Thesis

DTU Compute
Department of Applied Mathematics and Computer Science

Collaborative Requirements Engineering with RED Cloud

Ioannis Oikonomidis (s083928)

Kongens Lyngby 2015

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

Requirements engineering projects are inherently cooperative, requiring many stakeholders to coordinate their efforts to create a requirement specification. Conventional tools for file sharing and version tracking are not optimized for such projects. Version control has always been a tricky task, and even many computer science students have serious troubles with the concepts involved. So how can we ask domain experts to be proficient users of a VCS?

The goal of this thesis is to create a tool for requirements engineers where they can share, structure and keep track of their models in a collaborative environment. The evolution of a project should be visualized using activity diagrams allowing team members to plan and document their workflow. Since REDCloud functions as an extension to the Requirements Editor (RED) software tool that is used at the course 02264 - Requirements Engineering taught at DTU, an interface allowing the connection between these two should be designed.

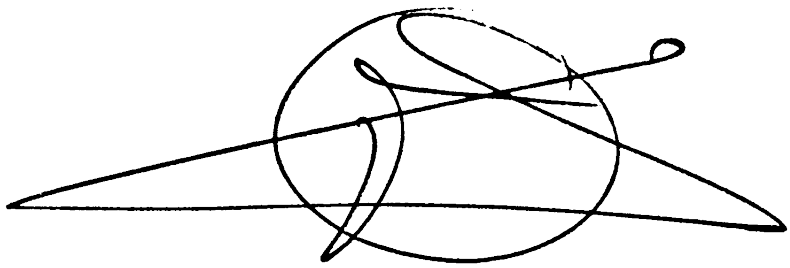
The platform should also provide superior usability and stability since it is intended to be used in the classroom. The development of this tool shall continue after this particular project has ended and this requires an extensible and robust software architecture.

Preface

This thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements for acquiring a master degree in Software Engineering.

The 32.5 ECTS worth project started on the 27th of January 2015 and finished on the 11th of July 2015. The supervisor of the project was Prof. Dr. Harald Störrle.

Kongens Lyngby, July 11, 2015



Ioannis Oikonomidis (s083928)

Acknowledgements

I would like to thank my supervisor Prof. Dr. **Harald Störrle** for showing a great interest in the project and for always taking time to supervise me. Without you this project would have never come to an end. I hope my work will bring much value to you and future students of your course.

A special thanks to my friends and family who supported me in writing and incented me to strive towards my goal, during a period of political and historical changes for my home country. At the end I would like to express appreciation to **Antigoni Kavoura** who spent sleepless nights and was always my support in the moments when there was no one to answer my queries.

Contents

List of Figures

CHAPTER 1

Introduction

By definition, requirements management is a collaborative task. Users, stakeholders, analysts, engineers, product managers, QAs, etc. they all must work together with the requirements. Every change, every modification must be documented, tracked and analysed by several roles. This thesis, describes the work of designing and implementing a tool that optimizes collaboration between team members of a *Requirements Engineering (RE)* project.

1.1 Motivation

The tool is mainly built to allow collaboration between *course participants* of the course 02264-Requirements Engineering, taught by the *teacher*, Harald Störrle, in *Denamrks Technical University (DTU)*. During the course, students form groups and work on case studies applying methods they learn about the theory and practices used in modern requirements engineering. The client software they use is called *RED* and it provides a tool support for creating a requirements specification. *Teaching assistants (TAs)* supervise course participants throughout the course. RED originates with 02264, but has the ambition to go beyond and reach out to *industrial users*.

The main issues that course participants are facing during the course, are mostly collaborative. While they work in groups, RED does not support collaboration between team members fact that forces students to experiment with different kinds of *file sharing* tools, including Dropbox, Git, SVN and similar alternatives. Furthermore, students communicate with their supervisors, the teacher and the teaching assistants mainly using emails and this is also how they receive feedback on their deliverables. It is trivial to understand the need for the development of a tool that enhances collaboration between students and their supervisors and in the same time gives a clear insight of student projects evolution.

1.2 Approach

To capture the evolution of RE projects, activity diagrams are used. In its basic form, an activity diagram is a simple and intuitive illustration of what happens in a workflow, what activities can be done in parallel, and whether there are alternative paths through the workflow. In REDCloud, evolution of student projects are considered to be business use cases and are captured through activity diagrams organized in vertical swim lanes.

For allowing RED clients to communicate and collaborate with each other, a client-server approach is followed where RED is the client and *REDCloud*, the application built in this thesis, is the server. RED and REDCloud should communicate over a computer network on seperate hardware via the Internet and for this reason RED-Cloud should provide an *application programming interface (API)*. Since REDCloud is a web server it should also handle browser clients and not be strictly bounded to RED clients. This gives more flexibility and adds use cases. This thesis mainly focuses on finalizing the backend and browser frontend of this approach while the API design is considered as a feature for a future release.

1.3 A Real Life Scenario

In order to properly document the use of the collaboration tool and identify its main features, we consider a real-life scenario from the course 02264-Requirements Engineering. This, functions as a base for this project and is later on referenced, allowing to focus only to what is important. The *Library Management System (LMS)* case study is part of the course material and is used as a running example in the lectures. The case study is developed by Harald Störrle and is close to a complete requirements specification document. The scenario described below can be extended to fit exactly the LMS case study but it is intentionally assumed that the students workload only consists of two deliverables, which are sufficient to identify key-features that should be considered.

1.3.1 Group Registration

During the last minutes of the first lecture, course participants are advised to form groups in order to work on the LMS case study. *Alice (A)*, *Bob (B)*, *Charlie (C)* and *David (D)* decide to form a group. The group has to define a *leader*, lets say C, who is responsible for communicating the project status to the teacher and the TA. They sign a paper and pass it to the TA, who then has to register the project teams manually on *campusnet*, DTU's student portal.

1.3.2 Workflow Planning & Feedback

In their first group meeting, A, B, C and D, plan the project deliverables and share tasks between them. Until now, this is mostly done using different methods and tools, including pen and paper, since RED, the client software they use, does not provide such a feature. The final version of the task distribution and the plan is shared between course participants, the teacher and the TA via campusnet. Feedback is sent to the students via email.

1.3.3 Workflow Documenting & File Sharing

A, B, C and D apply the feedback they received and they start the hard work. After discussing the LMS vision and its stakeholders, each student is assigned the task to define and document three stakeholders. B is then supposed to collect the stakeholders and **deliver** the merged document as a RED project via campusnet. A is later assigned the task to define the system context while C has to draw the physical system structure diagram and pass it to D so that he can draw the internal physical system structure diagram. B is responsible for merging and **delivering** the system context, the physical system structure and the personas via campusnet.

1.3.4 Reusable Components

Goals for the LMS requirements specification are discussed and defined in the course. The teacher uploads a **.red** file on campusnet, containing the top level goals of the LMS system and group leaders are asked to download and integrated it to their project.

1.3.5 Group Evaluation

The teacher has to evaluate course participants and for this reason he has to get into *who did what* sections of each final group report.

CHAPTER 2

Analysis

In this chapter, a stakeholders analysis is conducted and the goals of this thesis are summarized. The system context together with the domain architecture are then discussed. The last part describes the system entities and its physical architecture. This gives the reader a better understanding of how REDCloud and its components work.

2.1 Stakeholders

2.1.1 Course Participants

Dropbox is used as the main file sharing solution between course participants and this comes with the drawback that if a file is open by more than one person at the same time, conflicts may occur. While Dropbox is fairly good at resolving these conflicts by saving multiple copies of the file and designating which copy belongs to which computer, merging the changes made to both files can often be tedious and annoying. Extra time is also spent on setting up the required Dropbox accounts, as well as sending and accepting invitations for shared folders. Conventional *version control systems (VCS)* (SVN, Git etc.) is an alternative solution that allow students not only to share files but also to keep track of the versions of their project. It is often time-consuming though for students to surpass the learning curve of these tools. Computer science background is not a prerequisite of the course and for this reason course participants are not assumed to be familiar with with VCS. However, they are expected to have experience working with computers and use their preferable operating system. Furthermore, RED does not allow *workflow planning*, such as TODO lists or task assignments and prioritization. As a result, students have to find their own custom way for distributing tasks and collaborate effectively with each other.

2.1.2 Teacher

The teacher is the main responsible for the course and hence he is considered to be a major stakeholder for this project. His vision for the final tool is a service able to capture the evolution of *small-scale*[1] student projects using activity diagrams. He

[1] several (5 – 20) large (MB) files, 3-6 group members

recognized the need for a tool to support collaboration between future course participants and has several wishes for the project. It is not trivial for him to track who did what or when with the current assessment method used in the course. Additionally, RED does not support team synchronization and as a consequence there are cases of course participants being blocked waiting for each other for deliverables. Finally, the tool should offer high maintainability, since it is not supposed to be a production-ready product and it is expected to be extended by other students in the near future.

2.1.3 Teaching Assistants (TAs)

TAs are individuals, often past course participants, with instructional and administrative responsibilities. Right now, it is hard for them to offer remote assistance since sharing **.red** files via emails is not an optimized communication process. Therefore, there is a need for them to be able to get a fast overview of a RED project workflow, comment on it and modify its plan. Lastly, the tool should provide an administration interface that enables the creation of users and their linking to projects.

2.1.4 Industrial Users

RED is to be released as an open source project in the near future. To support this goal, increase the public interest and get more people involved, it is vital to offer unique or at least similar features comparing to tools already available on the market. There are tones of solutions already allowing team collaboration, brainstorming, discussions, task assignment and time tracking. *Trello*, *PivotalTracker*, *Rally* and *Breeze* are just a few examples. However, in most cases, they target a rather extended audience, as per example employees with different background working on different business functions of a company. This is happening mostly due to the fact that big companies prefer all-in-one comprehensive solutions rather than selecting different business processes depending on the size or the kind of a project. There are also some commercial tools focusing more on requirements lifecycle management, such as *Visure*, *Polarion* and *Blueprint*, which consider collaboration between users. Some of the aforementioned tools only visualize the evolution of a project from a user point of view using horizontal swim lanes (Figure A.1(a)), activity calendar lists (Figure A.1(b)) or lists of tasks (Figure A.1(c)), while others only provide visualization from an activity status perspective (Figure A.1(d)). At the time of writing, there are no alternative commercial or open source solutions providing user oriented visualization of small-scale, requirements engineering projects using activity diagrams. Thus, implementing this system can be seen as a major contribution to this clientèle.

2.2 Goals

Based on the real life scenario described in Section 1.3, the derived goals for this thesis are displayed in Figure 2.1.

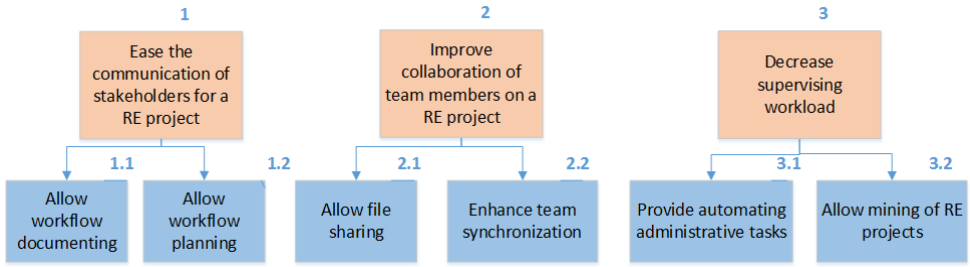


Figure 2.1: System Goals

1. The communication of stakeholders for a RE project needs to be improved. As discussed earlier in Section 1.3.2, course participants use third party tools to plan their workflow and they communicate with their *supervisors* - the TA and the teacher - via emails. Furthermore, workflow documenting should be provided, in order to ease project status tracking.

2. In Section 1.3.3, Bob needs to merge different parts of the LMS requirements specification before he delivers. This requires A, C and D to get synchronized with each other. Sharing and merging files has to be optimized since deliverables consist of one **.red** file and that makes conventional tools like dropbox hard to use.

3. TAs have to manually register approximately 15 groups of 4 members each on campusnet, as discussed in Section 1.3.1. For this reason, projects and team members registration should be automated. Last, evaluation of course participants, as described in Section 1.3.5, can be a painful process and the teacher should be able to mine RE projects and track the progress of the students at any time during the semester. That would allow him to give direct feedback on time.

2.3 System Context

The system context diagram is displayed in Figure 2.2. Course participants need to share information with each other while their supervisors should be able to manage, study and mine student projects. For this reason, a client-server model is used. Users can access the system using any modern browser, fact that ensures platform independence. Browser clients are used for communicating workflows and projects across stakeholders while the RED clients are used for creating and viewing models. Thus, all stakeholders are able to interact with the tool, regardless their level of familiarity with RED. In a later stage, an interface to RED clients should be provided.

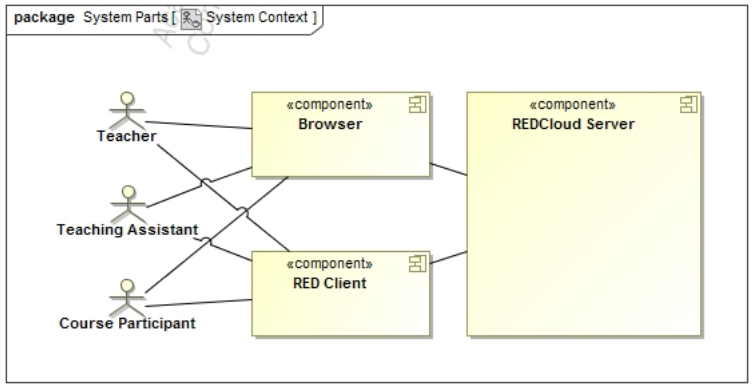


Figure 2.2: System Context Diagram

2.4 Domain Architecture

To get a broad view of REDCloud and understand how it works we need to split it in components.

For the purpose of stimulating RED or other projects, the system should provide a *Project* module allowing file system similar operations.

Activity diagrams are to be used enabling file sharing, team collaboration and synchronization as well as workflow planning. Thus, REDCloud should provider a *Graph Editor* module where users can create, read, update and delete (*CRUD*) different types of activity diagram elements and perform operations on them, using for example context menus.

As in most web applications, a customized experience that grants specific permissions to different users should be provided. For example, a course participant should not be able to edit projects that does not contribute on. This creates a need for an *Authentication* module.

Last, an *Administration* module would help with any administrative tasks such as creating projects and assigning permissions to users. Figure 2.3 displays REDCloud's domain model architecture diagram, in which components marked with blue represent modules that are most of the times already provided by modern web frameworks.

2.5 Analysis Level Information Model

An overview of the REDCloud metamodel is displayed in Figure 2.4.

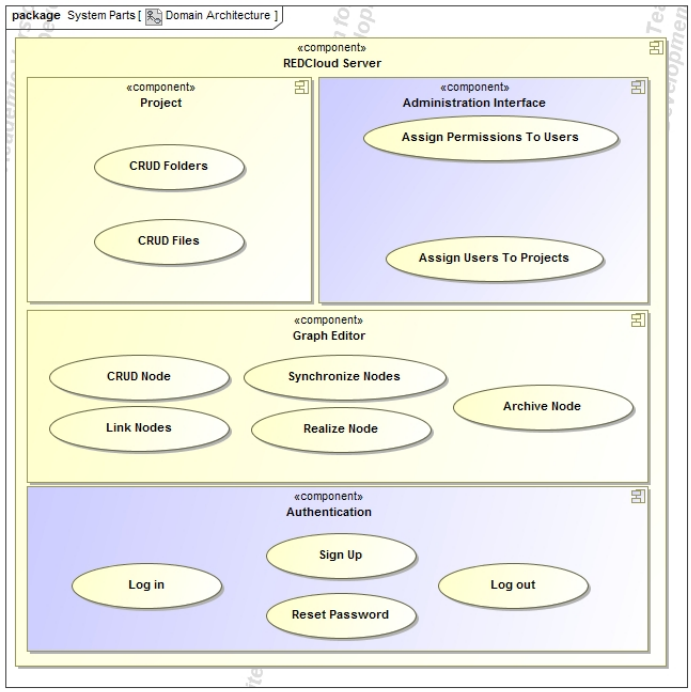


Figure 2.3: REDCloud Server Domain Model Architecture Diagram

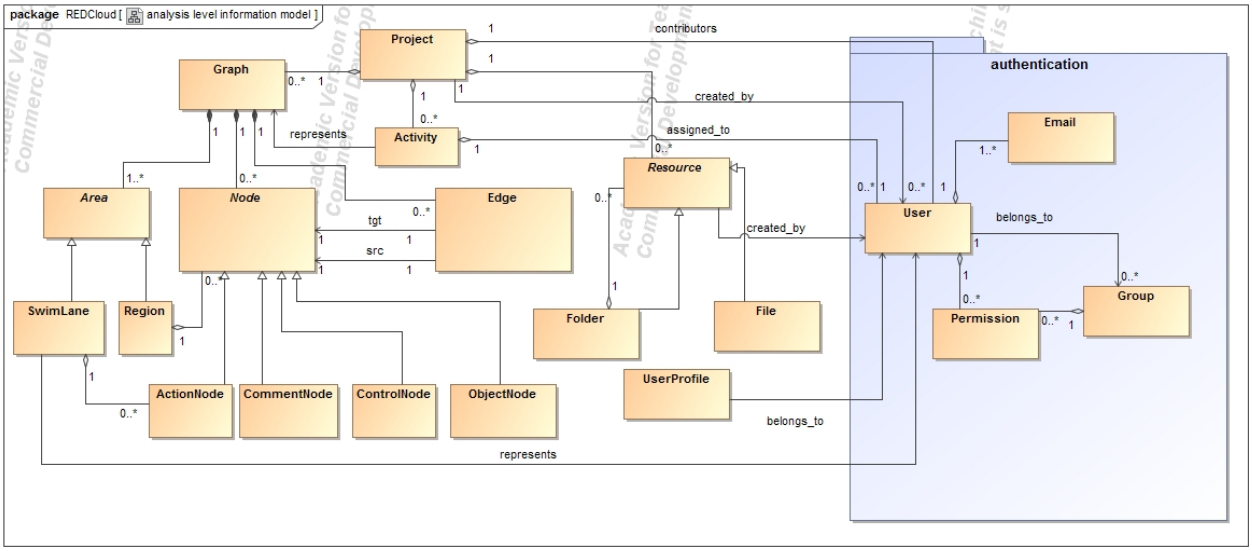


Figure 2.4: Analysis Level Information Model

To better understand how complex REDCloud is we need first to discuss its different entities.

