Web Platform Development 2 - Coursework

(M3I322955)

Team R



A Web Application by

Chris Connor, Daryl McAllister, Gavin Macleod & Niklas Olsson

Module Leader: Katrin Hartmann

|  |  |  |
| --- | --- | --- |
| **Name** | **SID** | **Email** |
| Chris Connor |  | cconno208@caledonian.ac.uk |
| Daryl McAllister | S1222204 | dmcall200@caledonian.ac.uk |
| Gavin Macleod | S1715408 | gmacle204@caledonian.ac.uk |
| Niklas Olsson | S1434184 | nolsso200@caledonian.ac.uk |

“I declare that all work submitted for this coursework is the work of Chris Connor, Daryl McAllister, Gavin Macleod and Niklas Olsson alone unless stated otherwise.”

Contents

Introduction 4

Design 5

Wireframes 5

UML 8

Servlet Structure 9

Mapping of Links to Functionality 10

Persistence Including the Database Schema 13

The Design of the Share Link Functionality 15

How it works 15

Generating the link hash 15

URL Scheme 15

Application Security Appraisal 16

Test Reports 18

Unit Tests 18

LinkDAO 18

UserDAO 19

ProjectDAO 20

MilestoneDAO 21

ProjectTest 22

MilestoneTest 22

Intergration Testing 23

Future Considerations 24

Video Walkthrough 24

Conclusion 25

Appendix A: Testing Logs 26

[Figure 1 Servlet mapping and url mapping 7](file:////Users/chrisconnor/OneDrive%20-%20GLASGOW%20CALEDONIAN%20UNIVERSITY/WPD2/REPORTS/GroupReport-WPD2.docx#_Toc512702168)

[Figure 2 Original URL Scheme Design 8](#_Toc512702169)

[Figure 3 Database Entity Relationship Diagram 9](#_Toc512702170)

[Figure 4 Activity Diagram - Generating a shareable link 11](file:////Users/chrisconnor/OneDrive%20-%20GLASGOW%20CALEDONIAN%20UNIVERSITY/WPD2/REPORTS/GroupReport-WPD2.docx#_Toc512702171)

# Introduction

During our Web Platform Development 2 module, we were tasked with creating a milestone planner solution, which would allow users to create a project and add a list of milestones to the project. We worked in a team of 4, sharing the work load equally, making using of GIT version control, pushing to a remote BitBucket repository.

The functional requirements of the application are as follows:

* A user must be able to create an account
* A user must be able to login to their account
* Users must be able to log-out of their account
* Users should be able to create, edit and delete projects
* Users should be able to create, edit and delete milestones within a project

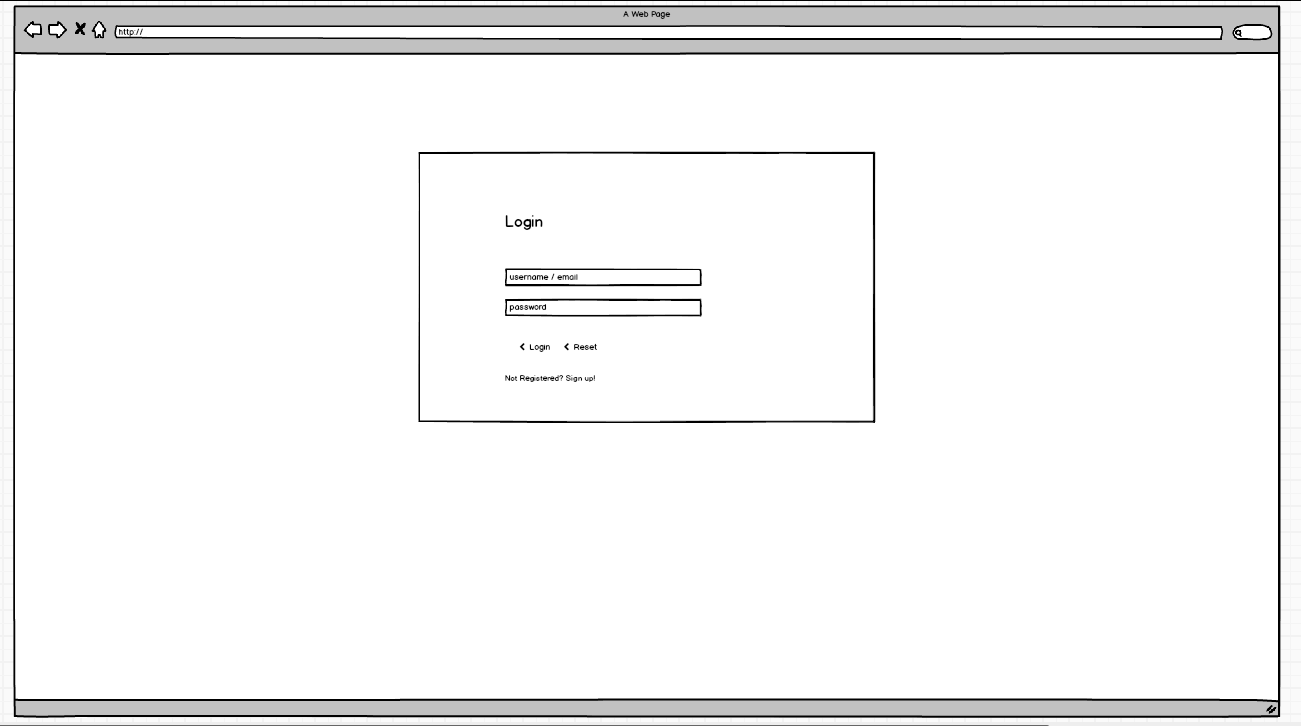
This report will discuss the team's work and the process of designing and development the application as well as features and functionality.

