CS673F14P4 Software Engineering
Group 4 Project - ProManage
Software Design Document

Your project Logo here if any

Team Member	Role(s)	Signature	Date
Luis Marion	Project Leader	LM	10/14/2014
Daniel Abramowitz	Lead Designer	DA	10/15/2014

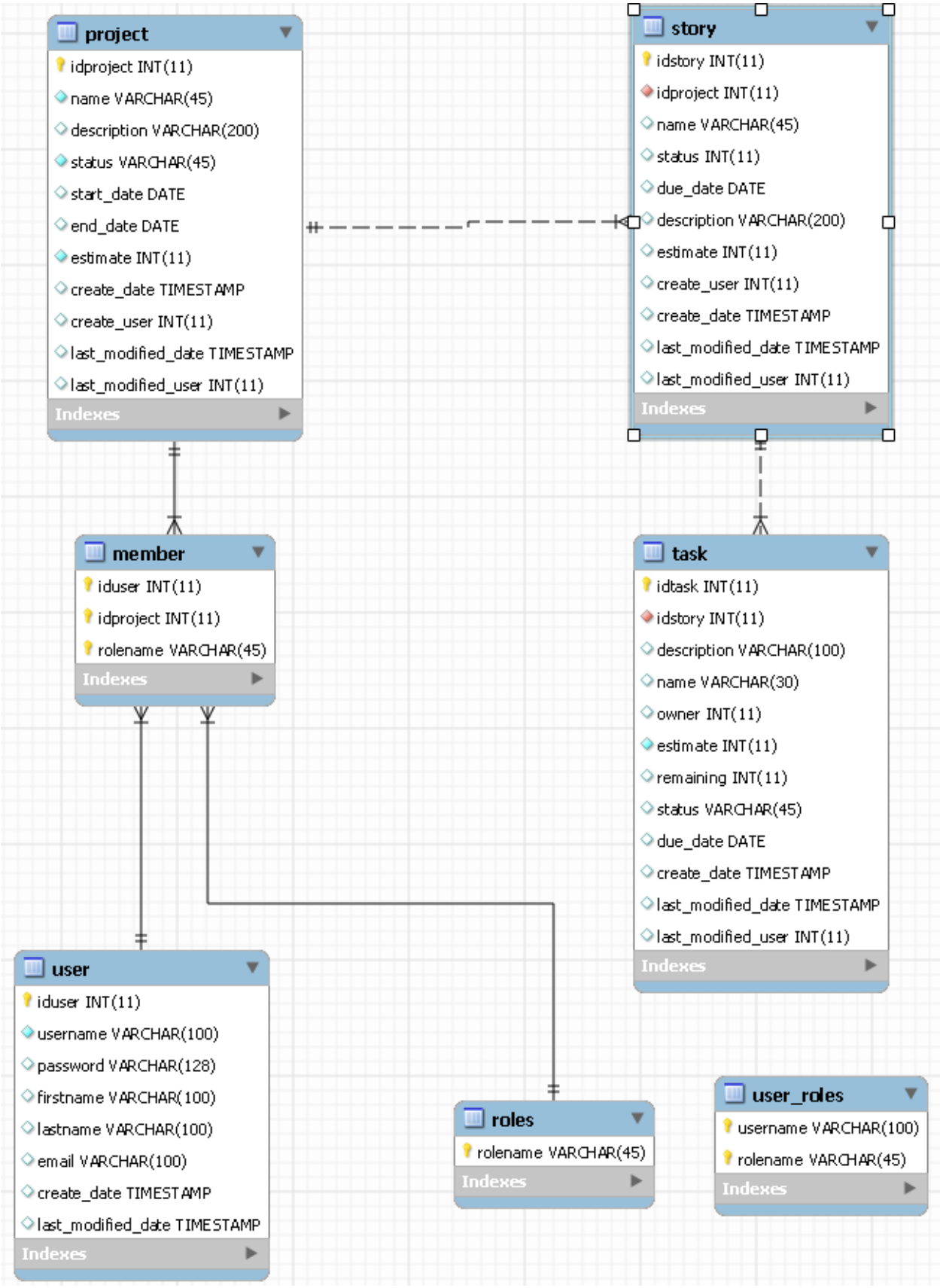
Revision history

Version	Author	Date	Change
1	Luis Marion	10/14/2014	Initial version
2	Luis Marion	11/06/2014	Updated data model. Expanded details on code structure as well as a short data model description. The updated datamodel was generated using mysql-workbench and should be more legible.
3	Luis Marion	12/7/2014	Updated data model. Re-arranged some of the details to move package structure under Classes. Gave summary of package rather than describing every single class.