2.5.1 Project

The project module metamodel is displayed on Figure 2.5. Defining a project as a collection of *folders* and *files*, helps us maintain a high level of abstraction and simplify things. Files and folders are following the *composite design pattern*; they are both *resources* and folders may reference multiple resources. A project has *contributors*, *leaders* and *supervisors* all together forming the *team* of a project. *Activities* are defined as a set of actions and deliverables or else as a business process and thus they reference a *graph*. Activities have due dates and are assigned to specific users.

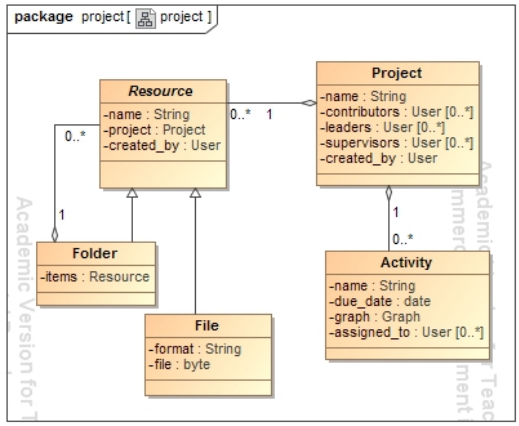


Figure 2.5: Project Module Metamodel

2.5.2 Graph

A graph represents a business process and consists of *nodes*, *edges* and *areas*. The graph metamodel is displayed in Figure 2.6. The different types of nodes that are used in REDCloud are listed below.

- *Object* nodes used to define object flow in an activity and may hold a resource reference. In this way file sharing can be achieved.
- *Action* nodes that represent project actions in a workflow.
- *Control* nodes used to route control and data through the graph. Specifically, the control nodes used in this scope are *start* nodes, *flow final* nodes, *join/fork* nodes and *choice* nodes.

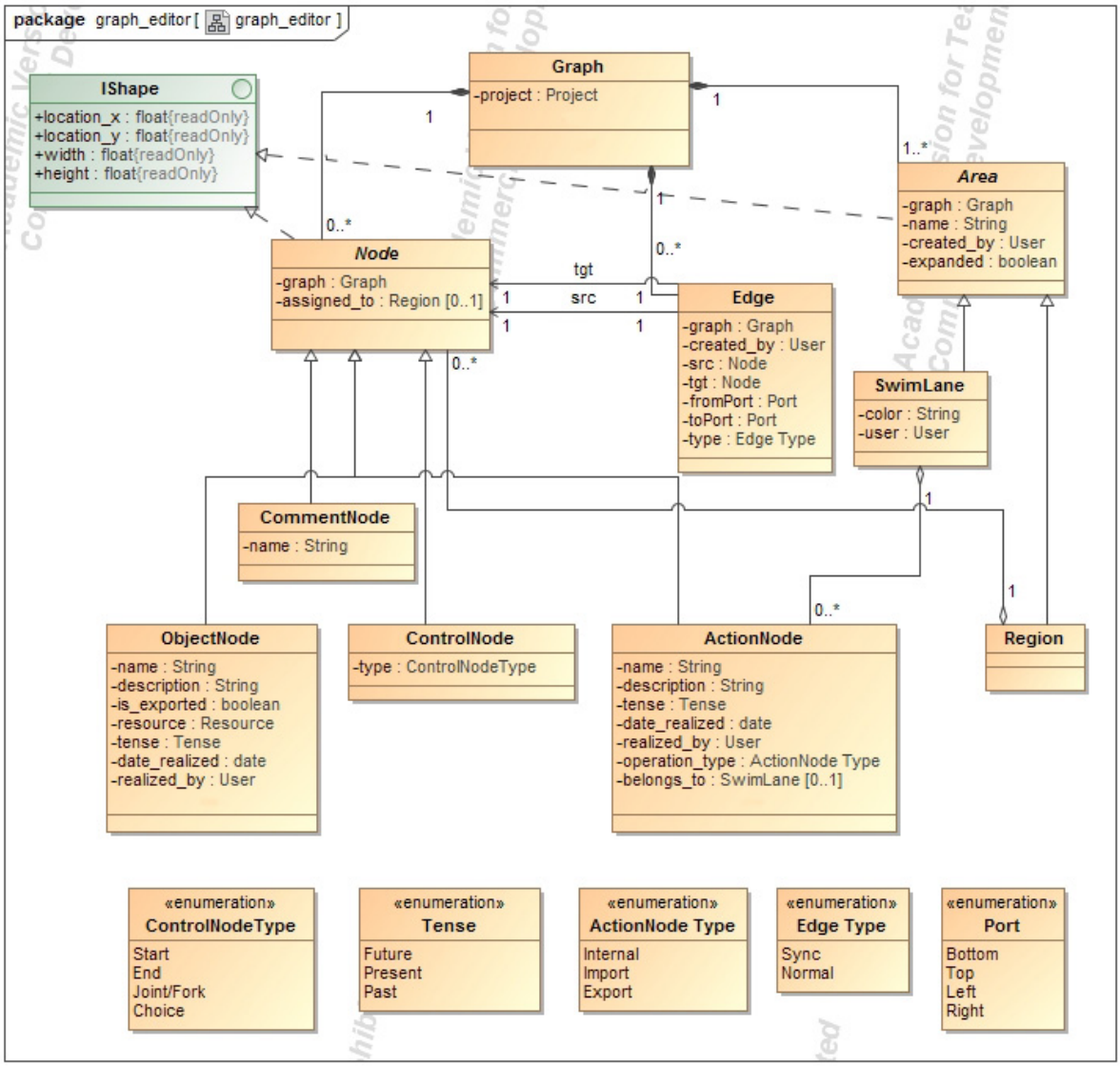


Figure 2.6: Graph Editor Module Metamodel

- *Comment* nodes are used for feedback or other comments. The communication between team members of a project or the teacher can be replaced by comment nodes instead of emails.

In order to achieve workflow planning, as discussed in 2.2, *Tenses* are introduced: *Future*, *Present* and *Past*. Initially all action and object nodes are future tensed. Future tensed nodes can be deleted in order to allow plan modifications. Future tense nodes can turn to present tense when someone *realizes* them. Present tense nodes represent activities and deliverables that are agreed to stay on the workflow and that someone is currently working on them. Thus, they can not be deleted. Nodes can also get *archived*. This action sets the tense of the node to past. Past tense nodes can not be modified.

Edges are arrows connecting two nodes. Edge can either be of type *Sync* or *Normal*. If an edge is of type sync, then it is used to connect two synchronized nodes while when it is of type normal it represents that control passes to the node the arrow points to.

Areas can either be *swim lanes* or *regions*. Swim lanes represent team members of a project and they contain action nodes while regions may group collections of nodes and edges.

Graph elements are visualized later in Section 3.4.2

2.5.3 Authentication

Users should have one or more *email* addresses and may belong to one or more *groups*. Groups and users should have specific *permissions*. The authentication metamodel is shown in Figure 2.7.

2.6 Physical System Architecture

As discussed in Section 1.2, connection of RED clients and browsers is achieved using a web server that is connected to the internet. In order to store and load data, the application should be tied to a database. To reduce the amount of time it takes for the host server to process a request and avoid repeating unnecessary complicated queries to the database, a key-value cache should be used. The physical components of the system that is to be developed are displayed in Figure 2.8.

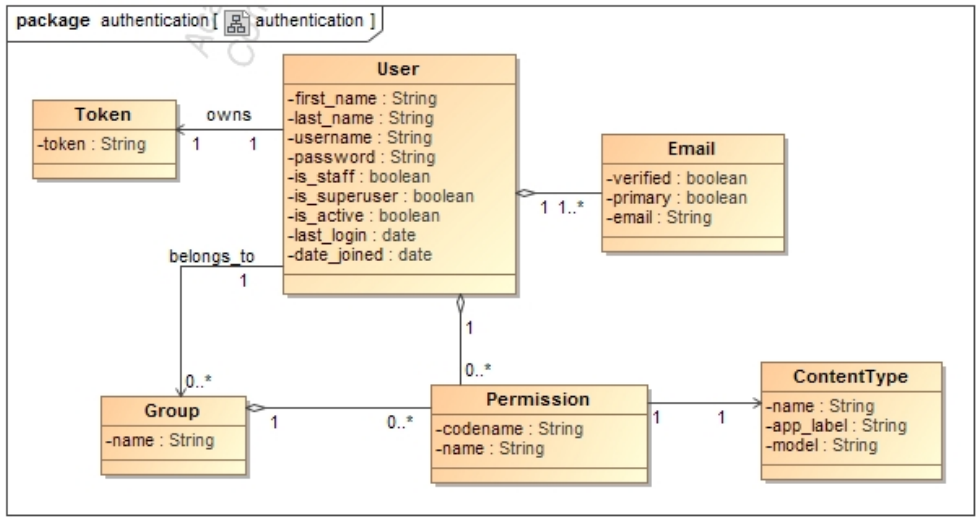


Figure 2.7: Authentication Module Metamodel

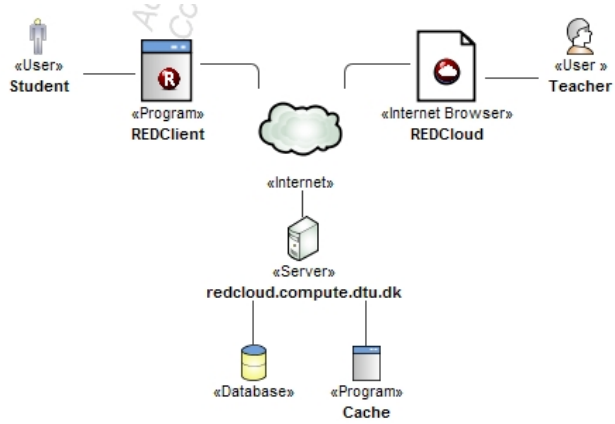


Figure 2.8: Physical System Architecture Diagram

CHAPTER 3

Design

In this chapter design choices made are described. Considering the real life scenario from Section 1.3 and the analysis made in Chapter 2, the main screens of the application are then sketched.

3.1 Applying Agile Principles

Agile and *waterfall* are two distinct software development methods. The waterfall model can essentially be described as a linear model of software design and it suggests a sequential design process. Development flows sequentially into several different stages: *requirement analysis*, *system design*, *implementation*, *testing*, *deployment*, *maintenance*. It requires upfront, extensive planning while feedback and testing are deferred until very late into the project. While waterfall processes are plan oriented and tend to be more secure they are incredibly rigid and inflexible. Modifying the design of a project at any stage, can be a nightmare and once a stage has been completed, making changes to it is nearly impossible.

In contrast, the agile method proposes an incremental and iterative approach to software design. Developed in response to the limitations of waterfall, as a way to give software designers mode flexibility, it promotes adaptive planning and evolutionary development. *Agile Software Engineering* [OH08] points out that communication is a central theme in software development. Indeed, the success or failure of software projects is sometimes attributed to people communication issues. In situations where the end-goals of projects are not initially clearly defined, agile can be especially beneficial.

A key feature of agile approaches is the underlying assumption that a project consists exclusively of a sequence of iterations, possibly with the exception of a very brief *vision and planning* phase prior to development, and a similarly brief *closure* phase after it. The fixed length of iterations gives teams a simple way to obtain, based on velocity and the amount of work remaining, a usually accurate (though not very precise) estimation of the project's remaining duration.

In this project, retrospective sessions were integrated on a weekly or sometimes bi-weekly basis into iteration summary meetings. That helped the author deeper understand the requirements and the vision of the teacher. Furthermore additional goals and features were set as the project evolved (i.e. workflow planning (May 2015) and template importing (June 2015)). In every meeting, the progress of the last iteration was discussed and valuable feedback was collected. The output of the

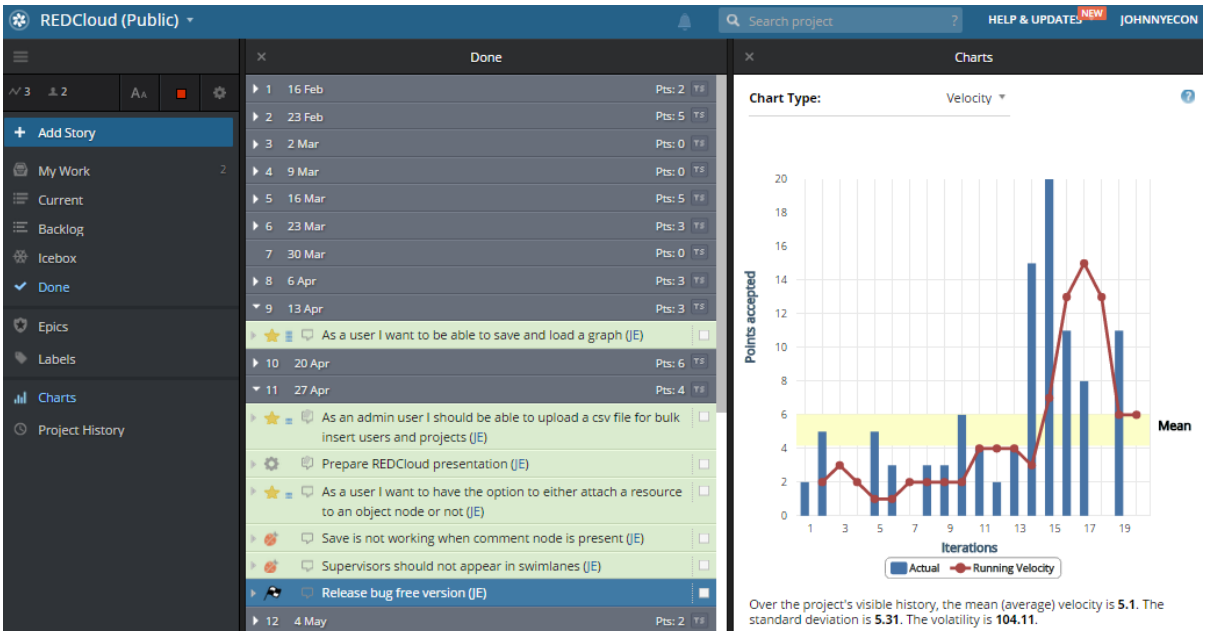


Figure 3.1: REDCloud on Pivotal Tracker

discussion was written down to user stories using pivotal tracker. Figure 3.1 displays stories listed in different iterations on the left and the velocity graph on the right. Pivotal tracker is a story-based project planning tool from *Pivotal Labs*. It is based on agile software methods, but can be used on a wide range of projects. Additionally, it integrates with bitbucket, where the project repository is hosted. That ensures that all actions taken from the beginning of the development phase are well described and documented and thus, maintainability is enhanced. User stories are later transformed in behavioural tests when possible. Behavioural tests are further discussed later in Section 5.1.

3.2 Design Level Information Model

Changes made in the design level information model are shown in Figure 3.2. As displayed in the metamodel in Figure 2.4 earlier, a project may have different graphs and activities, while an activity represents a graph. According to the *OOA96 Report*, [SS96], when abstracting a relationship that closes a loop of relationships, one needs to verify that the relationship is required and that it captures an association not already captured by other relationships in the loop. In the REDCloud context, activities are there for the users to import when working on a project using the graph editor. To simplify things, activities are not implemented. Instead a boolean

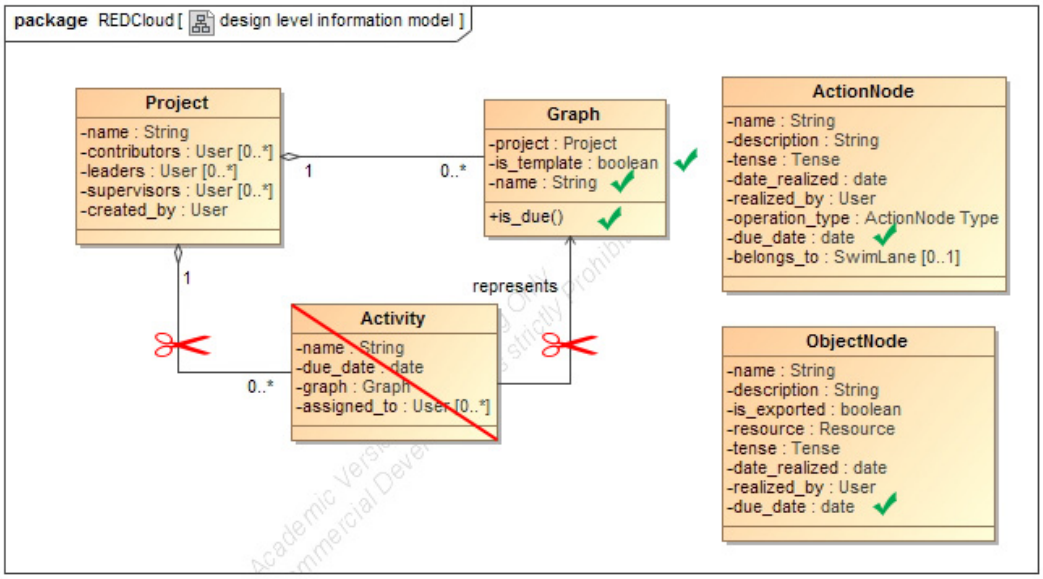


Figure 3.2: Changes in the Design Level Information Model

attribute `is_template` on the graph entity is introduced and the optional attribute `due_date` is added on nodes that can be realized (action nodes and object nodes). The `activity assigned_to users` relation, can be derived by the swim lanes in which different nodes of the graph belong. A graph provides an `is_due` operation which can be calculated by comparing all its unrealized action and object nodes `due_date` with the current date.