# Design

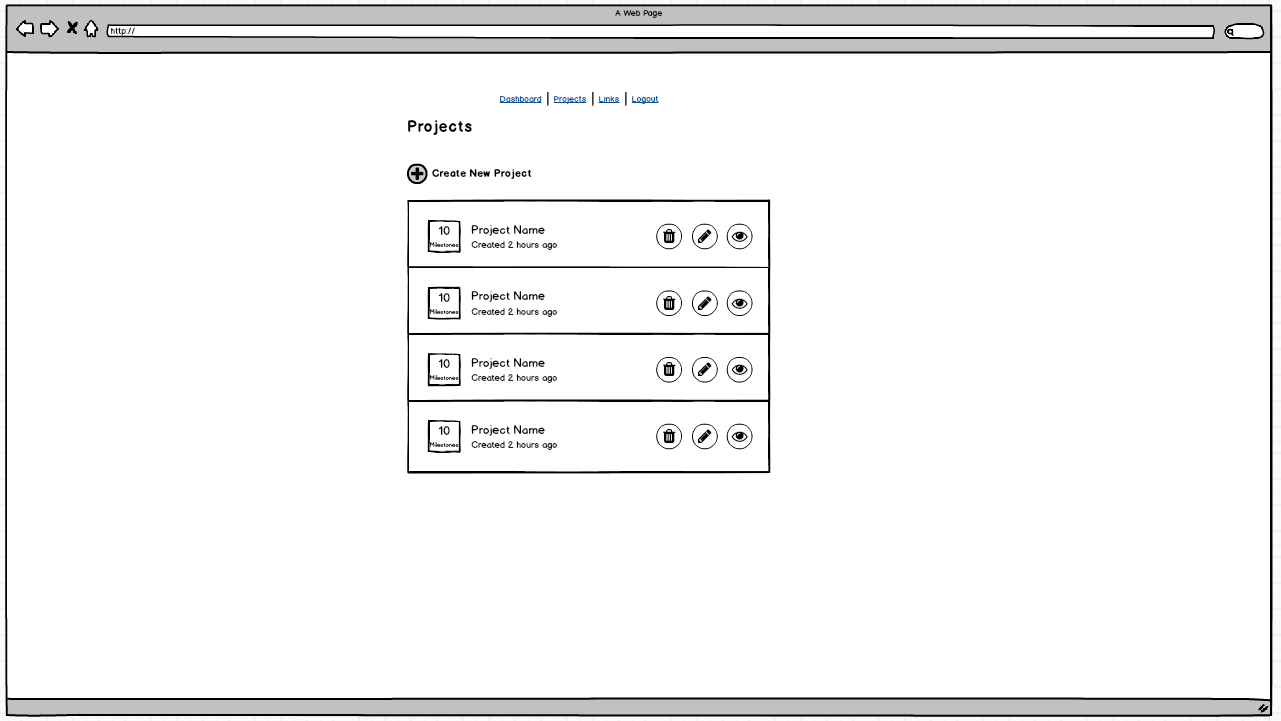
## Wireframes

The following images are the initial wireframe designs that were created at the start of the project to help envision what the final deliverable of the application will be. These designs were able to help map out what functionality will be needed and at what point they must be available. This was particularly useful when coming to figure out how the URL routing schema should be structured and what servlets would be needed to provide this functionality to the user. These designs were also used as a base template for how the web pages would be structured, in terms of were icons should be positioned and located on the page, and how they should be able to reflect operations. CSS styling and overall page aesthetics were later created and implemented in Mustache templates.

