within the database along with a salt.
  - An inactive user session should be terminated after a period of 10 minutes with no activity.
  - A user who has forgotten their passwords can use a password reset option which will send a one time password to their registered email address so that they may login once using it and reset their password.
- Flexibility
  - The client has stated that the system is not needed to be able to extend to accommodate a greater number of departments.
- Maintainability
  - The system should have as few bugs as possible so as to prevent having to constantly maintain it in the future.
  - The system should be built in a modular way so that all services are decoupled in such a manner that allows for the extension of the system at a later stage.
- Auditability/monitorability
  - Every action performed by a user should be logged and all details about said action should be stored.
  - These actions should be visible to admin users.
- Integrability
  - User's document entries should not be able to be deleted, if it is a case where the document will not be completed it should remain in the system and be terminated.

  - A user with no admin rights should not have access to admin privileges so that the system's data may remain integrable and safe.
  - The system should not ever be in a state where it is under pressure and the data is at risk of becoming corrupted. The system should be designed to handle the pressure for which it has been specified to handle.

- Cost
  - All software used should not be proprietary but rather open source so as to minimise cost as much as possible.

- Usability
  - The interface should be lightweight.
  - The interface should be intuitive to use as well as obey Human Computer Interaction guidelines so that it is efficient and easy to use.

1.3 Integration and access channel requirements

- The system must host a server to which API calls may be performed, to achieve this we use:
  - Django (A Python based web framework allowing easy implementation of the MVC principal.)
  - AngularJS (For easy unit testing functionality and dependency injection)
- The system should make API calls via browser, the browser functionality and calling is done via:
  - HTML5 (To ensure access to the latest and most stable versions of features)
  - Javascript with a JQuery framework (For the functionality)
- The system will require and android interface to interact with the API. To achieve this we make use of:
  - JAVA (The standard language of every android application.)
  - JQuery for mobile (A convenient method to manipulate data via JQuery on your android app.)
  - Sencha touch (An HTML5 framework for Android.)
- The system should be able to interact with the systems database, which will be hosted using:
  - PostgreSQL (An open source relational database that integrates perfectly with the Django framework.)
- The system should be able to send emails on a scheduled basis by sending mail from the CS mail server, for this we use:
  - Django-sitemessage (A framework developed to allow easy scheduling of emails as well as the ability to retry an email on a failure.)
- Protocols to be used include:
  - HTTP
  - SMTP
  - TCP

1.4 Architectural constraints

- The system should be developed using only non-proprietary technologies.
- The system is to be developed for Linux based systems (Mostly Arch Linux) used by the Computer Science department, however preferably the system should run on any general use operating system.
- The system should be browser independent, thus the system should be accessed via any browser without issue, including but not limited to (Firefox, Chrome, Safari, Edge etc.)
- The system should be handled locally and should not rely on outside internet sources in order to function.
- The API created must be queried by both a web browser, as well as an Android application.
- The system created should allow for users to use the application via mobile, employing technologies such as bootstrap in order to keep it mobile friendly. This will always allow users of iOS and Windows mobile to access the system via channels such as Safari and Edge.
- The system is required to be concurrent and make use of concurrent methods specifically in it's functionality, allowing up to a minimum of a hundred concurrent users to be active at the same time