3.3 A Transactional Approach

It is clear from the Analysis Chapter 2 that system users need to *save* and *load* graphs using the graph editor. In this section considerations and approaches on designing the save/load functionalities are discussed.

Save and load can be considered as write and read transactions to a shared graph resource. Transaction technology is originated from databases and has been widely used for a multitude of CAD applications. Transaction is defined as a unit of work wherein several operations can be treated as a single logical unit of work performed. There exist various transaction models in the literature. The classical model of transaction, as defined in [MY03], is based on the criteria called ACID, an acronym that represents the properties of *atomicity*, *consistency*, *isolation*, and *durability*. Transactions characterized by these properties are the most tractable and powerful construct for managing concurrency, allowing multiple clients to simultaneously read and write

data without fear of conflicts. For this reason the selected database management system should provide support for reliable transactions.

Saving the state of the graph on the database can be achieved either asynchronously or synchronously. In the first case, whenever changes are detected to the graph state, a save event is triggered without the user taking a special write action. To ensure atomicity and isolation, write transactions lock the database so that the state of a graph can not be changed by many users concurrently. According to *Justin Callison* [Cal13], auto saving would increase the risk for *lock contention* and *long term blocking* during peak times - for example during the course - where multiple logged in users may move around nodes in the graph concurrently. For this reason a transactional approach is preferred, where users have to take action in order to save and load the project.

Lets consider the scenario below:

- Alice and Bob, contributors to the same project, log in and navigate to the graph editor.
- Bob changes the name of a node from `foo` to `bar`.
- Alice deletes the same node that Bob has edited.
- Alice saves.
- Bob saves.

Here, Alice may or may not get notified for the change done. Since the assumed client is the browser, the obvious ways to get Alice notified about the change done, would be the use of either *AJAX polling* or *web sockets* technology. The two protocols are visualized in Figures A.2 and A.3 respectively in Appendix. Polling creates additional network traffic and web sockets are considered to be a better, but certainly a more complicated approach. Since web sockets require a lot of developing effort to implement, it is decided to exclude real time notifications for the initial prototype without excluding it as an option for future work. Thus, the server stack should support the WebSockets protocol and instead, a static html notification next to the swim lanes could indicate online users working concurrently on the same graph.

When Bob saves, he would get surprised realizing that the node he just edited is gone. This is intentionally not handled as users are suppose to only work on nodes that belong to their corresponding swim lane.

3.4 Achieving The Goals

As discussed in Section 2.5, the graph editor module consists of nodes, edges and areas. Users can interact with graph elements by right clicking on them and then selecting a menu item from the context menu that show up. Figure 3.3 presents the different types of nodes that are used in the graph editor.

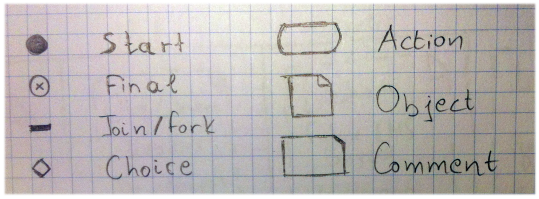


Figure 3.3: Graph Editor Nodes

3.4.1 Workflow Planning

Workflow planning is achieved using tenses. As discussed in Section 2.5.2, tenses only apply to action nodes and object nodes and they are tied to the node's color and its stroke line style, as shown in Figure 3.4.

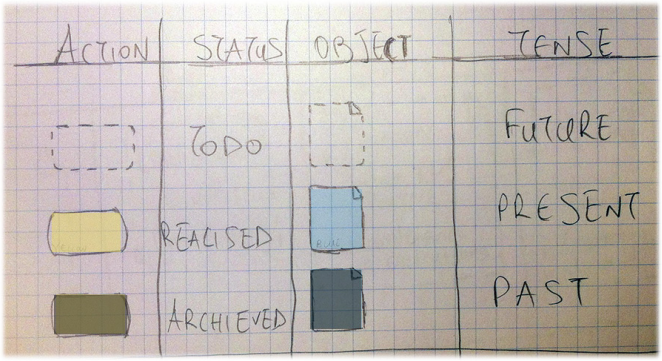


Figure 3.4: Tenses

3.4.2 Workflow Documenting

The use of activity diagrams itself, enhances workflow documenting, since they illustrate what happens in a workflow, who did what and in which order. Additionally, description fields in node creation and editing forms allow users to document in details their actions. Metadata, like creation dates, realization dates etc. are displayed on mouse over using infoboxes, as displayed in Figure 3.7 later.

3.4.3 File Sharing

File sharing between the stakeholders of an RE project is achieved through file uploading and downloading using object nodes. Object node context menus include a download option, while object node creation and editing forms include the option to upload a file attaching it to the selected object node.

Reusable components are another way of sharing files between users of the system. As discussed in 1.3.4, the teacher may want to share information with all the course participants and for this reason exporting and importing object nodes and sub-graphs should be considered. When an object node is exported, all users of the system should be able to import it in their project. The export option is included in the object node's context menu while the import option in the swim lane's one.

3.4.4 Team Synchronization

In order to enhance team synchronization, the sync edge type is introduced. This special link, connects nodes that are to be delivered - realized - on parallel. In other words, when the time comes to realize a node, a user is prompted to realize all its synchronized nodes. Synchronization can be achieved via the context menu of a selection of unrealized nodes. Figure 3.5 visualizes the two different edge types.

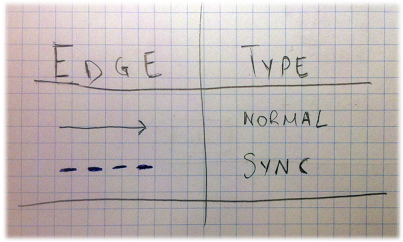


Figure 3.5: Edges

3.4.5 Automating Administrative Tasks

An administration interface, where TAs and the teacher can manage and configure the system, is designed. The interface provides a list of all the projects of the system and their members. Administrators should be able to perform CRUD operations. *Batch creating* projects and project team members should also be feasible. A sketch of the project administration area is designed in Figure 3.6.

Another contribution to this goal, is the ability for the students to import graph templates already prepared by their supervisors. In this way, workflow plans can be imported and there is no need to communicate plans between the team members. The boolean `is_template` setting can be configured in the graph administration area.

3.4.6 RE Project Mining

To provide an insight into practices and procedures followed during various phases of a requirements engineering project life-cycle an activity diagram is not enough. Considering the limited screen size, swim lanes can get very large and workflows can be too complicate to read and analyse for big RE projects like the LMS case study. For this reason an overview pane, displaying a zoomed out version of the activity

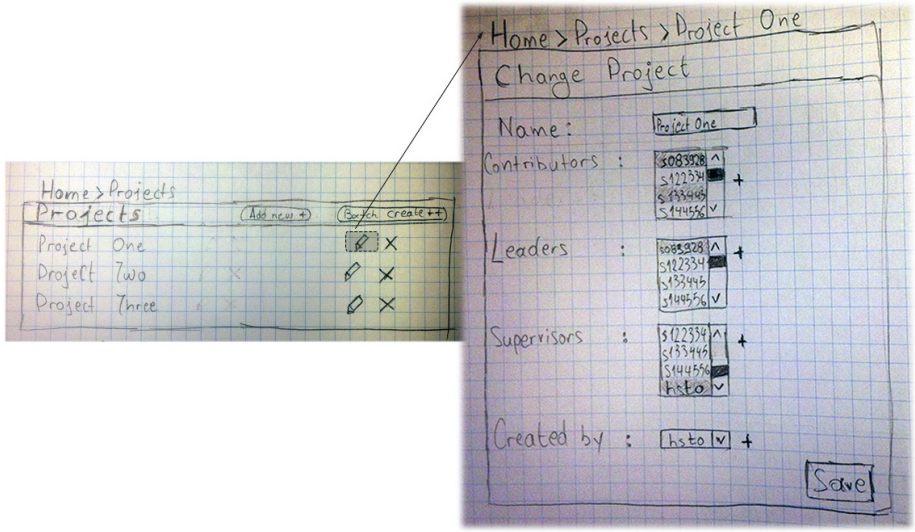


Figure 3.6: Project Administration Interface

diagram, is designed. This area should be mouse wheel zoomable and draggable in order to ease navigation. Infoboxes discussed in Section 3.4.2 also contribute to this goal. These two features are displayed in Figure 3.7.

Furthermore, in order for the teacher to easily track due activities, the ability to generate custom reports through the administration interface should be considered.

3.5 Context Menus

The graph editor's context menus are summarized below

- **Object Node** - An object node should be able to get archived (when not past tensed), deleted (when future tensed), edited (when not past tensed), exported and realized (when future tensed). When resources are attached to it, the user should also have the option to download them.
- **Action Node** - Similarly, action node context menus should allow users to archive (when not past tensed), edit (when not past tensed), delete (when future tensed) and realize (when future tensed).
- **Comment Node** - Users should be able to edit and delete comment nodes.
- **Swim lane** - Swim lane context menus should allow the user to create an action, an object or a comment node, to import an exported object and to select all its

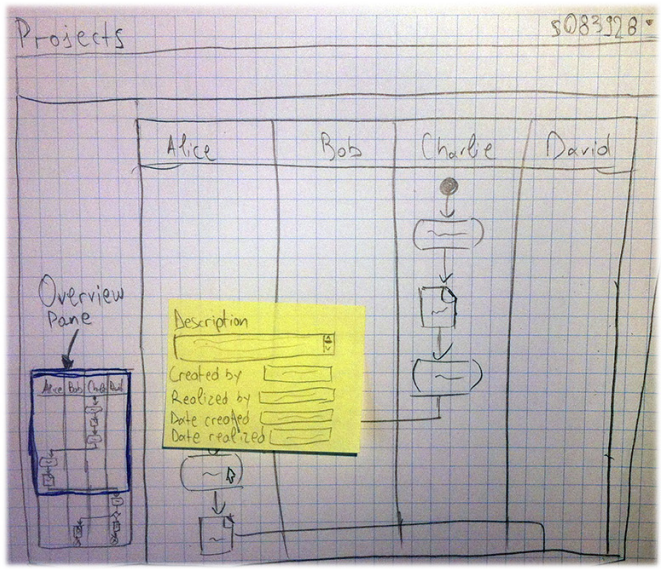


Figure 3.7: Overview Pane & Infoboxes

nodes. Swim lanes visually distinguish job sharing and responsibilities and for this reason users should also be able to edit their color.

- **Selection of Object and or Action Nodes** - Multiple action and object nodes should be able to get synchronized.

- **Selection of Object Nodes** - To ease file sharing between the system's users, multiple object nodes should be able to get exported. The exported object node should refer to a folder with all the resources of the selected object nodes. The operation should mirror a new folder in the server containing a copy of all the exported resources.

3.6 Applying The LMS Case Study

An example of how the use case described in 1.3.3 could be visualized in REDCloud is displayed in Figure 3.8.

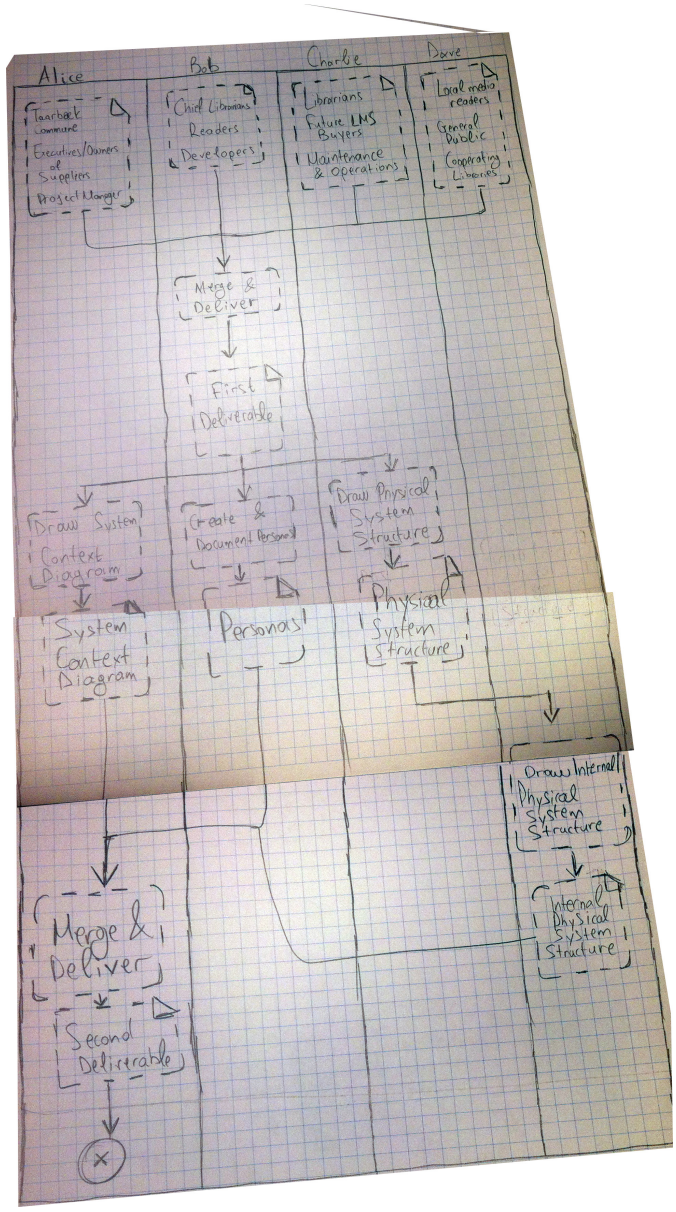


Figure 3.8: REDCloud Activity Diagram Example

CHAPTER 4

Implementation

4.1 JS Graph Visualization Library

In order to visualize the evolution of a project in a browser client a JavaScript library satisfying the requirements set in the Design Chapter 3 should be chosen. An open source library could be extended but that would require a lot of time and effort to develop the basics. Putting it all together we need a graph editor where users can create, read, update and delete predefined graph elements. Swim lanes and custom actions, such as context menus, should be supported. The library should also be able to run on all modern browsers. Taking into consideration the limited amount of time in which this thesis should be delivered and also the ability to get an academic license while using commercial software, the choice got narrowed down to two libraries: *jGraph* and *GoJS*.

4.1.1 jGraph

It seems that jGraph is the market leading JavaScript graph visualization component. It is being actively developed since 2005 and it provides a consistent and complete API that allows a relatively easy creation of a graph editor.

jGraph's main product, draw.io, focuses on manually creating and editing a graph. It took something less than two days in order to set up a prototype rendering a flow chart with 2 vertically oriented swim lanes. The effort spent to load this very simple initial graph was way bigger than when using GoJS. Furthermore, trying to remove what was unnecessary and limit the user options would require reading a huge user guide and getting deep into the jGraph API. The xml file that corresponds to a graph built using jGraph, was a lot more complex than the json GoJS one. After spending a few days trying to solve some auto layout problems the decision to drop jGraph as an option for the application stack was made.

As mentioned in jGraph official website [LTD15c], *"the current strategy shift to web applications is only intensifying"*. This makes jGraph an attractive alternative to extend RED Cloud with, in case more advanced visualization features are needed in the future, such as exporting directly to dropbox, one drive or google drive and making use of guides and validation on drawing.

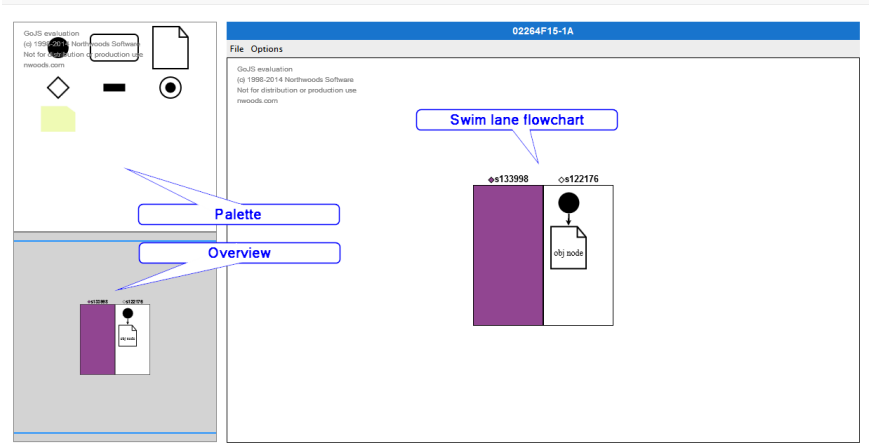


Figure 4.1: Flowchart with Swim Lanes using GoJS

4.1.2 GoJS

GoJS is a feature-rich JavaScript library for implementing interactive diagrams across modern browsers and platforms. It makes constructing diagrams of complex nodes, links and groups easy with customizable templates and it offers a lot of advanced features for user interactivity such as drag-and-drop, copy-paste, transactional state and undo management, palettes, overviews, data-bound models, event handlers and an extensible tool system for custom operations. Like jGraph, it is pure JavaScript and it does not depend on any JavaScript libraries or frameworks.

A big list of samples demonstrating how to build several common diagram types and showcase specific features and layouts, can be found on the library's home page [LTD15b]. The main reasons why GoJS was finally chosen are its developer-to-developer support[LTD15a] from the engineers who built the library and its simplicity.

4.1.2.1 Technical Concepts

In GoJS only the model, consisting of simple JavaScript objects, needs to be saved and restored. Many predefined tools and commands implement the standard behaviors that most diagrams need. Customization of appearance and behavior is mostly a matter of setting properties.

Diagrams consist of *Parts*: *Nodes* that may be connected by *Links* and that may be grouped together into *Groups*. All of these parts are gathered together in *Layers* and are arranged by *Layouts*. Each diagram has a *Model* that holds and interprets the application data to determine node-to-node, link and group-member relationships. Most parts are data-bound to the application data. The diagram automatically creates a node or a group for each data item in the model's `Model.nodeDataArray` and a link for each data item in the model's `GraphLinksModel.linkDataArray`. A sam-

ple input JavaScript object used to generate the swim lane flowchart in Figure 4.1 is presented in Listing B.1 in Appendix. The flowchart looks pretty similar to the Figure 3.7 fact that ensures that the layout is aligned with the initial design.