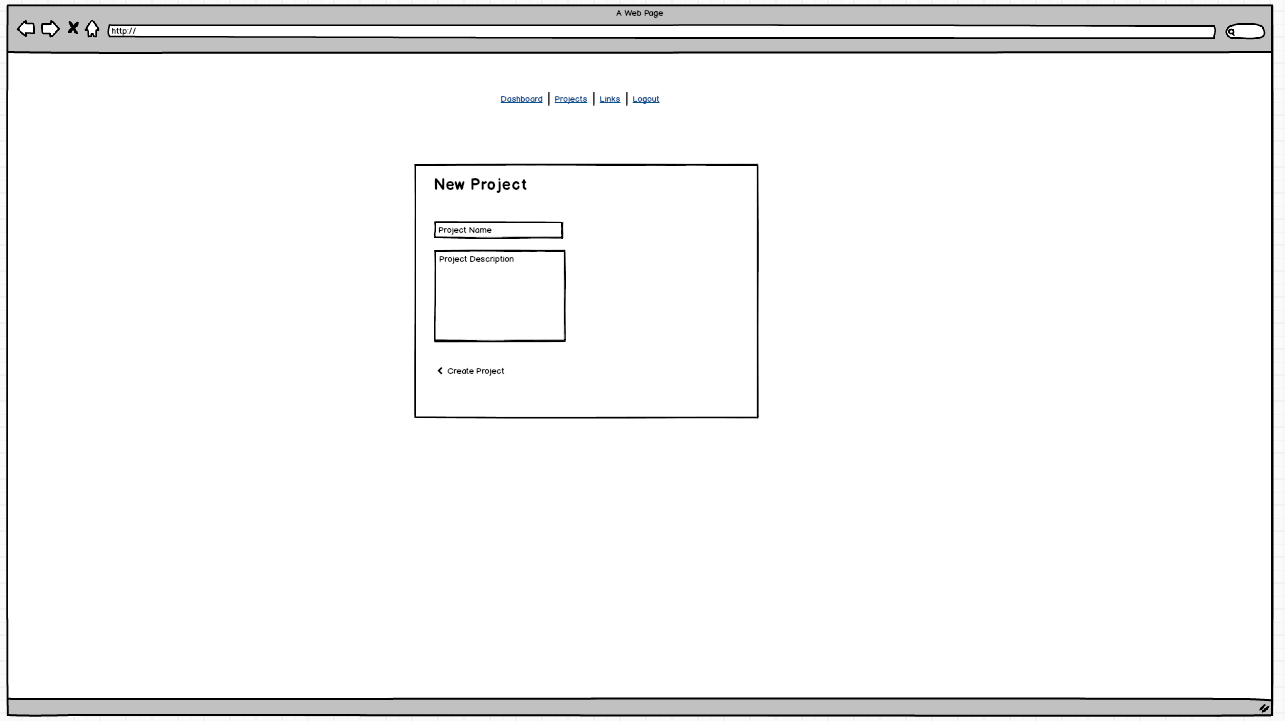
Login screen



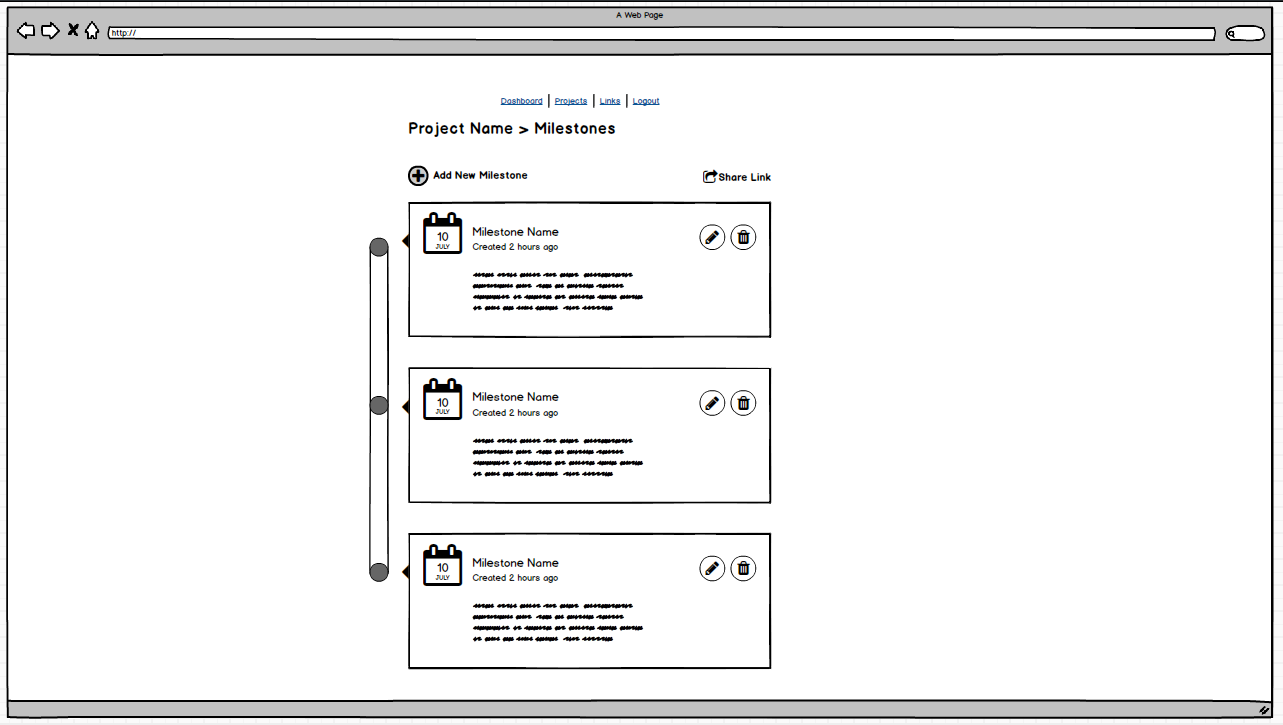
Project List



Create form



Milestone List

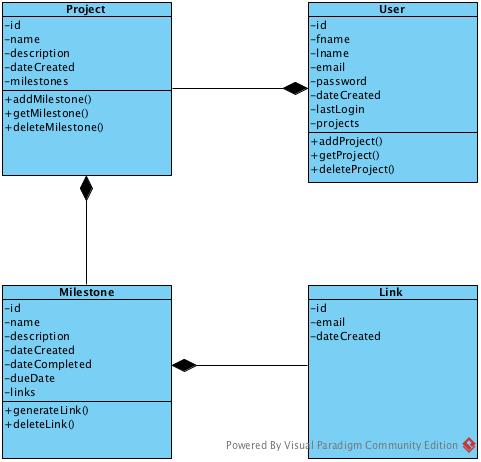


## UML

UML was used in the design of the system to get a grasp of the potential classes, functionality and overall structure of the backend of the system. The design of the UML for the system allowed the team to get a much deeper picture of how the system will function behind and gave light to the design of the front end of the system as this must work in tandem with the back end.

The main UML diagram that we produced was the class diagram which we spent sufficient time on to develop a solid class structure which was important as it would build up the base of our interactions in our system. The production of the class diagram also allowed for the entity relationships of our database to be visualized and aided in the design of it.

