Rares-Sebastian Moldovan
30433

Students' Algorithm teaching tool
Supplementary Specification

Version 1.0

Revision History

Date	Version	Description	Author
19/03/2018	1.0	First version of the supplementary specification documentation.	Rares-Sebastian Moldovan

Table of Contents

Supplementary Specification

1. Introduction

The Supplementary Specification captures the system requirements that are not present in the use case model. Such requirements belong to the non-functional domain.

Examples of such attributes are: quality attributes (availability, performance, security, testability and usability), operating systems (regarding the portability of the system), environments (hardware and software dependency of the application), compatibility (with other types of operating systems) and design constraints (design decisions that have been mandated and must be adhered to).

Another addressed problem is the application standard (standard performance requirements such as response time to external events and time needed to access the database for basic operations such as: insert, delete, select or update).

The following sections address these problems in an organized manner, starting with Section 2 that comprises the non-functional requirements of the system and its subsections for each of the quality attributes). Section 3 addresses the problem of design constraints (such as architecture, design patterns and technology, purchased components, class libraries etc.).

2. Non-functional Requirements

This section treats the main non-functional requirements of the system and how they are incorporated in the system, as well as what is the standard for each one of them and how it may or may not be accomplished upon delivering the final version of the product.

2.1 Availability

Availability is the ratio of time a system or component is functional to the total time it is required or expected to function.
The application must have high availability since the users may need to acquire information from it at any time (since during a semester, the algorithms may be parts of the disciplines weekly).

2.2 Performance

Performance is the amount of work accomplished by the application and can be expressed using many other notions such as: response time (needs to be short), utilization of computer resources (how much memory is needed to store intermediate results of the variables).
Performance can be measured, informally, by choosing one of the most complex use case (such as report generation) in the following way:

Source of stimulus: the administrator
Stimulus: selecting the "Generate report" button
Environment: the administrator is logged in and the system is in a stable state (no users are currently rating)
Artifact: the report generator
Response: the system provides the desired report.
Response measure: time passes since the report request and the actual delivery of the report, also measured as a function of the input data (the number of users registered in the application).
Tactic: try to optimize the computations by ignoring the 0-rated users and order only the ones which have contributed in one way or another to the algorithm pages.

2.3 Security

Software security is the idea of engineering software so that it continues to function correctly under malicious attack.
The security measure of such an application would be the ease of users to log into accounts that do not

belong to them or to alter pages of the application without being in the right position (such as a regular user trying to modify an algorithm page without asking for permission from the administrator of the page).
The security of this application can be approached classically. Therefore, the accounts are password protected and in order for a user to change the password account, it must request the administrator to do so since he/she is the most reliable.
However, the user may change its credentials also via email verification since users do not generally accept for the administrators to know their password information.
Regarding the malicious attempts to alter pages, the application will not allow any kind of modifications to be performed in the database unless the user is an administrator. In the case of algorithm administrators, they will be allowed to alter only the corresponding section of the database.

2.4 Testability

Software testability is the degree to which a software artifact (i.e. a software system, software module, requirements- or design document) supports testing in a given test context. If the testability of the software artifact is high, then finding faults in the system (if it has any) by means of testing is easier.
The testability of this application needs to be as good as possible, therefore a modular approach will be used in development such that the application will allow tests ranging from unit testing up to integration or even acceptance testing.
Both black-box and white-box testing must be performed to guarantee the correctness of the functional requirements as well as how much the system conforms to the non-functional requirements (such as maximum response time and computation time for

2.5 Usability

Usability is the ease of use and learnability of a human-made object such as a tool or device. In software engineering, usability is the degree to which a software can be used by specified consumers to achieve quantified objectives with effectiveness, efficiency, and satisfaction in a quantified context of use.
The usability of this tool is obvious from the way it is organized. Since every algorithm has its own page and the page aspires to be a common place for students to help each other, it is as usable as they want it to be. The quality and relevance of the information is not the job of the system, but rather the job of its administrators.
Regarding the interface with the user, the system aspires to be as easy as possible, providing classical components such as buttons, text fields and labels for the user to fill in the desired information. From the graphical user interface point of view, the system will be as usable as possible.

3. Design Constraints

The main design constraint of the application is the architectural pattern: Client-server architecture.
The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. The server will deal with the main operations on the database and will retrieve the information to the requesters.

The software language is Java and its libraries will be used to access the database and construct the web application. The application will also use Hibernate. The IDE can be either IntelJ or Eclipse since they both allow the development of Java applications in an environment where compilation and debugging is more facile.

Some of the classical Java libraries will be employed, such as libraries from java.util for processing data in an organized format (making use of lists: ArrayList, sets Set, HashSet, treesMapTreeand other data structures that make data accessing and modification easier and more organized).