Each node or link is normally defined by a template that declares its appearance and behavior. Templates consists of *Panels* and *GraphObjects* such as *TextBlocks* and *Shapes*. There are default templates for all parts, but in order to achieve the desired appearance and behavior custom templates are specified. Nodes can be either positioned manually (interactively or programmatically) or be arranged automatically by the `Diagram.layout` and by each `Group.layout`. The `Object Node` template used for REDCloud's graph editor is presented as an example in Listing B.2 in Appendix.

A GoJS diagram has a number of tools that perform interactive tasks such as selecting parts, dragging them and drawing a new link between two nodes. Tools handle mouse and keyboard events. The `ToolManager` determines which tool should be running, depending on the mouse events and current circumstances. Furthermore, each diagram has a `CommandHandler` that implements various commands, such as delete, undo, copy and paste. `CommandHandler` interprets keyboard events, such as `control-Z`, when `ToolManager` is running.

Overview control allows the user to see the whole model and to control what part of it the diagram displays while the *Palette* control holds parts that the user may drag-and-drop into a diagram. Both features are displayed in Figure 4.1 reflected by the code in Listing B.4 in Appendix. Note that in overview control the `standardMouseWheel` tool is being used in order to allow zooming in and out from this pane.

The template implementation may change the appearance of the node or link when it is selected. The diagram may also add *Adornments* to indicate selection and to support tools such as resizing a node or reconnecting a link. Adornments are being used on infoboxes and context menus. Listing B.3 in Appendix displays the code responsible for rendering the swim lane context menu.

GoJS interprets swim lanes as groups and only allows nodes to be positioned in swim lanes. In other words, nodes can not just stand in the middle of two swim lanes. For this reason, the metamodel needed to be changed. `Node` references an Area which can either be a Region or a `Swim Lane`, as shown in Figure 4.2.

4.1.3 Graph Editor Main Features

The main javascript code for this project is split into two files: **project.js** and **modals.js**. In **project.js** all generic behaviors, templates and diagram settings are set up while on **modals.js** all modal behaviors and custom user operations are defined. In **modals.js**, transactions are being used to record all user actions and support undoing and redoing of those.

Since REDCloud is created from scratch, there are a lot of features implemented. The majority of them are listed in Table C.1 in Appendix. Documenting each one in this report would be irrelevant and for this reason the reader can refer to the repository hosted on `https://bitbucket.org/hsto/redcloud` for the code.

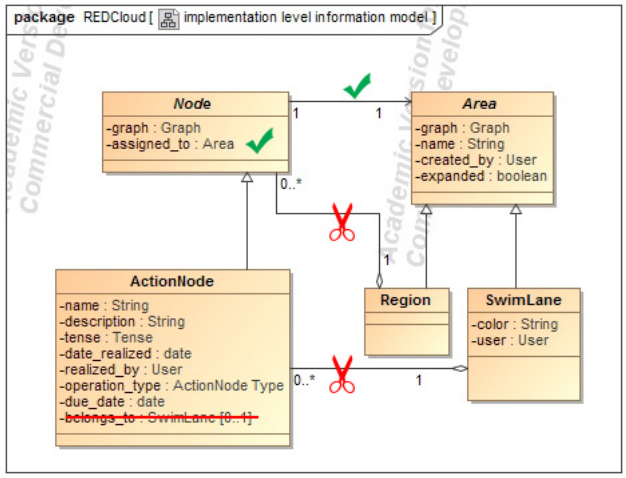


Figure 4.2: Changes in the Implementation Level Information Model

4.1.3.1 Tenses

In order to allow workflow planning, a goal set in 2.2, tenses are introduced allowing users to realize and archive nodes. Figure 4.3 shows steps to archive an action node.

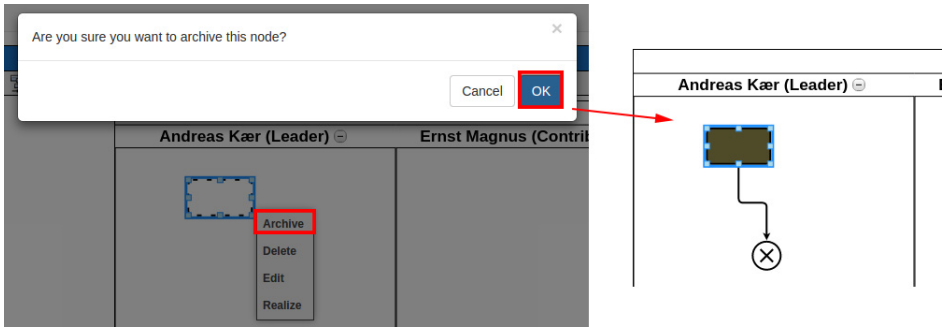


Figure 4.3: Archiving an Action Node

4.1.3.2 Team Synchronization

As discussed in 3.4.4, team synchronization is enhanced using sync edge types. Figure 4.4 shows how different nodes can get synchronized.

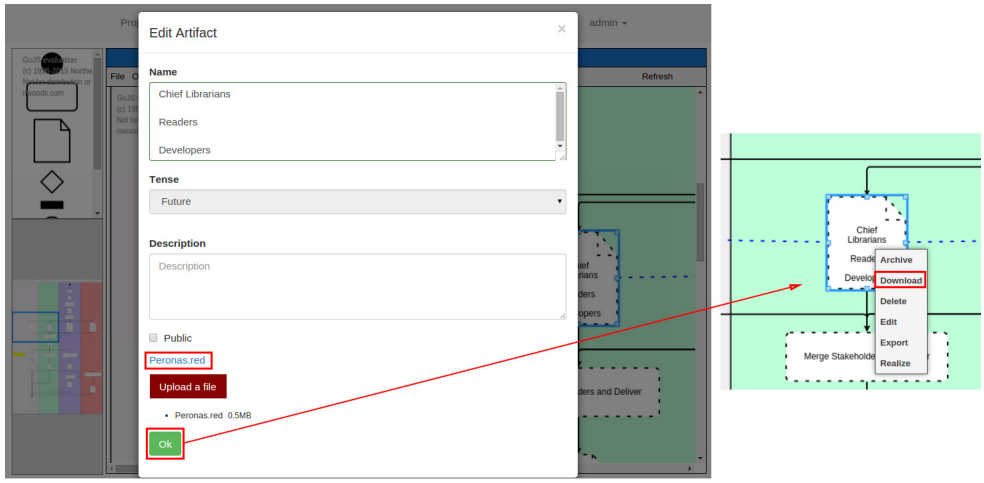


Figure 4.5: File Sharing

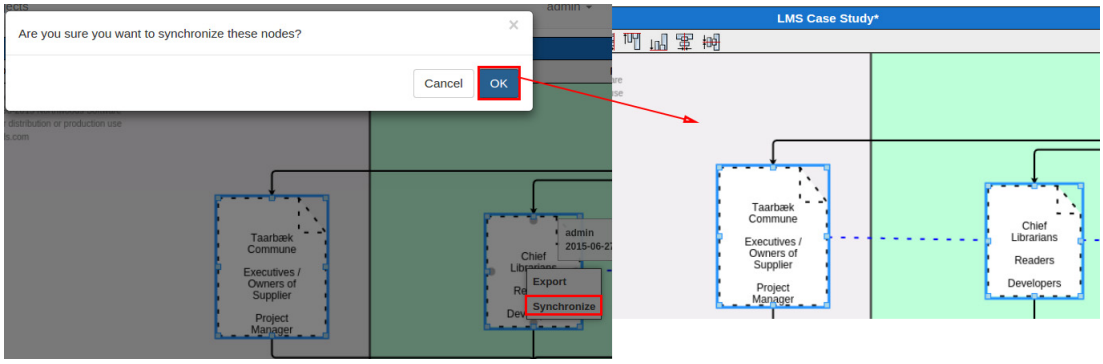


Figure 4.4: Nodes Synchronization

4.1.3.3 File Sharing

In Section 1.3.3 the nead for file sharing was discussed. This is achieved by allowing attaching resources to object nodes. A modal dialog pops up on double clicking of an object node, which includes the option to upload a file. Later on users can download the uploaded file using the corresponding context menu. Figure 4.5 visualizes this feature.

4.1.3.4 Reusable Components

In Section 1.3.4, the scenario where course participants are asked to import an exported **.red** file was discussed. Exporting an object node implies enabling it `public`

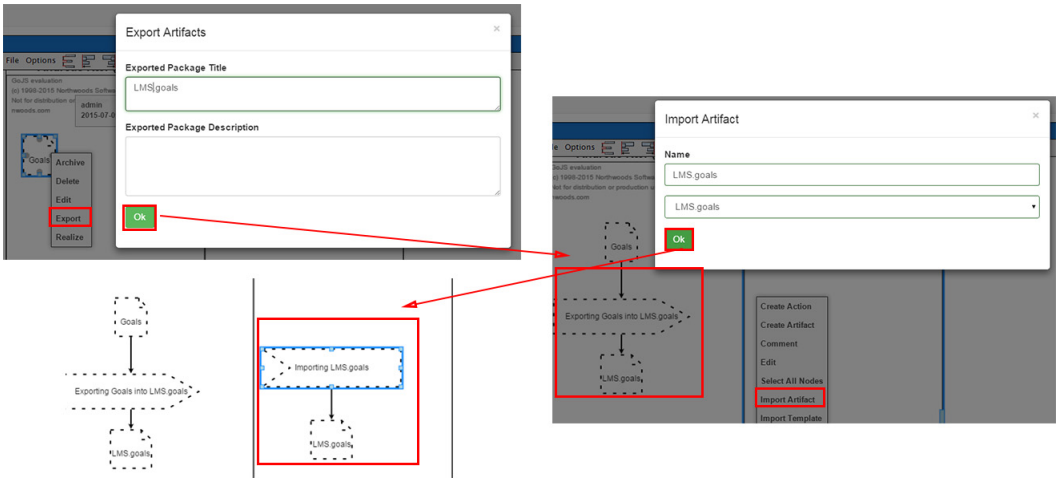


Figure 4.6: Exporting & Importing an Object Node

flag. This can be achieved either by editing its `public` flag from its edit form or by using the **Export** context menu. Importing it is done in a similar way. This feature is displayed in Figure 4.6.

4.2 Backend Web Framework

An application's frontend may be the most beautifully crafted web page, but if the application itself doesn't work, the application will be a failure. The backend of an application is responsible for things like calculations, business logic, database interactions and performance and most of the code that is required to make an application work is usually done on the backend. Backend code is run on the server, as opposed to the client. This means that backend developers not only need to understand programming languages and databases, but they must have an understanding of server architecture as well. If an application is slow, crashes often, or constantly throws errors at users, it is likely because of backend problems.

Backend development is not all ones and zeros though and it should be easy to understand and work with. Most backend languages – like *Ruby*, *Java*, *PHP* and *Python* – have standardized styles and idioms that make reading and writing code more efficient and enjoyable.

4.2.1 Why Python

One of the main reasons why Python is selected as the main implementation language is its academic versatility. As displayed in Figure 4.7, an analysis done by Philip Guo in July 2014 shows that Python earns popularity in universities in the U.S., fact

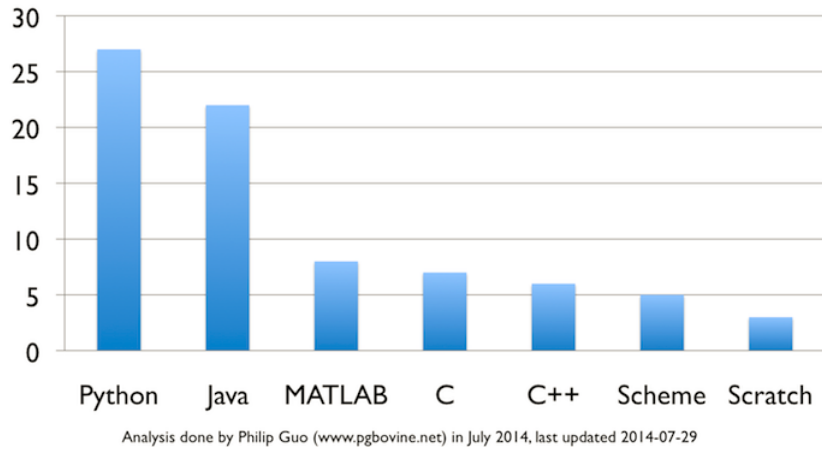


Figure 4.7: Programming Languages Used in Introductory Courses from Top 39 U.S. Computer Science Departments

that strengthens the maintainability of the project, a requirement set in the Analysis Chapter 2.

Additionally, there are numerous web servers working fluently in Python that support the WebSockets protocol, requirement set in Section 3.3. Some of them are *Tornado*, *pywebsocket* and *uWSGI*.

Last, Python provides flexibility and partial re-usability of the application. It has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a *module*; definitions from a module can be imported into other modules or into the main module. By well defining non-dependant modules (graph and project in our case) we can ensure that they can be re-used in the future for other dtu compute projects. Furthermore, Python comes with many different implementations.The most common Python implementation is called CPython and is written in C, but there are various other implementations built with different languages and on top of different virtual machines, such as Jython (using Java and JVM), IronPython (using C# and CLR), RubyPython (using Ruby and Ruby VM), Brython (using JavaScript and Javascript engine) and many more. With this in mind, we can create general purpose use modules which can then be plugged into other projects regardless the implementation language.

4.3 Django

Django is a Python framework, which follows the *model–view–controller (MVC)* software architectural pattern. Its primary goal is to ease the creation of complex, database-driven websites. It emphasizes reusability and pluggability of componenets, rapid development and the DRY principle. It comes with an ORM an incredibly

powerful database tool. It handles creation of the database, as well as inserts, updates, deletes and some quite advanced querying. Finally, it provides an optional administrative *CRUD* interface that is generated dynamically through introspection and configured via admin models.

4.3.1 Technical Concepts

Django was designed to make common Web-development tasks fast and easy. In this chapter its main concepts and the way these are used in the project are discussed.

4.3.1.1 Models

As mentioned in its official documentation [Fou15], django models are the single, definitive source of information about stored data, containing relevant essential fields and behaviours. Each model extends `django.db.models.Model` and each model attribute represents a database field. As discussed in the Analysis Chapter 2, the project is split into four main modules:

- The authentication module which corresponds to the **docServer/models/abstract.py** module, where django's `django.contrib.auth.models.User` model is extended with a `UserProfile` model holding some additional user details, like `phone` and `skype`. To allow inherited models to be able to downcast parent instances to their child types, the `InheritanceCastModel` is also defined. It is later extended by the `Graph.Node` parent model, fact that makes querying graphs a lot easier. Sample screen shots from the authentication interface are displayed in Figure 4.8.
- The **docServer/models/project.py** module, where all objects discussed in chapter 2.5.1 are defined.
- The **docServer/models/graph.py** module, where all objects discussed in Section 2.5.2 are defined.
- The **docServer/admin.py** module. Once the models are defined, Django can automatically create a professional, production ready administrative interface – a web site that lets authenticated users add, change and delete objects. Figure 4.9 displays the project admin interface, as designed in 3.6. The admin interface is hosted under `redcloud.compute.dtu.dk/admin`.

Sample code from these modules can be found in Listings B.6, B.7, B.8 and B.9 in Appendix.

Once a new model is created or modified, one has to run the django command-line utilities in Listing 4.1 to create and modify the database tables automatically.

```
1 python manage.py makemigrations
2 python manage.py migrate
```

Listing 4.1: Create and Run Django Migrations

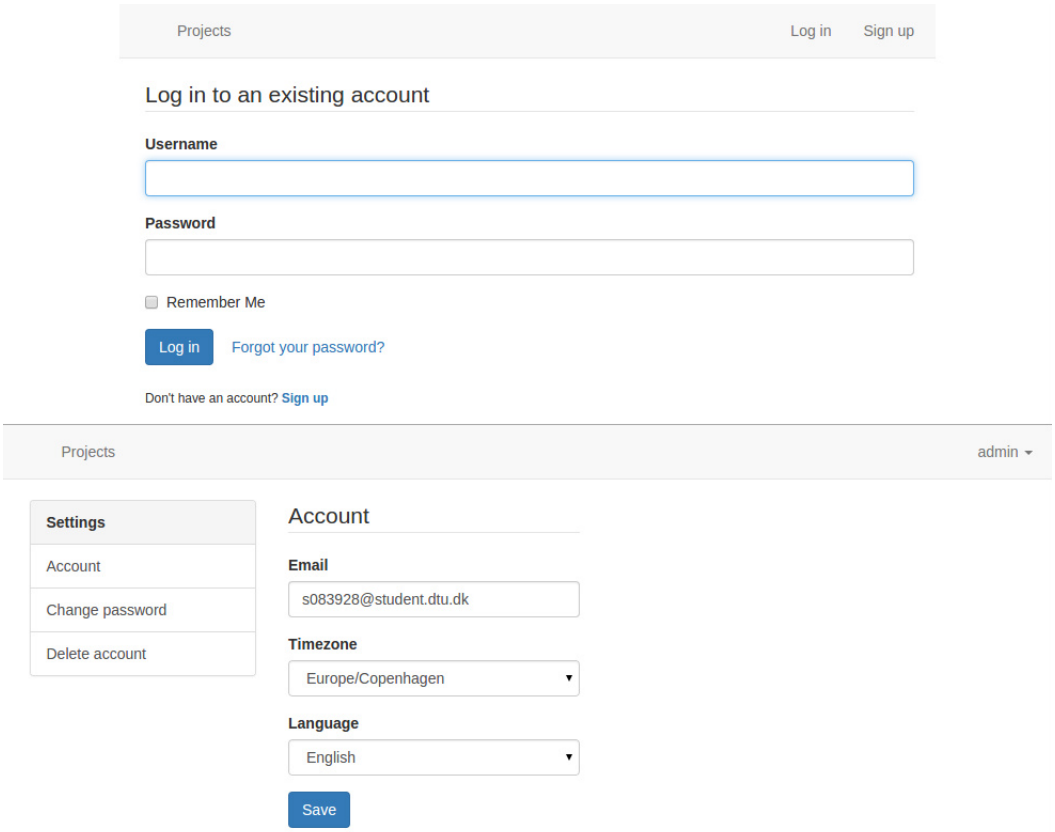


Figure 4.8: Authentication Interface

4.3.1.2 URL dispatcher, Views and Templates

A clean, elegant URL scheme is an important detail in a high-quality web application. Django lets you design URLs however you want, with no framework limitations by creating a *URLconf* module where URL patterns are mapped to Python view functions. A view function is simply a Python function that takes a web request and returns a web response. In this project, views serving and accepting json and xml can be found under **docServer/api_views.py** while views serving html content can be found under **docServer/views.py**. For the latter, django's template engine provides a powerful mini-language for defining the user-facing layer of your application, encouraging a clean separation of application and presentation logic. Templates, stored under **docServer/templates**, can be maintained by anyone with an understanding of HTML; no knowledge of Python is required. *Template inheritance* is a concept used in Djagno's template engine that lets you dramatically cut down on