Class Diagram



## Servlet Structure

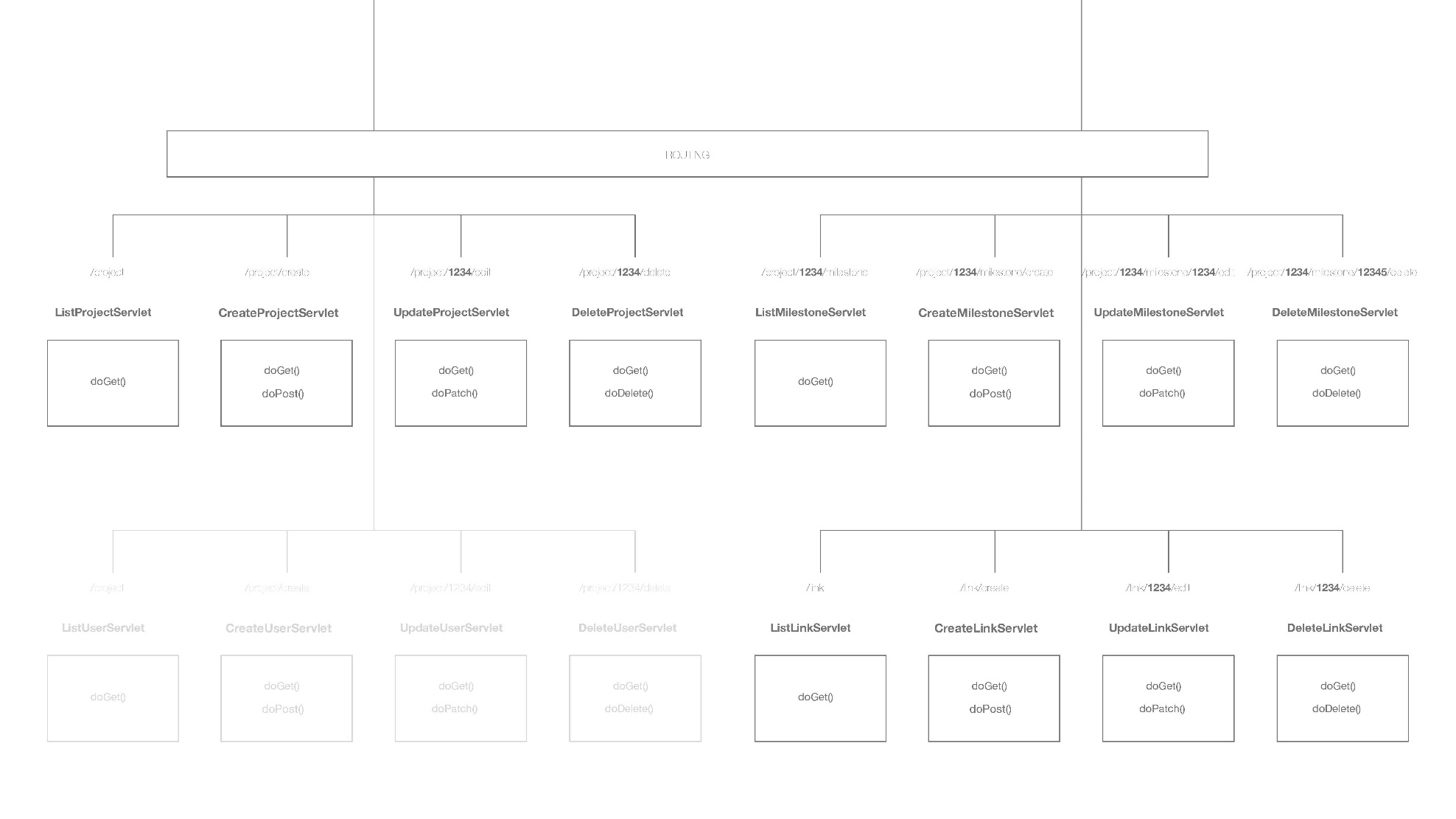


Figure Servlet mapping and url mapping

# Mapping of Links to Functionality

The initial design for the URL schema was to follow a RESTful pattern which would provide us with a familiar URL layout which is readable and consistent across the entire system. This provides our users with a clear and predictable URL with which to interact with the system giving a clear of actions and entities within system.

This approach was also designed with security in mind, intentionally not divulging personal data in the URL, and as little system information as possible to ensure that malicious attempts to exploit vulnerabilities would be much harder and not possible. These benefits allow for a solid and robust set of linking features that provide the ability to have expansion within our application due to the simple layout of the separation of functions.

This initial schema in the end was not possible to implement due to the wild card restrictions imposed on us by the Java Servlet Specification. This meant our intended routing structure had to be rethought and reworked. The team wished to find a solution that provided as much as the initial benefits as possible as we felt these where largely necessary to provide users with a robust and uniform experience even with something that is often as overlooked like the URL design.

Initial schema

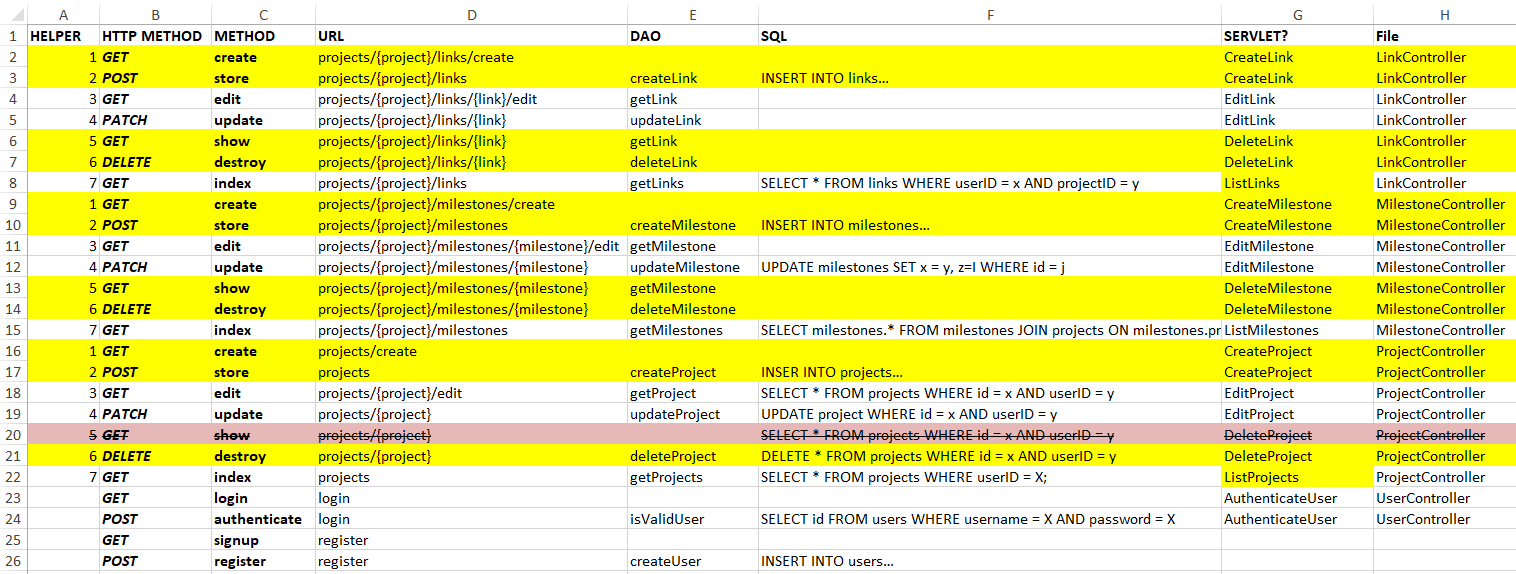


Figure 2 Original URL Scheme Design

The revised scheme (provided below) keeps a large amount of the structure of the original but contains the wildcard element at the end of the URL. This change dictated a rethink of how our data was gained from the URL in the system as well. While less ideal than the first initial approach we felt the implementation and use of this approach worked nearly as well and gave the same benefits. The team managed to keep the URL's short and open so that further pages could be implemented easily without changing the site structure. Keeping the human readable format also was done by having clear ASCII only characters that have no complicated search terms or symbols that would confuse readers using it as we intended originally as well.

The new approach to mapping also dictated that we had a middle interaction when changing from dealing with projects to milestones. This was needed as we would keep the project that the milestones dealt with in a session variable rather in the URL for security and readability. This meant that we would have to implement a middle interaction which was in the form of a dispatch servlet. This dispatch servlet is called from the clicking of a project to view the milestones and in this servlet the session variable that contains the ProjectId is set to have a persistant track of the project the user is working in. Throughout further interactions within the project the sessionID is called and used to get the project from which the data is being viewed or manipulated, which eliminated the need to implement this data in the URL.

Final schema

|  |  |  |
| --- | --- | --- |
| URL | Servlet | HTTP METHODS |
| /projects | ProjectListServlet | GET |
| /projects/create | ProjectCreateServlet | GET / POST |
| /projects/delete/\* | ProjectDeleteServlet | GET / POST |
| /projects/edit/\* | ProjectEditServlet | GET / POST |
| /milestones/project/\* | MilestoneDispatcherServlet | GET |
| /milestones | MilestoneListServlet | GET |
| /milestones/create | MilestoneCreateServlet | GET / POST |
| /milestones/edit/\* | MilestoneEditServlet | GET / POST |
| /milestones/delete/\* | MilestoneDeleteServlet | GET / POST |
| /links | LinkListServlet | GET |
| /links/create | LinkCreateServlet | GET / POST |
| /links/delete/\* | LinkDeleteServlet | GET / POST |
| /shared/milestones | SharedMilestoneListServlet | GET |
| /shared/\* | SharedLoginServlet | GET / POST |
| /login | LoginServlet | GET / POST |
| /logout | LogoutServlet | GET |

# Persistence Including the Database Schema

For our web application we decided to use a MySQL database to persist the information in our model classes with DAOs for each model class. The tables contain all the information of models that are manipulated within our application and as well provide the logical relationships between them. The information we chose to persist was decided to be an appropriate level detail that a user would need when using this application, as well as relevant to the nature of milestone tracking.