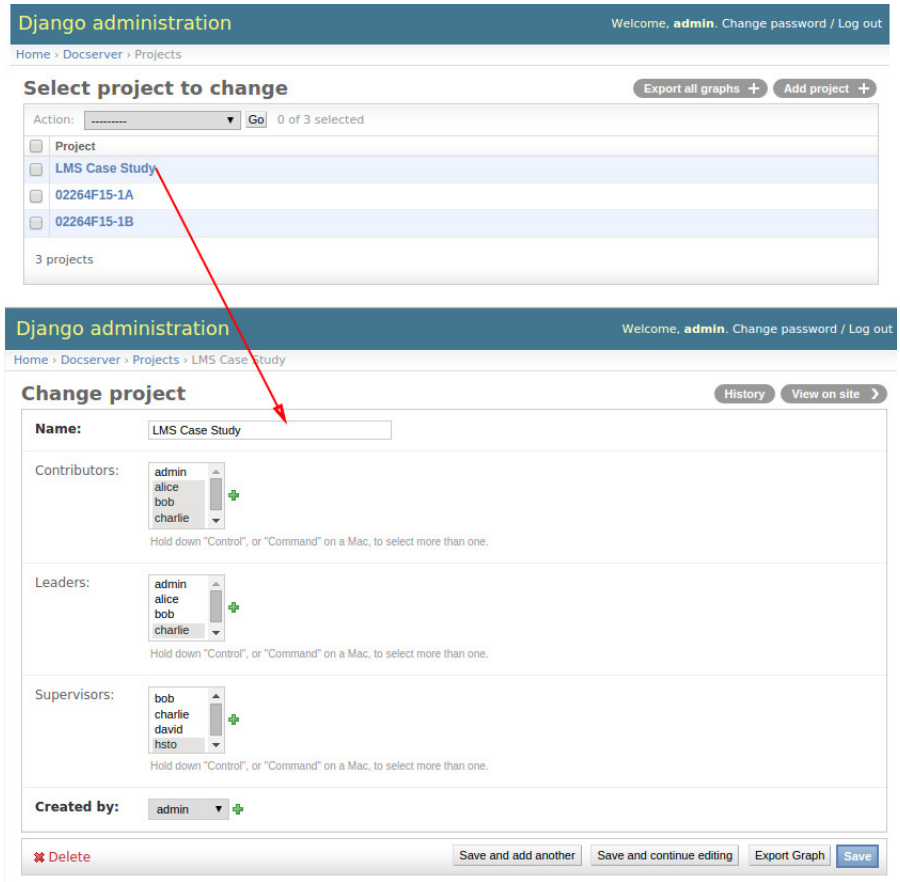


Figure 4.9: Project Administration Interface

redundancy in templates since each template has to define only what's unique to that template. Listing B.10 in Appendix displays sample code that is executed when REDCloud's homepage is requested.

4.3.2 Providing an API

One of the dependencies used that significantly decreased the time and the effort spent to reach the goals set for this project is *DjangoRESTFramework (DRF)*, a powerful and flexible toolkit that makes it easy to build web APIs. A fully functional web browsable API able to custom serialize the main ORM data sources that are used by GoJS was implemented in less than three days. DRF can be used in the future for providing a complete API to the RED client. In this project, it helped serializing python objects to json objects, a vital feature for connecting the backend and the fron-

tend of the application. All serializers are stored under **docServer/serializers.py**. Sample code of the project serializer is presented in Listing B.5 while a screen shot of the web browsable API result is shown in Figure A.4 in Appendix.

Furthermore, a DRF view was build to allow posting a csv file with projects and team members and thus automating administrative tasks. A sample CSV configuration file can be found under **docServer/participants.csv**. Postman request builder, a google chrome plugin, helped in testing REDCloud's API. The request displayed in Figure A.5 in Appendix, creates all projects and team members defined in the **participants.csv** file and assigns members the proper roles. It is considered to be a contribution to automate and ease group registration, as discussed in Section 1.3.1.

4.3.3 Satisfying ACID

In order to satisfy atomicity we need to ensure that each transaction either saves all the modifications or, in case something goes wrong, none. That means that if one part of the transaction fails, the entire transaction should fail and the database state should left unchanged. MySQL, PostgreSQL, Oracle and SQLite provide transactions support and are all supported out-of-the-box by django assuming the relevant Python libraries are installed.

Django's `transaction.atomic()` is a thin abstraction over the transaction facilities of the database. So its behaviour really depends on the database layer, which in turn depends on which database it is and what transaction isolation level is being used. In this project PostgreSQL is used as a database and the code snippet below shows the capabilities of `transaction.atomic()`.

```
# When test_one and test_two are called sequentially
# test_two blocks until test_one finishes
def test_one():
  with transaction.atomic():
      t = Test.objects.filter(id=1).select_for_update()[0]
      sleep(10)
      t.value = t.value + 10
      t.save()

def test_two():
  with transaction.atomic():
      t = Test.objects.filter(id=1).select_for_update()[0]
      t.value = t.value - 20
      t.save()
```

Listing 4.2: Project Module Sample Models

4.4 Deployment

This section explains the technology used in order to deploy the application to the REDCloud server.

4.4.1 Docker

According to the analysis made in Chapter 2, the application should be fairly easy to maintain and its dependencies should be well defined. Hence deployments are handled by docker, a high level tool that offers several powerful functionalities using *Linux Containers (LXC)* as a base technology. LXC refers to capabilities of the Linux kernel that allow sandboxing processes from one another and controlling their resource allocations. This is achieved through a combination of kernel security features such as namespaces, mandatory access control and control groups. Docker is an open platform for developers and sysadmins that does the heavy lifting of building, shipping (distributing) and running containers. It is often considered as a lightweight form of virtualisation that uses operating system mechanisms on the host to achieve strong isolation and resource management instead of requiring full-blown virtual machines. Imagine a virtual machine without the overhead that comes with running a separate kernel and simulating all the hardware. These isolated entities are called *docker containers* and they can be transferred to any machine running docker regardless the kernel, the platform distribution or deployment model, offering portability across machines. They include the minimal runtime requirements of the application and their reduced size allows rapid application deployment.

Docker containers are created using base *images*. An image can be basic with nothing but the operating-system fundamentals, or it can consist of a more sophisticated pre-built application stack ready for launch. A *dockerfile* is a script, composed of various commands and arguments listed successively to automatically perform actions on a base image in order to create or form a new one [Tez15] [Tea15]. Using a dockerfile, not only provides an automated build pipeline but also reduces "works on my machine" finger-pointing and risk of problems with application dependencies. Thus, differences in OS distributions and underlying infrastructure are abstracted away and application maintenance is simplified [Hat15]

Docker hub is a hosted registry server that can hold docker images. It offers git-like capabilities and image versioning making it easy to roll-back to previous versions and securing the deployment workflow. In this project, docker hub is used in order to publish a base image (*redcloud server*) for the REDCloud web application[1]. Then on deployment a new container is built on top of this image, which is only *really* pulled if docker hub holds a different version than what the server does, fact that speeds things up. For this reason two docker files are maintained:

- **docServer/ServerDockerfile** - base image.
- **docServer/Dockerfile** - running container.

4.4.2 Nginx-uWSGI-Fabric

To deploy a Django application we need to hook it into an industrial-strength Web server such as Apache. The *Web Server Gateway Interface (WSGI)* is a specification

[1] `https://registry.hub.docker.com/u/dtured/redcloud_server/`

for simple and universal interface between web servers and web applications or frameworks for the Python programming language. To process a WSGI request, the server side provides environment information and a callback function to the application side. The application processes the request, and returns the response to the server side using the callback function it was provided. The so-called WSGI middleware implements both sides of the API so that it can intermediate between a WSGI server and a WSGI application: the middleware acts as an application from some WSGI server's point of view and as a server from some WSGI application's point of view. A "middleware" component can perform such functions as routing a request to different application objects based on the target URL, after changing the environment variables accordingly, allowing multiple applications or frameworks to run side-by-side in the same process, load balancing and remote processing and performing content postprocessing, such as applying *XSLT* stylesheets.

In this project *nginx* is used as a web server. It intercepts and routes HTTP requests; requests for static content are being served directly off the server's file system while requests for dynamic content are routed to a *uWSGI* server. uWSGI uses a *pre-forking* technique as many other Unix daemons. This means that in order to to save the amount of time taken to fork a child process they keep a pool of processes waiting for work to come in. When a job arrives then one of the existing processes is used and the overhead of the `fork()` system call is saved. The main reasons why uWSGI was chosen are that nginx supports the uwsgi protocol natively and that it performs slightly better than other alternatives such as *Gunicorn* [Gri12]. Both nginx and uWSGI provide excellent support for the WebSockets protocol, as required in the Design Section 3.3.

Fabric is a library and command-line tool for streamlining the use of SSH for application deployment or system administration tasks. All tasks allowing our interaction with the server can be found under the **fabfile.py**. The deploy task used for deploying the application pulls the application code from a bitbucket repository, stops and removes the running container, builds a fresh image and then runs it linking it to a PostgreSQL db and a Redis cache container, as shown in Listing 4.3 below

```
1  from __future__ import with_statement
2  from fabric.api import *
3  from fabric.contrib.console import confirm
4
5  env.hosts = ['redcloud.compute.dtu.dk']
6  env.user = "s083928"
7
8  def prod():
9      env.APP_ENV = "prod"
10
11 def _pull():
12     with cd("/home/s083928/dtu-masterthesis/"):
13         run("git pull --rebase")
14
15 def deploy():
16     import time
17     print("Executing deploy() on %s as %s, %s" % (env.host, env.user, time.strftime('%Y%m%d%H%M%S')))
18     _pull()
19     sudo("docker stop redcloud")
20     sudo("docker rm redcloud")
21     with cd("/home/s083928/redcloud/docServer"):
22         sudo("docker build -t redcloud_server .")
```

```
23          sudo("docker run -e APPLICATION_ENV=prod --link redcloud-redis:redis --link redcloud-postgres:
                db --name redcloud -v /srv/redcloud/media:/home/code/media  -p 443:443 -p 80:80 -d
                redcloud_server")
24          sudo("docker tag -f redcloud_server dtured/redcloud_server")
25      print("Done with deployment, %s" % (time.strftime('%Y%m%d%H%M%S')))
```

Listing 4.3: Deploy Task

One can notice the `-v /srv/redcloud/media:/home/code/media` argument that is applied when the container runs. This is used to mount the host directory **/srv/redcloud/media** as a data volume in our web application container at **/home/code/media**. Having this volume mounted, user uploaded files are kept safe when when stopping and removing a running docker container. Similar rule for stored data applies with the db container linking.

4.4.3 RequireJS and Almond

RequireJS does a great job of packaging up JavaScript files into defined, decoupled modules, and on top of that provides a dependency loading mechanism. In order to load all the modules into a single production ready file *almond.js* is used, an *Asynchronous Module Definition (AMD)* API shim that is very small, so it can be used in place of RequireJS when all modules are built into one file using the *RequireJS optimizer (r.js)* [Tea12].

Django-require is a Django staticfiles post-processor and it is used in REDCloud in order to optimize our static assets using the excellent **r.js** optimizer and compile standalone modules using the **almond.js** shim. Running `python manage.py collectsatic`, all our static files get optimized and copied from **/static** (used on development environment) to **/s** directory (used on production environment).

Using all technologies mentioned above, the deployment workflow shown in Figure 4.10 is achieved. Steps 1) and 2) are only needed when the server needs to get updated while normally deployments are performed running the commands from Listing 4.4.

```
1 python manage.py collectstatic
2 git add . -A && git commit -m "a commit message" && git push
3 fab prod deploy migrate
```

Listing 4.4: Deployment Commands

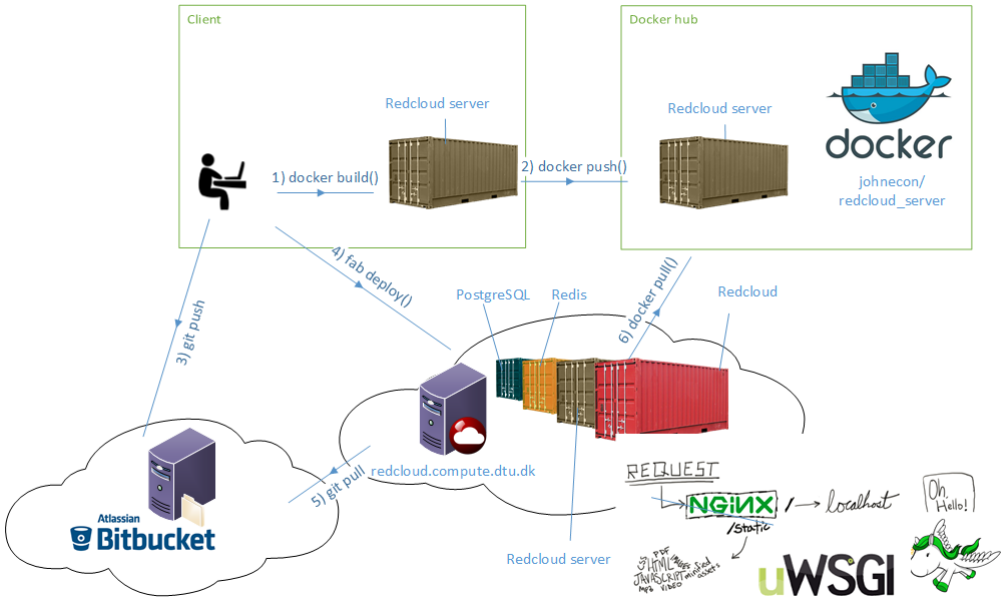


Figure 4.10: Deployment Flow

CHAPTER 5

Evaluation

Functionality, Usability, Reliability, Performance and Supportability (FURPS) model [1] has been used to validate the outcome of the solution of this thesis.

5.1 Functionality

As discussed in Section 3.1 earlier, user stories are collected and transformed in behavioural tests. In *behaviour-driven development BDD*, acceptance tests provide the starting point for the software design flow and serve as a basis for the communication between developers, QA and non-technical or business participants in a software project. According to [BR14], it was originally named in 2003 by Dan North as a response to *test-driven development (TDD)*, including acceptance test or customer test driven development practices as found in extreme programming. In this agile software development technique, acceptance tests are written in natural language in order to ensure a common understanding between all members of the project.

In this project, the *django-behave* python module was used as a `Django.TestRunner` for *behave*, an open source behavior-driven development framework. The tests written will not only provide confidence to developers performing refactorings in the future but also offer human-readable stories describing and documenting the features of the application.

All the tested features can be found under **bdd_tests/features**. Figure 5.1 shows behat's output for Synchronization feature.

[1] A model for classifying software quality attributes developed by Hewlett-Packard

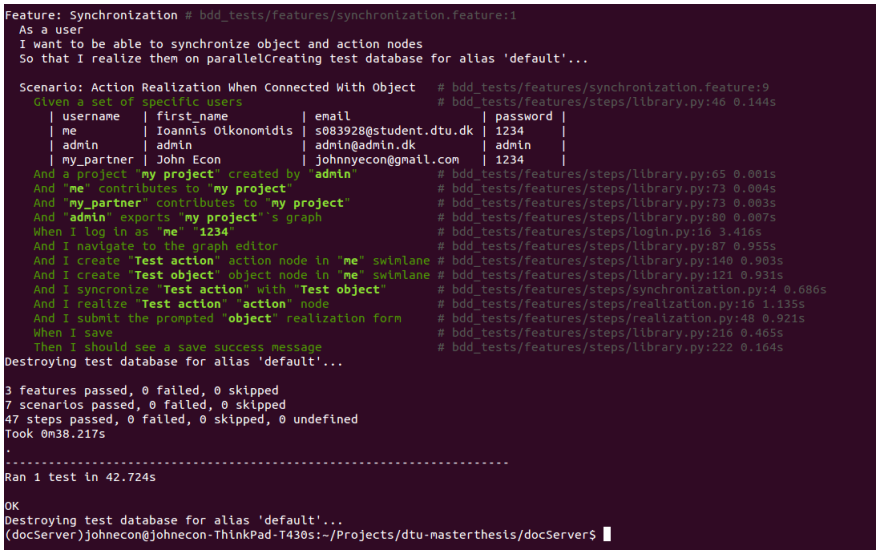


Figure 5.1: Behat Output

The full list of the test cases performed and the results can be found in Table C.1 in Appendix. The following platform has been used to run the testing:
Operating System: Linux - Ubuntu 64 bit
Processor: Intel Core i7 CPU@2.9GHz
Memory: 8 GB

5.2 Usability

In order to ease the use and learnability of the graph editor some extra elements are introduced. Figure 5.2 visualizes features discussed in this section.

A menu bar that contains drop down menus is designed. The menu bar's purpose is to supply a common housing for the graph editor's specific menus. It provide access to the following functions:

- **Load** - Loads the current project. Pressing this button any unsaved changes are lost. The functionality is similar to refreshing the browser.
- **Save** - Saves any changes occurred in the current project.
- **Export to JSON** - Downloads the current project on JSON format.
- **Open** - Opens a project from a JSON file.
- **Render as SVG** - Renders a SVG that includes all the graph in a new page and allows printing it.

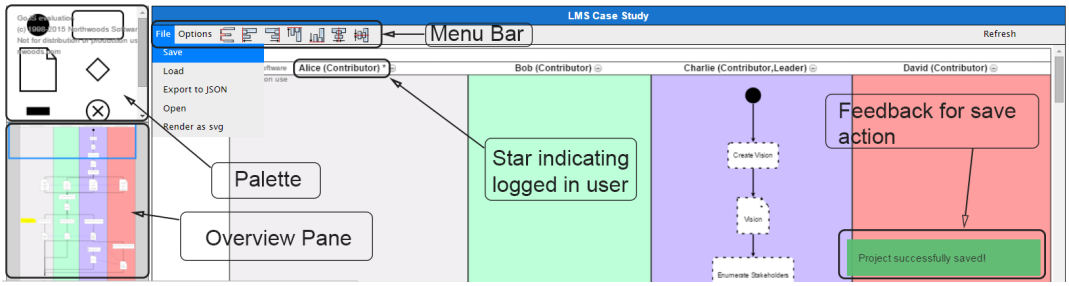


Figure 5.2: Graph Editor Usability Features

- **Show Grid** - Displays a grid, a series of vertical and horizontal lines that are used to subdivide a page vertically and horizontally.
- **Align** - Aligns a selection of nodes.
- **Refresh** - Saves and loads the current project.

A *palette* next to the editor from which a user can drag and drop nodes, eases the creation of nodes and speeds things up. In order to avoid conflicts when people working at the same project concurrently an asterisk (*) is displayed next to the corresponding swim lane, indicating an logged in user. Last, when a user engages and performs an action, the system needs to respond to acknowledge the action and to let the user know what it is doing. This is achieved with informative success or failure messages that appear on the user's browser.

5.3 Reliablitiy

DTU compute department run a nessus[2] security scan in the REDCloud server. The only complaint was the self-signed certificate, which complies with DTU compute's security policies.

Furthermore, in order to make sure that it is possible to rollback to any previous state of the application and keep important data safe and secure, backups of the database and the user uploaded files are being captured in a daily basis. This is scheduled using `crontab`. Figure 5.3 displays the current schedule configuration for the REDCloud server. Every morning at 05:15 a database snapshot is taken and stored under **/srv/redcloud/db/dumps** and at 05:30 `rzyncbackup` [3] runs. `rzyncbackup` uses the `rsync` command and synchronises the directories against a remote backup directory on a dedicated server (`userbackup1.compute.dtu.dk`). Figure 5.4 lists backups that the server has performed by the time of writing. Listing 5.1 shows how to restore data from a database snapshot.

[2] A vulnerability assessment ,management solution
[3] A script provided by DTU compute https://itswiki.compute.dtu.dk/index.php/Backup$_o n_L inux_{/O} SX$

```
s083928@redcloud:~$ sudo crontab -l
15 5 * * * docker exec -t redcloud python3 manage.py dumpdata --natural --indent
=4 -e sessions -e contenttypes -e auth.Permission -e authtoken -e account > /srv
/redcloud/db_dumps/`date +\%Y\%m\%d\%H\%M\%S`.json
* 6 * * * drm

s083928@redcloud:~$ crontab -l
30 5 * * * rzyncbackup
```

Figure 5.3: Crontab Configuration on REDCloud Server

```
rs1:~ $ ls /backup/s083928/redcloud/srv/redcloud/db_dumps/
20150706053001.json  20150707053001.json  20150708053001.json
```

Figure 5.4: List of db Dumps on Backup Server

```
1 scp s083928@userbackup1.imm.dtu.dk:/backup/s083928/redcloud/db_dumps/{db_dump}.json ~/Desktop/
2 cd redcloud/docServer
3 python manage.py loaddata ~/Desktop/{db_dump}.json
```

Listing 5.1: Steps to Restore a db Dump

5.4 Performance

REDCloud server has been stress tested using *Apache HTTP Server Benchmarking Tool (ab)*. 100 simultaneous connections are performed until a thousand requests are met. The results, displayed in Figure A.6 in Appendix, were expected to look quite satisfying simply because no complicated database queries are executed on the background for generating the home page HTTP response.

The figures for the graph editor page though were quite different, as shown in Figure A.7 in Appendix. The numbers in this case look quite disappointing since the server handles less than 2 requests per second and the mean time per request is 22 seconds. In order to tackle this issue caching techniques are introduced. Redis, an open source advanced key-value cache, is used in order to cache result of complicated queries executed when populating a graph. The results are stored directly in a json format and cache keys get invalidated every time the graph gets updated. As shown in Figure 5.5 the mean time dropped by 14 seconds while the server handles 11 requests per second when responding using cache.

```
ab -k -c 100 -n 1000 -A admin:admin
redcloud.compute.dtu.dk/projects/3/load
```

```
Server Software:        nginx/1.4.6
Server Hostname:        redcloud.compute.dtu.dk
Server Port:            80

Document Path:          /projects/3/load
Document Length:        23713 bytes

Concurrency Level:      100
Time taken for tests:   88.602 seconds
Complete requests:      1000
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      23961000 bytes
HTML transferred:       23713000 bytes
Requests per second:    11.29 [#/sec] (mean)
Time per request:       8860.184 [ms] (mean)
Time per request:       88.602 [ms] (mean, across all concurrent requests)
Transfer rate:          264.10 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        2   46  84.3     14    1011
Processing:  4891 8406 898.9   8495   12893
Waiting:     4753 8353 901.9   8468   12785
Total:       4908 8451 910.5   8513   13120

Percentage of the requests served within a certain time (ms)
  50%   8513
  66%   8559
  75%   8606
  80%   8634
  90%   8763
  95%   9461
  98%  11638
  99%  12382
 100%  13120 (longest request)
```

Figure 5.5: Graph Editor Stress Test Using Redis Cache

A code snipet on how caching is handled is displayed in Listing B.11 in Appendix.

5.5 Supportability

A set of tools that enhance supportability are listed below.

- **Test suite** - It is easier for a developer to understand, refactor and add a new feature when all existing features are tested. For this reason the test suite is considered to function as the main user guide for the application.

- **Pivotal Tracker** - All the tasks that the author went through in order to develop this tool are described and documented in pivotal tracker, which is considered to function as a secondary application documentation tool.

- **Server Technology** - The server technology used (nginx, uWSGI, Django) provide high configurability. Nginx, uwsgi and django config files can be found under **redcloud/docServer/configs**.

As a proof of concept, the *template import* feature was implemented in less than 3 hours a week before submitting this report.

CHAPTER 6

Conclusion

6.1 Summary

The main goals of this thesis was to create a prototype that eases the communication and collaboration of stakeholders for a RE project and in the same time is able to connect and extend the RED client. This is achieved using activity diagrams and implemented using web technologies.

A real life scenario from the course 02264-Requirements Engineering is described in details and functions as a base for the analysis and design chapters.

In the Analysis Chapter 2, a stakeholder analysis is conducted in order to identify high-level goals. The system architecture together with its metamodel are analysed in order to have a better understanding of how the system components work and put a scope on the functional requirements.

The Design Chapter 3 discusses design choices taken while developing REDCloud. The real life scenario and the results of the analysis are used for sketching the main features of the application. This gives an overview of its main screens to the reader and allows the implementation of the tool.

The Implementation Chapter 4 documents software and frameworks used for the purpose of enhancing the maintenance of this project.

The outcome of this thesis was tested and evaluated in the Evaluation Chapter 5. The real life scenario described in 1.3 is implemented as shown in Figure 6.1 and it is available at `redcloud.compute.dtu.dk/projects/3`.

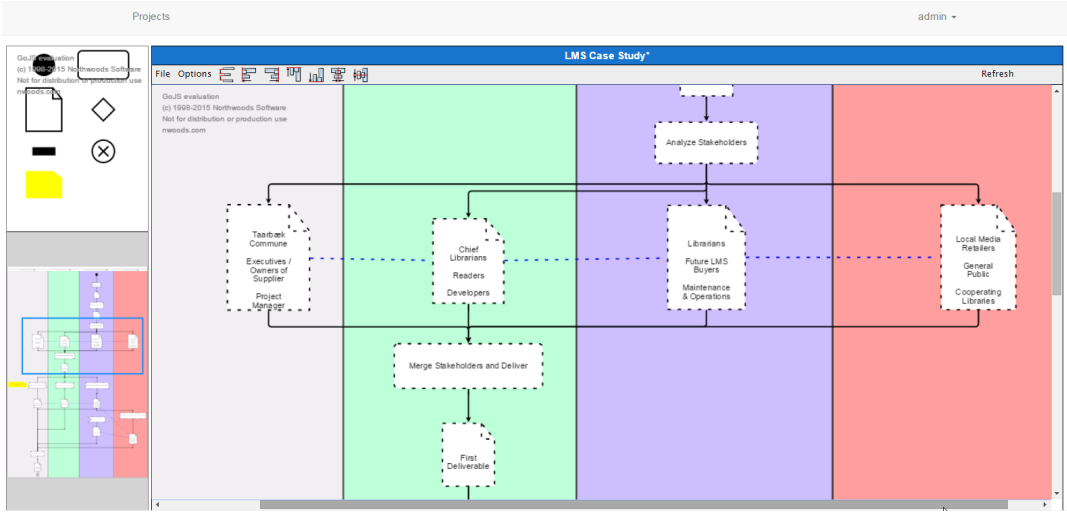


Figure 6.1: LMS Case Study in REDCloud

The main challenge for this thesis was to get familiar with all the technologies used for the implementation, testing and deployment of the tool.

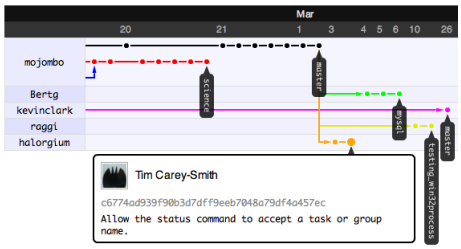
6.2 Future Work

All features that are left for future development are summarized in the list below.

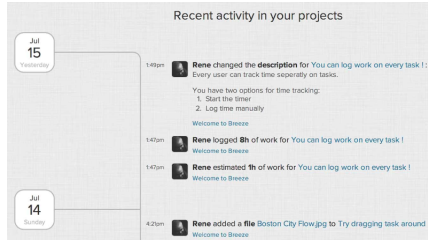
1. Build an API to connect the REDCloud server with the RED client.
2. Use WebSockets to allow real time notifications.
3. Right now a project has many graphs but the user interface only displays one graph attached to a project. Improve the UI so that it supports many graphs (activities) per project.
4. In order to allow sub-activities to get collapsed and expanded in a graph, users should be able to group a node selection from a graph.
5. Users should be able to set due dates in action and object nodes.
6. Django's administration area should be overwritten since there is a lot of space for improvement on the UI.
7. Allow sorting nodes by time and display a time axis next to the graph editor.
8. Enable email notifications on demand.

APPENDIX A

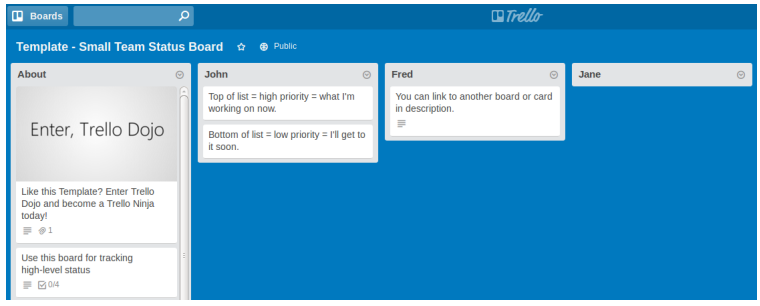
Appendix A - Figures



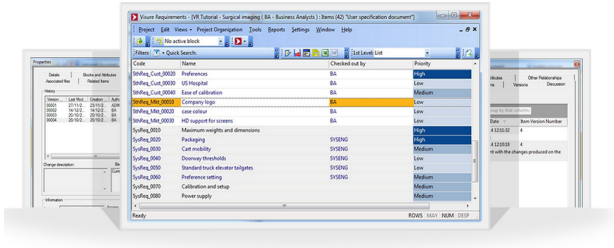
(a) Github's Network Graph



(b) Breeze's Activity Screen



(c) Small Team Status Board On Trello



(d) Collaboration Visualization On Visure

Figure A.1: Project Visualization Techniques

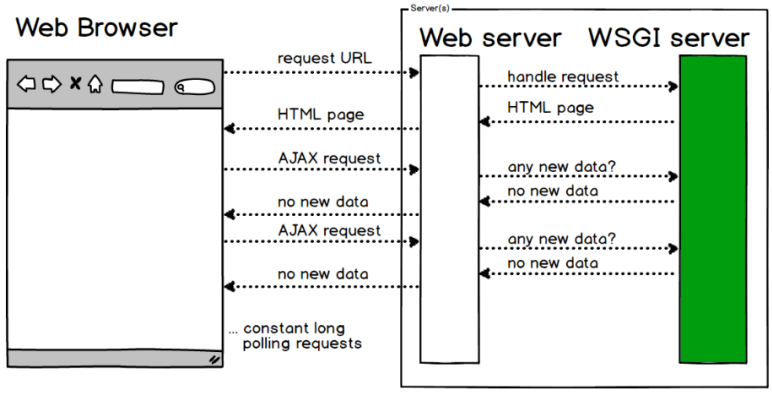


Figure A.2: Polling via AJAX

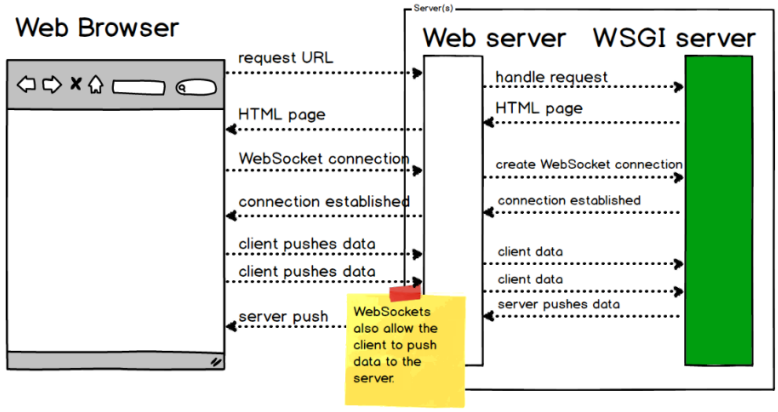


Figure A.3: WebSockets

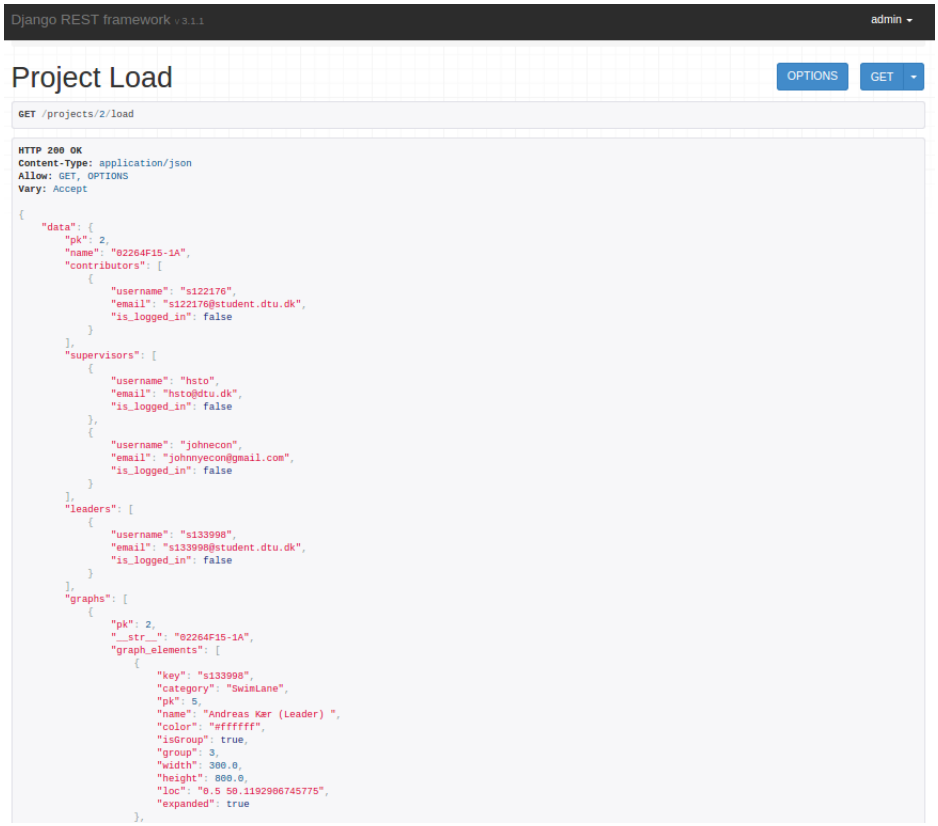


Figure A.4: Serializing a Project with DRF

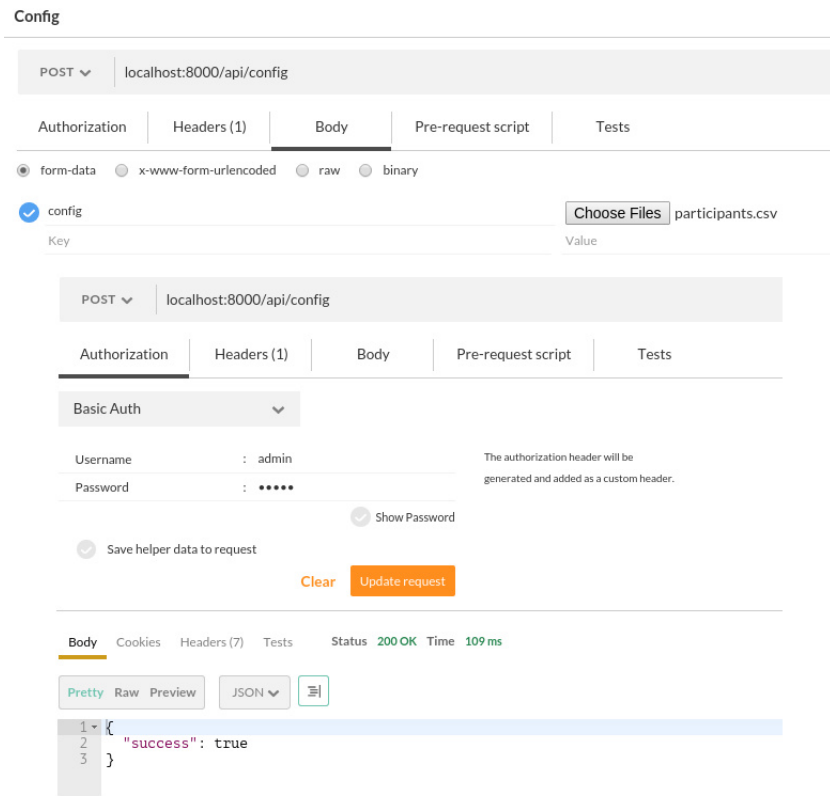


Figure A.5: Automating Administrative Tasks with Postman

```
ab -k -c 100 -n 1000 redcloud.compute.dtu.dk/
```

```
Server Software:        nginx/1.4.6
Server Hostname:        redcloud.compute.dtu.dk
Server Port:            80

Document Path:          /
Document Length:        3828 bytes

Concurrency Level:      100
Time taken for tests:   20.146 seconds
Complete requests:      1000
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      4176000 bytes
HTML transferred:       3828000 bytes
Requests per second:    49.64 [#/sec] (mean)
Time per request:       2014.581 [ms] (mean)
Time per request:       20.146 [ms] (mean, across all concurrent requests)
Transfer rate:          202.43 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        2  513 1080.9    123    7263
Processing:   213 1422 895.6   1463   11940
Waiting:      213 1411 872.3   1459   11680
Total:        535 1936 1364.1   1651   13002

Percentage of the requests served within a certain time (ms)
  50%   1651
  66%   1684
  75%   1846
  80%   2102
  90%   2604
  95%   4067
  98%   8265
  99%   8561
 100%  13002 (longest request)
```