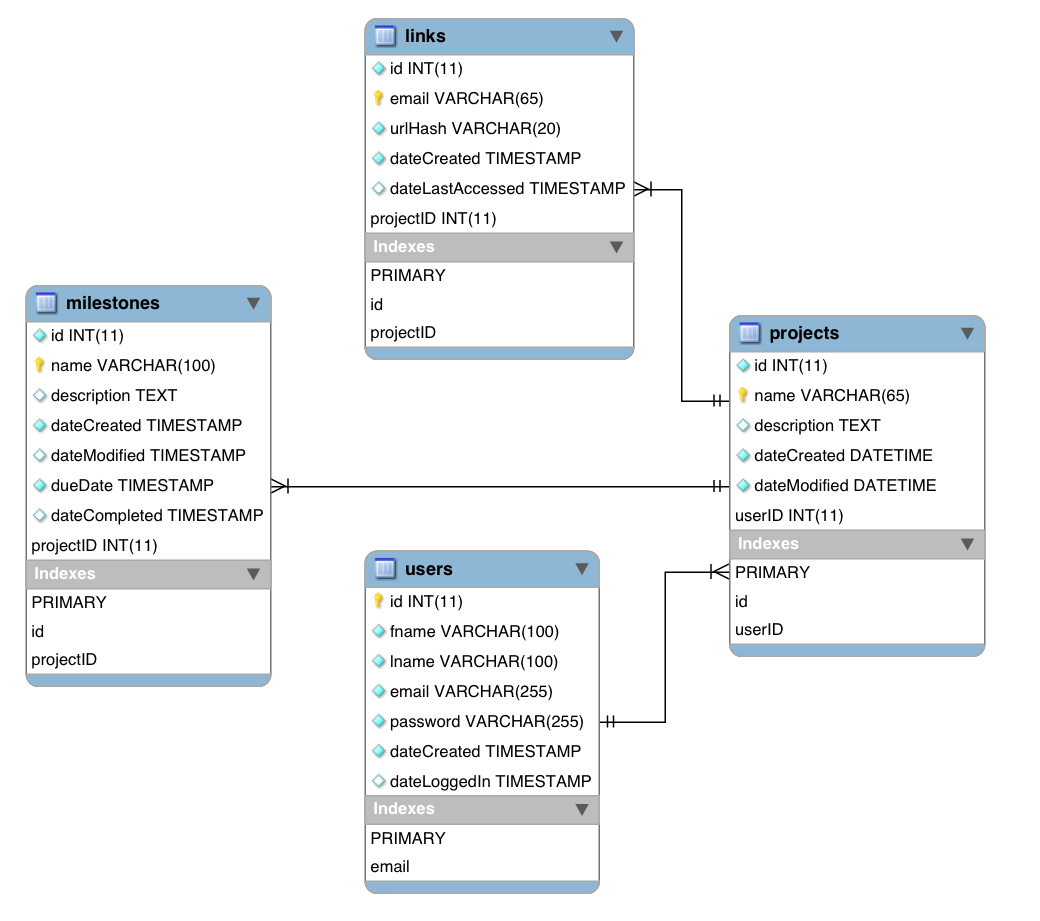


Figure 3 Database Entity Relationship Diagram

When designing our web application, we saw that the main four models that we would be interacting with would be users, projects, milestones, and links, and were able to quickly distinguish the relationship between these models. As shown in the class diagram, the models are depicted to have a has-a relationship, as well as their multiplicity, between each other, starting with Users having Projects, and then Projects having Milestones and having Links. To represent this relationship between the models we used sets of primary-keys (PK) and foreign-keys(FK) to bind the models to each other. The unique IDs of the Users were used as foreign-keys for Projects, and then ID of Projects were used as FK for Milestones and Links. We also choose to make User email a primary-key of users as a way to ensure the login details would be unique and method for users to identify themselves.

Our application is able to interact and query our database by using DAOs, in which we created a separate DAO for each model in order to logically group together SQL statements that query a specific table. For example, within the UserDAO, the two methods provided are to checkIsValidUser or registerUser. This was done to ensure that only users with the correct login credentials can use the application by querying the database to check whether the login credentials are true, else asking the user to register an account to then use to the application. The DAOs within the application, ProjectDAO, MilestoneDAO, and LinkDAO, all implement CRUD functions by SQL statements to get information that our application will use.

# The Design of the Share Link Functionality

One of the requirements of the project was to have the ability to create a link which could be shared in order to allow others to see user’s milestones. Given our design, and the way we were grouping milestones under a project, we decided to build the link functionality around projects – allowing a member to share their projects. For there – an external non-member user could view their project.

## How it works

When a user is logged in, they can access the “Manage Links” section of the application which will display all the links belonging to that user. From there – they can create and remove shared links within their account. This section also gives them the ability to automatically copy links into their clipboard with a single button click.

## Generating the link hash

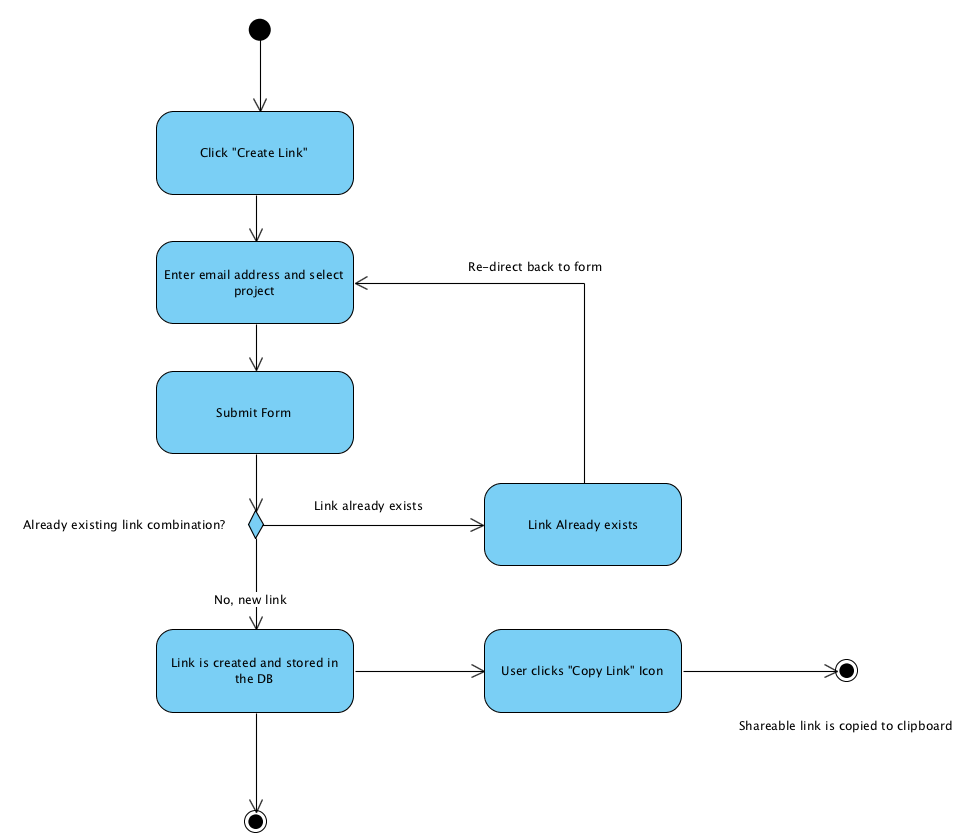
Each link has a sequential id in our database however, we did not want to feature this in our publicly shareable links. Due to the id’s being sequential, there could be potential for malicious users to guess link id’s which could ultimately result in data being compromised. We instead wanted to create a link with a small alphanumeric hash – similar to the approach that URL shortening services use – e.g. <https://tinyurl.com/y9kv2ja8> . This not only would look better, but if a user had to manually type this in to the address bar, it would be much easier than a long hash. Having researched online - there are literally hundreds of cryptographic algorithms and check sum generators out there – we settled on Adler32. The Adler32 checksum algorithm is part of the zlib library and can generate a 32-bit checksum – its original use was for validating unzipped data.

Figure Activity Diagram - Generating a shareable link

To create the final hash which is stored in our database, and which forms part of the generated link, we take the Link object and serialise it as a string. This string is passed through the Adler32 algorithm which generates the checksum. Finally, to ensure a random string which is fairly pleasing to the human eye, we convert this to hex which is then written as a string to the database column.