Figure A.6: Home Page Stress Test

```
ab -k -c 100 -n 1000 -A admin:admin
redcloud.compute.dtu.dk/projects/3/load
```

```
Server Software:        nginx/1.4.6
Server Hostname:        redcloud.compute.dtu.dk
Server Port:            80

Document Path:          /projects/1/load
Document Length:        13777 bytes

Concurrency Level:      100
Time taken for tests:   227.974 seconds
Complete requests:      1000
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      14025000 bytes
HTML transferred:       13777000 bytes
Requests per second:    4.39 [#/sec] (mean)
Time per request:       22797.369 [ms] (mean)
Time per request:       227.974 [ms] (mean, across all concurrent requests)
Transfer rate:          60.08 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        2   10  15.8      5     207
Processing:   616 21680 3878.5  22747   23513
Waiting:      610 21671 3877.9  22739   23498
Total:        620 21690 3873.1  22754   23518

Percentage of the requests served within a certain time (ms)
  50%  22754
  66%  22872
  75%  22932
  80%  22977
  90%  23107
  95%  23188
  98%  23289
  99%  23324
 100%  23518 (longest request)
```

Figure A.7: Graph Editor Stress Test

APPENDIX B

Appendix B - Code Samples

```
{
  "nodeDataArray":[
    {
      "category":"Object",
      "key":3,
      "name":"obj node",
      "loc":"152.0 102.0",
      "group":"s122176"
    },
    {
      "category":"Start",
      "key":4,
      "loc":"152.5 38.0",
      "group":"s122176"
    },
    {
      "key":"s133998",
      "category":"SwimLane",
      "name":"s133998",
      "color":"#914591",
      "isGroup":true
    },
    {
      "key":"s122176",
      "category":"SwimLane",
      "name":"s122176",
      "color":"#ffffff",
      "isGroup":true
    }
  ],
  "linkDataArray":[
    {
      "from":4,
      "to":3,
      "pk":1
    }
  ]
}
```

Listing B.1: JS Objects Representing a Sample Swim Lane Flowchart

```
myDiagram.nodeTemplateMap.add("Object",
  $(go.Node, "Spot", nodeStyle(),
      $(go.Panel, "Auto",
          $(go.Shape, "File",
              {
                // INFO-SHAPE is a predifined GoJS shape
                  name: "INFO-SHAPE",
                  desiredSize: new go.Size(50, 60),
                  fill: "white",
                  strokeWidth: 2,
                  stroke: "black"
              }),
          $(go.TextBlock, {font: "9pt serif"},
            // every time the name property of the model changes
            // the displayed text gets updated
              new go.Binding("text", "name").makeTwoWay()
          ),
          {
              contextMenu: $(go.Adornment)
          }
      ),
      // ports allow linking
      makePort("T", go.Spot.Top, true, true),
      makePort("L", go.Spot.Left, true, true),
      makePort("R", go.Spot.Right, true, true),
      makePort("B", go.Spot.Bottom, true, true)
  )
);
```

Listing B.2: Object Node Template

```
var cxTool = myDiagram.toolManager.contextMenuTool;

cxTool.showContextMenu = function (contextmenu, obj) {
    // find the type of the selected element
    var category = getSelectionCategory();

    //setup local varriables and hide all cx menus
    this.setUpMenus();
    this.hideMenus();
    ...
    switch (category) {
        case SWIMLANE:
            // and display it
            this.swimLaneCxMenu.show();
            break;
        ...
    }
    this.display(category);
    this.currentContextMenu = contextmenu;
}
```

Listing B.3: Displaying Context Menus

```
 myPalette =
    $(go.Palette, "myPalette",
        {
            "animationManager.duration": 300,
            nodeTemplateMap: myDiagram.nodeTemplateMap,
            // list of objects the pallette holds
            model: new go.GraphLinksModel([
                {category: "Start"}, {category: "Object"},
                {category: "Choice"}, {category: "Fork"},
                {category: "End"}, {category: "Comment"}
            ])
        });
// overview pane definition
var myOverview =
    $(go.Overview, "myOverview",
        {observed: myDiagram, maxScale: 0.5, contentAlignment: go.Spot.
            Center});
myOverview.box.elt(0).stroke = "dodgerblue";
myOverview.toolManager.standardMouseWheel = function () {
    if (myOverview.observed === null) return;
    var cmd = myOverview.observed.commandHandler;
    if (myOverview.lastInput.delta === 0) return;

    if ((myOverview.lastInput.delta > 0 ? cmd.canIncreaseZoom() : cmd.
        canDecreaseZoom())) {
        if (myOverview.lastInput.delta > 0)
            cmd.increaseZoom();
        else
            cmd.decreaseZoom();
        myOverview.lastInput.bubbles = false;
    }
}
```

Listing B.4: Palette and Overview

```
from rest_framework import serializers
from docServer.models.graph import *

class ProjectSerializer(serializers.HyperlinkedModelSerializer):
  # instructions on how to serialize different fields
  # if not listed here, the model.Project.{field} will be used
    created_by = serializers.HiddenField(default=serializers.
        CurrentUserDefault())
    contributors = UserSerializer(many=True, read_only=True)
    supervisors = UserSerializer(many=True, read_only=True)
    leaders = UserSerializer(many=True, read_only=True)
    graphs = GraphSerializer(many=True, read_only=True)
    class Meta:
        model = Project
        # list of fields to be serialized
        fields = ('pk', 'name', 'contributors', 'supervisors', 'leaders', '
            graphs', 'created_by')
```

Listing B.5: Project Serializer

```
class UserProfile(AbsModel):
    user = models.OneToOneField(User)
    phone = PhoneNumberField(blank=True)
    skype = models.CharField(max_length=255, blank=True)
    comment = models.TextField(blank=True)

def user_post_delete(sender, instance, **kwargs):
    try:
        UserProfile.objects.get(user=instance).delete()
    except:
        pass

def user_post_save(sender, instance, **kwargs):
    try:
        UserProfile.objects.get_or_create(user=instance)
    except:
        pass

models.signals.post_delete.connect(user_post_delete, sender=User)
models.signals.post_save.connect(user_post_save, sender=User)
```

Listing B.6: Abstract Module Sample Models

```
# Extending InheritanceCastModel, allows us to call Node.cast()
class Node(InheritanceCastModel):
    graph = models.ForeignKey(Graph, related_name='nodes')
    created_by = models.ForeignKey(User, related_name="nodes_created")
    location_x = models.FloatField(default=0)
    location_y = models.FloatField(default=0)
    width = models.FloatField(default=0)
    height = models.FloatField(default=0)
    assigned_to = models.ForeignKey(Area, null=True, blank=True,
        related_name="nodes")

class ObjectNode(Node):
    name = models.CharField(max_length=255, blank=True)
    description = models.TextField(blank=True)
    is_exported = models.BooleanField(default=False)
    resource = models.ForeignKey(Resource, null=True, blank=True,
        related_name="object_nodes")
    tense = enum.EnumField(Tense, default=Tense.FUTURE)
    date_realized = models.DateTimeField(blank=True, null=True)
    realized_by = models.ForeignKey(User, null=True, blank=True,
        related_name="object_nodes_realized")
```

Listing B.7: Graph Module Sample Models

```
class Project(AbsModel):
    name = models.CharField(max_length=255)
    contributors = models.ManyToManyField(User, blank=True, related_name="
        projects_contributed")
    leaders = models.ManyToManyField(User, blank=True, related_name="
        projects_leaded")
```

```
    supervisors = models.ManyToManyField(User, blank=True, related_name="
        projects_supervised")
    created_by = models.ForeignKey(User, null=True, related_name="
        projects_created")

class Resource(InheritanceCastModel):
    name = models.CharField(max_length=255)
    project = models.ForeignKey(Project)
    created_by = models.ForeignKey(User)

class Folder(Resource):
    items = models.ManyToManyField('Resource', related_name="parent_folder",
         symmetrical=False, blank=True, null=True)

class File(Resource):
    file = models.FileField(upload_to=resources_upload_path, blank=True,
        null=True)
    format = models.CharField(max_length=255, blank=True)
```

Listing B.8: Project Module Sample Models

```
from django.contrib import admin
from docServer.models.project import *
from docServer.models.graph import *
from docServer.models.abstract import *

admin.site.register(Project, ProjectAdmin)
admin.site.register(UserProfile)
admin.site.register(File)
admin.site.register(Folder)
```

Listing B.9: Registering Models to Admin Interface

```
# urls.py
urlpatterns = patterns('', url(r'^$', 'docServer.views.index', name='home'))

# views.py
def index(request):
    params = {'current': 'home'}
    return render(request, 'index.html', params)

# templates/site_base.html
{% extends "base.html" %}
{% load staticfiles %}
{% block head_title %}{{ params.current }}{% endblock %}
{% block nav %}
<ul class="nav navbar-nav navbar-left">
    <li><a href="{% url 'project_list' %}">Projects</a>
</ul>
{% endblock %}
{%  block footer_base %}
<div class="container" style="height:100%">
    <hr>
    <footer>
```

```
        <p>&copy; Dtu, 2015</p>
    </footer>
</div>
{% endblock %}

# templates/index.html
{% extends 'site_base.html' %}
```

Listing B.10: Different Pieces of Code for Rendering the Home Page

```
@api_view(['GET', ])
@authentication_classes(
    (authentication.SessionAuthentication,
      authentication.BasicAuthentication,
      authentication.TokenAuthentication)
)
def project_load(request, pk):
    json = cache.get('project_{pk}'.format(pk=pk))
    if not json:
        project = Project.objects.get(pk=pk)
        serializer = ProjectSerializer(project)
        json = serializer.data
    cache.set('project_{pk}'.format(pk=pk), json, 60)
    return Response({"data": json})
```

Listing B.11: Caching in Django

APPENDIX C

Appendix C - Functional Tests

```
Feature: Auth-Login
    # As a [Role]
    # I want [feature]
    # So that [Benefit]
  As a user
  I want a functional login feature
  So that the application can authenticate a user

  Scenario: Successful sign up

    When I sign up as "me", "1234", "me@me.com"
    Then log in for "me" should be successful


  Scenario: Failing sign up
  Given a set of specific users
     | username     | first_name            | email                    |
         password |
     | me           | Ioannis Oikonomidis   | s083928@student.dtu.dk   | 1234
          |
    When I sign up as "me", "1234", "me@me.com"
    Then I should see "This username is already taken. Please choose another
        ."

  Scenario: Successful login

    Given a set of specific users
     | username     | first_name            | email                    |
         password |
     | me           | Ioannis Oikonomidis   | s083928@student.dtu.dk   | 1234
          |
    When I log in as "me" "1234"
    Then log in for "me" should be successful

  Scenario: Failed login

    Given a set of specific users
     | username     | first_name            | email                    |
         password |
     | me           | Ioannis Oikonomidis   | s083928@student.dtu.dk   | 1234
          |
```

```
    When I log in as "me" "1235"
    Then I should see a login error message
```

Listing C.1: Auth Feature

```
Feature: GE-Creation
  As a user
  I want to be able to create object and action nodes
  So that I can distribute tasks and allow collaboration of a team

  Scenario: Comment Creation

    Given a set of specific users
     | username     |
     | me           |
     | admin        |
     | my_partner   |
    And a project "my project" created by "admin"
    And "me" contributes to "my project"
    And "my_partner" contributes to "my project"
    And "admin" exports "my project"`s graph
    When I log in as "me" "1234"
    And I navigate to the graph editor of "my project"
    And I create "Some feedback" "comment" node in "me" swimlane
    Then I should see "Some feedback" "comment" node in "me" swimlane
    When I save
    Then I should see a save success message

  Scenario: Object Creation

    Given a set of specific users
     | username     |
     | me           |
     | admin        |
     | my_partner   |
    And a project "my project" created by "admin"
    And "me" contributes to "my project"
    And "my_partner" contributes to "my project"
    And "admin" exports "my project"`s graph
    When I log in as "me" "1234"
    And I navigate to the graph editor of "my project"
    And I create "Test object" "object" node in "me" swimlane
    Then I should see "Test object" "object" node in "me" swimlane
    When I save
    Then I should see a save success message

  Scenario: Action Creation

    Given a set of specific users
     | username     |
     | me           |
     | admin        |
     | my_partner   |
    And a project "my project" created by "admin"
    And "me" contributes to "my project"
```

```
    And "my_partner" contributes to "my project"
    And "admin" exports "my project"`s graph
    When I log in as "me" "1234"
    And I navigate to the graph editor of "my project"
    And I create "Test action" "action" node in "me" swimlane
    Then I should see "Test action" "action" node in "me" swimlane
    When I save
    Then I should see a save success message
```

Listing C.2: GE Creation Feature

```
Feature: GE-Editing
  As a user
  I want to be able to edit nodes
  So that I can configure the graph of a project

Scenario: Comment Edit

    Given a set of specific users
     | username     |
     | me           |
     | admin        |
     | my_partner   |
    And a project "my project" created by "admin"
    And "me" contributes to "my project"
    And "my_partner" contributes to "my project"
    And "admin" exports "my project"`s graph
    When I log in as "me" "1234"
    And I navigate to the graph editor of "my project"
    And I create "Some feedback" "comment" node in "me" swimlane
    And I change "Some feedback" "comment" node in "me" swimlane name to "
        Some Feedback 2"
    Then I should see "Some Feedback 2" "comment" node in "me" swimlane
    When I save
    Then I should see a save success message

Scenario: Swimlane Edit

    Given a set of specific users
     | username     |
     | me           |
     | admin        |
     | my_partner   |
    And a project "my project" created by "admin"
    And "me" contributes to "my project"
    And "my_partner" contributes to "my project"
    And "admin" exports "my project"`s graph
    When I log in as "me" "1234"
    And I navigate to the graph editor of "my project"
    And I change the color of "me" swimlane to "#fffff"
    Then The color of "me" should be "#fffff"
    When I save
    Then I should see a save success message

Scenario: Future Object Edit
```

```
44
45      Given a set of specific users
46       | username     |
47       | me           |
48       | admin        |
49       | my_partner   |
50      And a project "my project" created by "admin"
51      And "me" contributes to "my project"
52      And "my_partner" contributes to "my project"
53      And "admin" exports "my project"`s graph
54      When I log in as "me" "1234"
55      And I navigate to the graph editor of "my project"
56      And I create "Test object" "object" node in "me" swimlane
57      And I change "Test object" "object" node in "me" swimlane name to "Test2
            object"
58      Then I should see "Test2 object" "object" node in "me" swimlane
59      When I save
60      Then I should see a save success message
61
62  Scenario: Future Action Edit
63
64      Given a set of specific users
65       | username     |
66       | me           |
67       | admin        |
68       | my_partner   |
69      And a project "my project" created by "admin"
70      And "me" contributes to "my project"
71      And "my_partner" contributes to "my project"
72      And "admin" exports "my project"`s graph
73      When I log in as "me" "1234"
74      And I navigate to the graph editor of "my project"
75      And I create "Test action" "action" node in "me" swimlane
76      And I change "Test action" "action" node in "me" swimlane name to "Test2
            action"
77      Then I should see "Test2 action" "action" node in "me" swimlane
78      When I save
79      Then I should see a save success message
80
81  Scenario: Present Object Edit
82
83      Given a set of specific users
84       | username     |
85       | me           |
86       | admin        |
87       | my_partner   |
88      And a project "my project" created by "admin"
89      And "me" contributes to "my project"
90      And "my_partner" contributes to "my project"
91      And "admin" exports "my project"`s graph
92      When I log in as "me" "1234"
93      And I navigate to the graph editor of "my project"
94      And I create "Test object" "object" node in "me" swimlane
95      And I realize "Test object" "object" node
```

```
96      And I change "Realized Test object" "object" node in "me" swimlane name
            to "Test2 object"
97      Then I should see "Test2 object" "object" node in "me" swimlane
98      When I save
99      Then I should see a save success message
100
101 Scenario: Present Action Edit
102
103     Given a set of specific users
104      | username     |
105      | me           |
106      | admin        |
107      | my_partner   |
108     And a project "my project" created by "admin"
109     And "me" contributes to "my project"
110     And "my_partner" contributes to "my project"
111     And "admin" exports "my project"`s graph
112     When I log in as "me" "1234"
113     And I navigate to the graph editor of "my project"
114     And I create "Test action" "action" node in "me" swimlane
115     And I realize "Test action" "action" node
116     And I change "Realized Test action" "action" node in "me" swimlane name
            to "Test2 action"
117     Then I should see "Test2 action" "action" node in "me" swimlane
118     When I save
119     Then I should see a save success message
120
121 Scenario: Past Object Edit
122
123     Given a set of specific users
124      | username     |
125      | me           |
126      | admin        |
127      | my_partner   |
128     And a project "my project" created by "admin"
129     And "me" contributes to "my project"
130     And "my_partner" contributes to "my project"
131     And "admin" exports "my project"`s graph
132     When I log in as "me" "1234"
133     And I navigate to the graph editor of "my project"
134     And I create "Test object" "object" node in "me" swimlane
135     And I archive "Test object" "object" node
136     Then I should not be able to edit "Test object" "object" node in "me"
            swimlane
137     When I save
138     Then I should see a save success message
139
140 Scenario: Past Action Edit
141
142     Given a set of specific users
143      | username     |
144      | me           |
145      | admin        |
146      | my_partner   |
147     And a project "my project" created by "admin"
```

```
    And "me" contributes to "my project"
    And "my_partner" contributes to "my project"
    And "admin" exports "my project"`s graph
    When I log in as "me" "1234"
    And I navigate to the graph editor of "my project"
    And I create "Test action" "action" node in "me" swimlane
    And I archive "Test action" "action" node
    Then I should not be able to edit "Test action" "action" node in "me"
        swimlane
    When I save
    Then I should see a save success message
```

Listing C.3: GE Editing Feature

```
Feature: GE-Realization
  As a user
  I want to be able to realize object and action nodes
  So that I can distinct between planned and realized work

  Scenario: Object Realization

    Given a set of specific users
     | username     |
     | me           |
     | admin        |
     | my_partner   |
    And a project "my project" created by "admin"
    And "me" contributes to "my project"
    And "my_partner" contributes to "my project"
    And "admin" exports "my project"`s graph
    When I log in as "me" "1234"
    And I navigate to the graph editor of "my project"
    And I create "Test object" "object" node in "me" swimlane
    And I realize "Test object" "object" node
    And I save
    Then I should see a save success message

  Scenario: Action Realization

    Given a set of specific users
     | username     |
     | me           |
     | admin        |
     | my_partner   |
    And a project "my project" created by "admin"
    And "me" contributes to "my project"
    And "my_partner" contributes to "my project"
    And "admin" exports "my project"`s graph
    When I log in as "me" "1234"
    And I navigate to the graph editor of "my project"
    And I create "Test action" "action" node in "me" swimlane
    And I realize "Test action" "action" node
    And I save
    Then I should see a save success message
```

Listing C.4: GE Realization Feature

```
Feature: GE-Export
  As a user
  I want to be able to align graph elements
  So that usability and layout is improved

  Scenario: Object Import

    Given a set of specific users
     | username     |
     | me           |
     | admin        |
     | my_partner   |
    And a project "my project" created by "admin"
    And "me" contributes to "my project"
    And "my_partner" contributes to "my project"
    And "admin" exports "my project"`s graph
    When I log in as "me" "1234"
    And I navigate to the graph editor of "my project"
    And I create "Test object" "object" node in "me" swimlane
    And I export "Test object" object node to "smthn"
    And I save
    Then I should see a save success message
    When I import "smthn" object node as "Imported Test object" in "
        my_partner" swimlane
    Then I should see "Importing smthn" "import action" node in "my_partner"
         swimlane
    And  I should see "Imported Test object" "object" node in "my_partner"
        swimlane
    When I save
    Then I should see a save success message

  Scenario: Objects Import

    Given a set of specific users
     | username     |
     | me           |
     | admin        |
     | my_partner   |
    And a project "my project" created by "admin"
    And "me" contributes to "my project"
    And "my_partner" contributes to "my project"
    And "admin" exports "my project"`s graph
    When I log in as "me" "1234"
    And I navigate to the graph editor of "my project"
    And I create "Test object" "object" node in "me" swimlane
    And I create "Test object 2" "object" node in "me" swimlane
    And I export "Test object" and "Test object 2" object nodes to "smthn"
    And I save
    Then I should see a save success message
    When I import "smthn" object node as "Imported Test object" in "
        my_partner" swimlane
```

```
48      Then I should see "Exporting Test object and Test object 2 into smthn" "
            external action" node in "me" swimlane
49      Then  I should see "smthn" "object" node in "me" swimlane
50      When I save
51      Then I should see a save success message
52
53  Scenario: Template Import
54
55      Given a set of specific users
56       | username      |
57       | me            |
58       | admin         |
59       | my_partner    |
60      And a project "my project" created by "admin"
61      And "me" contributes to "my project"
62      And "my_partner" contributes to "my project"
63      And "admin" exports "my project"`s graph
64      And "admin" sets "my project"`s graph as a template
65      And a project "second project" created by "admin"
66      And "me" contributes to "second project"
67      And "my_partner" contributes to "second project"
68      And "admin" exports "second project"`s graph
69      When I log in as "me" "1234"
70      And I navigate to the graph editor of "my project"
71      And I create "Test object" "object" node in "me" swimlane
72      And I create "Test object 2" "object" node in "my_partner" swimlane
73      And I save
74      And I navigate to the graph editor of "second project"
75      And I import "my project" template from "me" swimlane
76      Then I should see "Test object" "object" node in "me" swimlane
77      And  I should see "Test object 2" "object" node in "my_partner" swimlane
78      When I save
79      Then I should see a save success message
```

Listing C.5: GE Import Feature

```
1   Feature: GE-Export
2     As a user
3     I want to be able to align graph elements
4     So that usability and layout is improved
5
6     Scenario: Object Export
7
8       Given a set of specific users
9        | username      |
10       | me            |
11       | admin         |
12       | my_partner    |
13      And a project "my project" created by "admin"
14      And "me" contributes to "my project"
15      And "my_partner" contributes to "my project"
16      And "admin" exports "my project"`s graph
17      When I log in as "me" "1234"
18      And I navigate to the graph editor of "my project"
19      And I create "Test object" "object" node in "me" swimlane
```

```
    And I export "Test object" object node to "shared"
    Then I should see "Exporting Test object into shared" "external action"
        node in "me" swimlane
    Then I should see "shared" "object" node in "me" swimlane
    When I save
    Then I should see a save success message

  Scenario: Objects Export

    Given a set of specific users
     | username     |
     | me           |
     | admin        |
     | my_partner   |
    And a project "my project" created by "admin"
    And "me" contributes to "my project"
    And "my_partner" contributes to "my project"
    And "admin" exports "my project"`s graph
    When I log in as "me" "1234"
    And I navigate to the graph editor of "my project"
    And I create "Test object" "object" node in "me" swimlane
    And I create "Test object 2" "object" node in "me" swimlane
    And I export "Test object" and "Test object 2" object nodes to "shared"
    Then I should see "Exporting Test object and Test object 2 into shared"
        "external action" node in "me" swimlane
    Then I should see "shared" "object" node in "me" swimlane
    When I save
    Then I should see a save success message
```

Listing C.6: GE Export Feature

```
Feature: GE-Align
  As a user
  I want to be able to align graph elements
  So that usability and layout is improved

  Scenario: Align Top

    Given a set of specific users
     | username     |
     | me           |
     | admin        |
     | my_partner   |
    And a project "my project" created by "admin"
    And "me" contributes to "my project"
    And "my_partner" contributes to "my project"
    And "admin" exports "my project"`s graph
    When I log in as "me" "1234"
    And I navigate to the graph editor of "my project"
    And I create "Test object" "object" node in "me" swimlane
    And I create "Test object 2" "object" node in "my_partner" swimlane
    And I "align-top" nodes
     | name             |
     | Test object      |
     | Test object 2    |
```

```
25      Then nodes below should be "top alligned"
26       | name              |
27       | Test object       |
28       | Test object 2     |
29
30    Scenario: Align Bottom
31
32      Given a set of specific users
33       | username      |
34       | me            |
35       | admin         |
36       | my_partner    |
37      And a project "my project" created by "admin"
38      And "me" contributes to "my project"
39      And "my_partner" contributes to "my project"
40      And "admin" exports "my project"`s graph
41      When I log in as "me" "1234"
42      And I navigate to the graph editor of "my project"
43      And I create "Test object" "object" node in "me" swimlane
44      And I create "Test object 2" "object" node in "my_partner" swimlane
45      And I "align-bottom" nodes
46       | name              |
47       | Test object       |
48       | Test object 2     |
49      Then nodes below should be "bottom alligned"
50       | name              |
51       | Test object       |
52       | Test object 2     |
53
54    Scenario: Align Left
55
56      Given a set of specific users
57       | username      |
58       | me            |
59       | admin         |
60       | my_partner    |
61      And a project "my project" created by "admin"
62      And "me" contributes to "my project"
63      And "my_partner" contributes to "my project"
64      And "admin" exports "my project"`s graph
65      When I log in as "me" "1234"
66      And I navigate to the graph editor of "my project"
67      And I create "Test object" "object" node in "me" swimlane
68      And I create "Test object 2" "object" node in "me" swimlane
69      And I "align-left" nodes
70       | name              |
71       | Test object       |
72       | Test object 2     |
73      Then nodes below should be "left alligned"
74       | name              |
75       | Test object       |
76       | Test object 2     |
77
78
79    Scenario: Align Right
```

```
80
81      Given a set of specific users
82        | username     |
83        | me           |
84        | admin        |
85        | my_partner   |
86      And a project "my project" created by "admin"
87      And "me" contributes to "my project"
88      And "my_partner" contributes to "my project"
89      And "admin" exports "my project"`s graph
90      When I log in as "me" "1234"
91      And I navigate to the graph editor of "my project"
92      And I create "Test object" "object" node in "me" swimlane
93      And I create "Test object 2" "object" node in "me" swimlane
94      And I "align-right" nodes
95        | name             |
96        | Test object      |
97        | Test object 2    |
98      Then nodes below should be "right alligned"
99        | name             |
100       | Test object      |
101       | Test object 2    |
102
103   Scenario: Align Center X
104
105     Given a set of specific users
106       | username     |
107       | me           |
108       | admin        |
109       | my_partner   |
110     And a project "my project" created by "admin"
111     And "me" contributes to "my project"
112     And "my_partner" contributes to "my project"
113     And "admin" exports "my project"`s graph
114     When I log in as "me" "1234"
115     And I navigate to the graph editor of "my project"
116     And I create "Test object" "object" node in "me" swimlane
117     And I create "Test object 2" "object" node in "me" swimlane
118     And I "align-center-x" nodes
119       | name             |
120       | Test object      |
121       | Test object 2    |
122     Then nodes below should be "center-x alligned"
123       | name             |
124       | Test object      |
125       | Test object 2    |
126
127   Scenario: Align Center Y
128
129     Given a set of specific users
130       | username     |
131       | me           |
132       | admin        |
133       | my_partner   |
134     And a project "my project" created by "admin"
```

```
    And "me" contributes to "my project"
    And "my_partner" contributes to "my project"
    And "admin" exports "my project"`s graph
    When I log in as "me" "1234"
    And I navigate to the graph editor of "my project"
    And I create "Test object" "object" node in "me" swimlane
    And I create "Test object 2" "object" node in "me" swimlane
    And I "align-center-y" nodes
     | name              |
     | Test object       |
     | Test object 2     |
    Then nodes below should be "center-y alligned"
     | name              |
     | Test object       |
     | Test object 2     |

  Scenario: Distribute Vertically

    Given a set of specific users
     | username      | first_name                | email                     |
         password |
     | me            | Ioannis Oikonomidis | s083928@student.dtu.dk  | 1234 |
     | admin             | admin                   | admin@admin.dk
         | admin |
     | my_partner    | John Econ             | johnnyecon@gmail.com        |
         1234 |
    And a project "my project" created by "admin"
    And "me" contributes to "my project"
    And "my_partner" contributes to "my project"
    And "admin" exports "my project"`s graph
    When I log in as "me" "1234"
    And I navigate to the graph editor of "my project"
    And I create "Test object" "object" node in "me" swimlane
    And I create "Test object 2" "object" node in "me" swimlane
    And I create "Test object 3" "object" node in "me" swimlane
    And I "distribute-vertically" nodes
     | name              |
     | Test object       |
     | Test object 2     |
     | Test object 3     |
    Then nodes below should be "vertically distributed"
     | name              |
     | Test object       |
     | Test object 2     |
     | Test object 3     |
```

Listing C.7: GE Align Feature

```
Feature: GE-Grid
  As a user
  I want to be able to configure grid options
  So that usability and layout is improved

  Scenario: Enabling Snapping

```

```
8     Given a set of specific users
9      | username     |
10     | me           |
11     | admin        |
12     | my_partner   |
13    And a project "my project" created by "admin"
14    And "me" contributes to "my project"
15    And "my_partner" contributes to "my project"
16    And "admin" exports "my project"`s graph
17    When I log in as "me" "1234"
18    And I navigate to the graph editor of "my project"
19    And I toogle "snapping"
20    Then "snapping" should be "enabled"
21    When I toogle "snapping"
22    Then "snapping" should be "disabled"
23
24  Scenario: Enabling Grid
25
26    Given a set of specific users
27     | username     |
28     | me           |
29     | admin        |
30     | my_partner   |
31    And a project "my project" created by "admin"
32    And "me" contributes to "my project"
33    And "my_partner" contributes to "my project"
34    And "admin" exports "my project"`s graph
35    When I log in as "me" "1234"
36    And I navigate to the graph editor of "my project"
37    And I toogle "grid"
38    Then "grid" should be "enabled"
39    When I toogle "grid"
40    Then "grid" should be "disabled"
```

Listing C.8: GE Grid Feature

Feature	Scenario	Result
Auth - Login	Successful sign up	PASS
Auth - Login	Failing sign up	PASS
Auth - Login	Successful login	PASS
Auth - Login	Failed login	PASS
GE - Creation	Object Creation	PASS
GE - Creation	Action Creation	PASS
GE - Creation	Comment Creation	PASS
GE - Editing	Future Object Edit	PASS
GE - Editing	Future Action Edit	PASS
GE - Editing	Present Object Edit	PASS
GE - Editing	Present Action Edit	PASS
GE - Editing	Past Object Edit	PASS
GE - Editing	Past Action Edit	PASS
GE - Editing	Swimlane Edit	PASS
GE - Editing	Comment Edit	PASS
GE - Realization	Object Realization	PASS
GE - Realization	Action Realization	PASS
GE - Synchronization	Realization of Synchronized Nodes	PASS
GE - Import/Export	Object Export	PASS
GE - Import/Export	Objects Export	PASS
GE - Import/Export	Graph Template Import	PASS
GE - Align	Left Align	PASS
GE - Align	Right Align	PASS
GE - Align	Top Align	PASS
GE - Align	Bottom Align	PASS
GE - Align	Align Center X	PASS
GE - Align	Align Center Y	PASS
GE - Align	Distribute Vertically	PASS
GE - Grid	Snapping	PASS

Table C.1: Functional Test Cases

Bibliography

[BR14] Jens Engel Benno Rice Richard Jones. *Behave For Python*. 2014. URL: http://pythonhosted.org/behave/philosophy.html (visited on 06/15/2015).

[Cal13] Justin Callison. *Database Locking: What it is, Why it Matters and What to do About it*. Tech. rep. Methods Tools, 2013. URL: http://www.methodsandtools.com/archive/archive.php?id=83 (visited on 06/15/2015).

[Fou15] Django Software Foundation. *Django Models*. 2015. URL: https://docs.djangoproject.com/en/1.7/topics/db/models/ (visited on 06/15/2015).

[Gri12] Kurt Griffiths. *uWSGI vs. Gunicorn*. 2012. URL: http://blog.kgriffs.com/2012/12/18/uwsgi-vs-gunicorn-vs-node-benchmarks.html (visited on 06/15/2015).

[Hat15] Red Hat. *Advantages Of Using Docker*. 2015. URL: http://red.ht/1FXaqGL (visited on 06/15/2015).

[LTD15a] JGraph LTD. *JGraph Forum*. 2015. URL: http://forum.nwoods.com/c/gojs (visited on 06/15/2015).

[LTD15b] JGraph LTD. *JGraph Samples*. 2015. URL: http://gojs.net/latest/samples/index.html (visited on 06/15/2015).

[LTD15c] JGraph LTD. *JGraph Strategy*. 2015. URL: https://www.jgraph.com/about-jgraph-company.html (visited on 06/15/2015).

[MY03] Rahat Iqbal Muhammad Younas. *Developing Collaborative Editing Applications using Web Services*. 2003. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.4480&rep=rep1&type=pdf (visited on 06/15/2015).

[OH08] Yael Dubinsky Orit Hazzan. *Agile Software Engineering*. 2008.

[SS96] Neil Lang Sally Shlaer. *Shlaer-Mellor Method: The OOA96 Report*. 1996. URL: http://ooatool.com/docs/OOA96.pdf (visited on 06/15/2015).

[Tea12] RequireJS Development Core Team. *uWSGI vs. Gunicorn*. 2012. URL: http://requirejs.org/docs/optimization.html (visited on 06/15/2015).

[Tea15] Docker Development Core Team. *Dockerfile Reference*. 2015. URL: https://docs.docker.com/reference/builder/ (visited on 06/15/2015).

[Tez15] O.S. Tezer. *Docker Explained: Using Dockerfiles to Automate Building of Images*. 2015. URL: https://www.digitalocean.com/community/tutorials/docker-explained-using-dockerfiles-to-automate-building-of-images (visited on 06/15/2015).