## URL Scheme

To keep the operation of this function extremely simple and user-friendly, the entry point for accessing a shared URL is simply “shared”, followed by the hashed URL string:

**/shared/{url-hash}**

# Application Security Appraisal

Security of web applications is a vital aspect to keep in mind, as there are many ways in which an individual can abuse system exploits or access unauthorised data. One of the most common vulnerabilities in web applications is not protecting web page input from SQL injections. This is when a user can input SQL statements straight into site input such as “enter name” text boxes, which will then be run on the database. While developing our application, we made sure to combat SQL injections by making use of ‘prepared statements’, this allows SQL statements that are used repeatedly to be executed more efficiently, as well as protect input from SQL injections. Prepared statements use unspecified values, which need to be set by passing parameters into each value. These values are parsed and transmitted at a later time, binding the values to the parameters using a different protocol. These parameter values don’t need to be correctly escaped, so SQL injections cannot happen as the original statement template isn’t derived from external input.

While preparing these statement templates in our CRUD methods, we made sure that the statement would only INSERT, UPDATE, GET and DELETE to a specific table by ensuring these actions were only done where a project was being inserted to the correct user ID, as well as ensuring a milestone was inserted to the correct project ID *and* the currently signed in user ID from the session. This would stop the ability for anyone to come in and access any project or milestone by simply changing the data in the URL.

When dealing with user accounts, we used salted password hashing techniques to keep user passwords secure in the database. Password hashing algorithms are one-way functions that change data into irreversible, fixed length fingerprints. This hash is then stored in the database and when a user tries to log in using their password, it is hashed and checked against the hash in the database when a user tries to log in. Salting a password provides additional level of security, by adding additional information to a password it can make it more difficult for attackers to guess the password using software that can run through multiple common password phrases or attacks like a dictionary attack.

Within our application we make use of sessions to keep track of where a user is within the application and to ensure they are able to perform the actions they wish to. However, sessions can be prone to interception or hijacking if the sessionID is exposed within the URL. To prevent against this, we built the applications functionality around on having users login to gain access to any part of the application, and check their login details in the database before assigning a session. We continually check the user is authenticated throughout the project which ensures a not logged in user cannot access pages they should not be able to. This added layer of security is common place on web applications to stop unauthorized use of sections.

Routeing in the program was specifically designed to ensure no personal date or sensitive information was visible to any user of the system. Security vulnerabilities can be numerous when data that is sensitive to a user or a system is passed through URL's, as this data can be seen by anyone and taken and then manipulated or used to gain access to restricted sections that would usually be closed off. To combat security vulnerabilities of this nature we set a clear and structured set of URLS that have a consistent layout without showing data where possible. Most of the data passed is done so using session variables and posted forms. These two methods provide a far more secure and non-transparent method of providing data to the server where users cannot see what is being passed easily.

While most security vulnerabilities of the system have been considered, there are still venerable parts of the system that the team are aware of and would fix in future iterations. One of these would be to make the login aspect more secure. Currently this method of login is open to bot login attempts as there is no limit to the amount of times a user may try and login. This opens this area up to brute force attacks which could be improved in the future by adding login maximum attempts and locking accounts. The system is also vulnerable with the current input checking as it is not fully robust against SQL Injection attacks. Many of the input fields have basic validation checking on the client side and the use of prepared statements on the server side. Even with these measures, there are certain types of newer SQL injection attacks that can still be done by circumventing these methods and implementing customer SQL in a system. For the future of this system, a deeper research and potential solution for these attacks would be required and the implemented to make the system as secure as possible.

# Test Reports

## Unit Tests

Throughout development several unit tests where generated to aid in testing parts of the code that were already written as well as providing a clear base and solid set of consisting test data that could be run when required to test the integration of new features or changes.

Implementation of the unit tests was accomplished using JUnit 4 which provided a robust set of methods and structure to be able to implement the automatic unit tests. Having this class already at our disposable meant that testing could be undertaken right away without complicated logic being deployed to test the systems progress. This further aided in development as certain sections of code that where implemented could be tested through unit tests without first having to code parts that interacted with it like would be required in traditional integration testing.

The following is a detailed list of all unit tests and their purpose:

### LinkDAO

Unit test file that tests the functionality of the methods of the LinkDAO that interact with the databse when concerning Links

Name: findById - LinkDAO

Description: Asserts true or false depending on whether the link object that was set up is null depending if it was populated by the findById method.

Method Tested: Link findById()

Pass: Passed – Link not null – asset true

Name: save - LinkDAO

Description: Asserts true if the newly created link can be saved to the database

Method Tested: save()

Pass: Passed - returned true – assert true

Name: findByProjectId - LinkDAO

Description: Asserts false depending on whether the list of links that was set up is not empty depending if it was populated by the findByProjectId method.

Method Tested: findByProjectId

Pass: Passed - not empty – assert false

Name: findByUserId - LinkDAO

Description: Assert whether the statement equals the supplied condition. Sets up a list of links and populates it using the findByUserId and asserts whether the size of it is 2

Method Tested: findByUserId

Pass: Passed - size is 2 – true condition

Name: findAll - LinkDAO

Description: Sets up a list of links variable and uses findAll to populate it. Condition checks weather the links is empty, and test completes if assert false

Method Tested: findAll()

Pass: Passed - not empty – assert false

Name: deleteByEmail - LinkDAO

Description: Asserts true if deleting a link by an email is sucessfull

Method Tested: deleteByEmail()

Pass: Passed - delete was successful – assert true

### UserDAO

Unit test file that tests the functionality of the methods of the UserDAO that interact with the databse when concerning Users

Name: registerUser- UserDAO

Description: Asserts true if the creation and saving of a user to the database is true

Method Tested: registerUser()

Pass: Passed - delete was successful – assert true

Name: checkIsVaidUser - UserDAO

Description: Asserts false if the checkIsValidUser returns empty meaning the check failed

Method Tested: checkIsValidUser()

Pass: Passed - check was not empty – assert false

Name: checkIsNotValidUser - UserDAO

Description: Assert true if the check of an invalid user is empty meaning the check failed

Method Tested: checkIsValidUser()

Pass: Passed - check was empty – assert true

### ProjectDAO

Unit test file that tests the functionality of the methods of the ProjectDAO that interact with the databse when concerning projects

Name: getProjectById – ProjectDAO

Description: Asserts true if the attempt of getting a project by a Id is not null.

Method Tested: getProjectById()

Pass: Passed – project returned was not null – assert true

Name: deleteProjectById– ProjectDAO

Description: Asserts false if the attempt at deleting a project by its ID is not successful

Method Tested: deleteProjectById()

Pass: Passed – Project could not be deleted – assert false

Name: getAllProjects– ProjectDAO

Description: Calls the method being tested to get a list of all Projects. Test will pass if the list that is returned is not null meaning that it has been populated

Method Tested: getAllProjects()

Pass: Passed – returned list was not null – assert not null

Name: getProjectsByUser– ProjectDAO

Description: Calls the getProjectByUser method to get a user's projects. The test will pass if the returned list is not empty

Method Tested: getProjectByUser()

Pass: Passed – list returned was notempty – assert true

Name: getProjectByIdAndUser – ProjectDAO

Description: Gets a single project by matching ID and user account. The test will assert true if the project returned is not null

Method Tested: getProjectByIdAndUser()

Pass: Passed – project returned was not null – assert true

Name: projectUpdate– ProjectDAO

Description: Sets a milestone with data that will override an existing milestone in the database. The test will assert true if the update is successful

Method Tested: projectUpdate()

Pass: Passed – update was successfull – assert true

### MilestoneDAO

Unit test file that tests the functionality of the methods of the MilestoneDAO that interact with the database when concerning milestones

Name: createMilestone– MilestonetDAO

Description: Asserts true if the newly initialized milestone object is successfully inserted into the database

Method Tested: CreateMilestone()

Pass: Passed – milestone created – assert true

Name: getMilestoneByIdAndUser– MilestonetDAO

Description: Attempts to get a milestone using the supplied user email and Id. Test passes if the milestone returned is null.

Method Tested: getMilestoneByIdAndUser()

Pass: Passed – milestone not null – assert false

Name: getMilestonesById– MilestonetDAO

Description: Asserts false if the returned milestone from getMilestoneById does not match the Id or is returned null

Method Tested: getMilestonesById()

Pass: Passed – milestone not null – assert false

Name: getAllMilestonesByProjectAndUser– MilestonetDAO

Description: Attempts to get a all the milestones that relate to a project and the current user. The test passes if the milestone list is not null.

Method Tested: getAllMilestonesByProjectAndUser()

Pass: Passed – list not null – assert false

Name: getAllMilestonesByProjectId– MilestonetDAO

Description: Attempts to get all the projects that containe a single project ID. Test passes if the list is not empty

Method Tested: getAllMilestonesByProjectId()

Pass: Passed – list not empty – assert true

### ProjectTest

Unit test file that tests the functionality of methods of a Project base class

Name: getPrettyDateCreated– ProjectTest

Description: Creates a prettyDate string of the creation date in a project. Test is passed if the string returned equals the expected output.

Method Tested: getPrettyDateCreated()

Pass: Passed – strings match – assert equals

Name: getPrettyDateModified– ProjectTest

Description: Creates a prettyDate string of the modified date in a project. Test is passed if the string returned equals the expected output.

Method Tested: getPrettyDateCreated()

Pass: Passed – strings match – assert equals

### MilestoneTest

Unit tests that test the functionality of the methods of a milestone base class.

Name: getPrettyDateCreated– MilestoneTest

Description: Creates a prettyDate string of the creation date in a milestone. Test is passed if the string returned equals the expected output.

Method Tested: getPrettyDateCreated()

Pass: Passed – strings match – assert equals

Name: getPrettyDateModified– MilestoneTest

Description: Creates a prettydate string of the modifieddate in a project. Test is passed if the string returned equals the expected output.

Method Tested: getPrettyDateModified()

Pass: Passed – strings match – assert equals

Name: getDueDate– MilestoneTest

Description: gets the dueDate for a project from the supplied information and the test passes if the output assert equals the expected output

Method Tested: getDueDate()

Pass: Passed – strings match – assert equals

## Intergration Testing

SEE APPENDIX

# Future Considerations

Considering future possibilities, there are a few features and extra functionality that we discussed about implementing. One of these would be an admin page, where a system administrator would have their own login with access to all users, projects and milestones. Several DAO classes we implemented already could support these features, as we included methods for this possibility, but they are not currently used.

There are a few features that our system lacks in terms of user accounts, this includes no limit to login attempts, and no email verification. These are commonly used features and it is good practice to have these in a web application, however it was low priority and did not get implemented.

# Video Walkthrough

Below is a video walkthrough of the system in action.



<https://www.youtube.com/watch?v=zSgXbD4ZalE>

# Conclusion

Overall, we believe this application to have met all the requirements necessary and is able to perform the functionality we had designed initially in the beginning stages of this project. The functionality has been tested to perform as expected, by the means of unit testing and functionality testing, and has been translated appropriately to the interface that is able to be served from a collection of Java servlets. We were able as well to provide URL routing that is logical and provides clear indication as to the position of where a user is within the application and use this to provide functionality that is useful and logical. The persistence method we implemented can perform CRUD functionality that is protected against SQL injection and structured in a way that can provide information that is of use to our application and all functionality has been tested rigorously and pass their respective test cases.

To conclude this report, we believe the implemented solution has met the functional requirements and followed the envisioned design. Considering this, we believe that the project was successful in producing what was required to be built, a milestone tracking web application. The team worked well together and communicated effectively, with no real controversy or overlapping work.

# Appendix A: Testing Logs