1. Introduction

ProMange is a web-application for managing and tracking Agile based software development projects. The application uses an HTML5/CSS/JQuery based front-end in order to communicate with the server through AJAX restful service calls. The application enables users to create projects, invite members to projects, and create and assign stories and tasks to those members. The application is hosted on a Tomcat 7.0 server using a MyBATIS SQL database for persistence. The application is built using Java (jdk1.6).

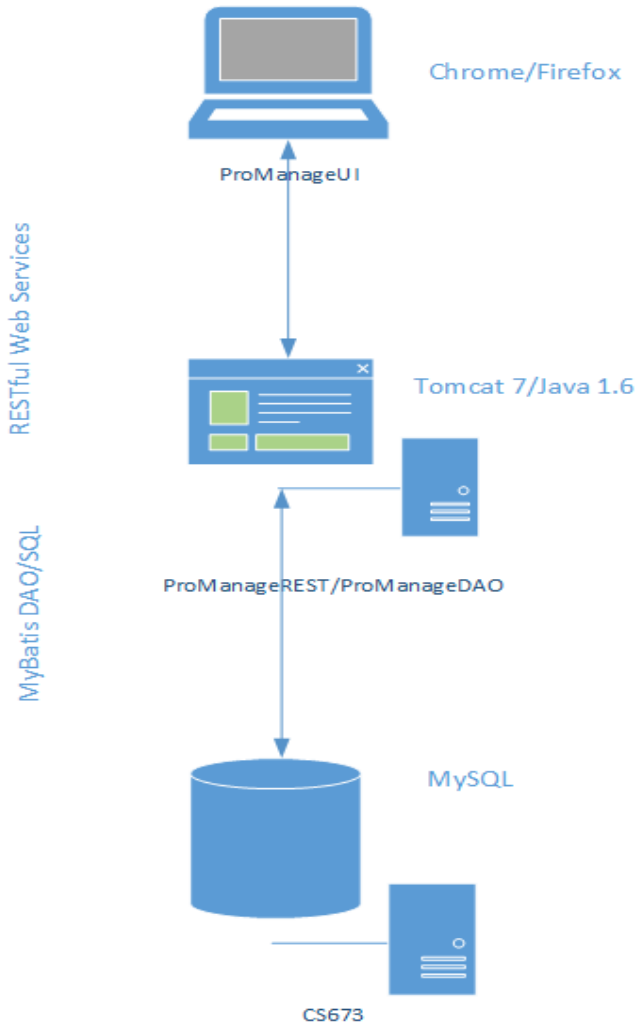
2. Software Architecture


project
idproject INT(11)
name VARCHAR(45)
description VARCHAR(200)
status VARCHAR(45)
start_date DATE
end_date DATE
estimate INT(11)
create_date TIMESTAMP
create_user INT(11)
last_modified_date TIMESTAMP
last_modified_user INT(11)
Indexes
story
idstory INT(11)
idproject INT(11)
name VARCHAR(45)
status INT(11)
due_date DATE
description VARCHAR(200)
estimate INT(11)
create_user INT(11)
create_date TIMESTAMP
last_modified_date TIMESTAMP
last_modified_user INT(11)
Indexes
member
iduser INT(11)
idproject INT(11)
rolename VARCHAR(45)
Indexes
task
idtask INT(11)
idstory INT(11)
description VARCHAR(100)
name VARCHAR(30)
owner INT(11)
estimate INT(11)
remaining INT(11)
status VARCHAR(45)
due_date DATE
create_date TIMESTAMP
last_modified_date TIMESTAMP
last_modified_user INT(11)
Indexes
user
iduser INT(11)
username VARCHAR(100)
password VARCHAR(128)
firstname VARCHAR(100)
lastname VARCHAR(100)
email VARCHAR(100)
create_date TIMESTAMP
last_modified_date TIMESTAMP
Indexes
roles
rolename VARCHAR(45)
Indexes
user_roles
username VARCHAR(100)
rolename VARCHAR(45)
Indexes

The data model consists of four primary entities.

User - the user that can access the system.
Project - the projects that each user is associated to.
Story - the different work items necessary in order to complete a project.
Task - the different work items necessary in order to complete a story.

The member table is the relationship table between users and their respective projects. It is a many to many relation, as 1 user can belong to many projects, and 1 project will have many users associated to it. The role relation is in place since users can be associated to projects as either the leader or a member. As of this moment, the application does not authorize any differently with the member/leader distinction, but its in place for future enhancements.

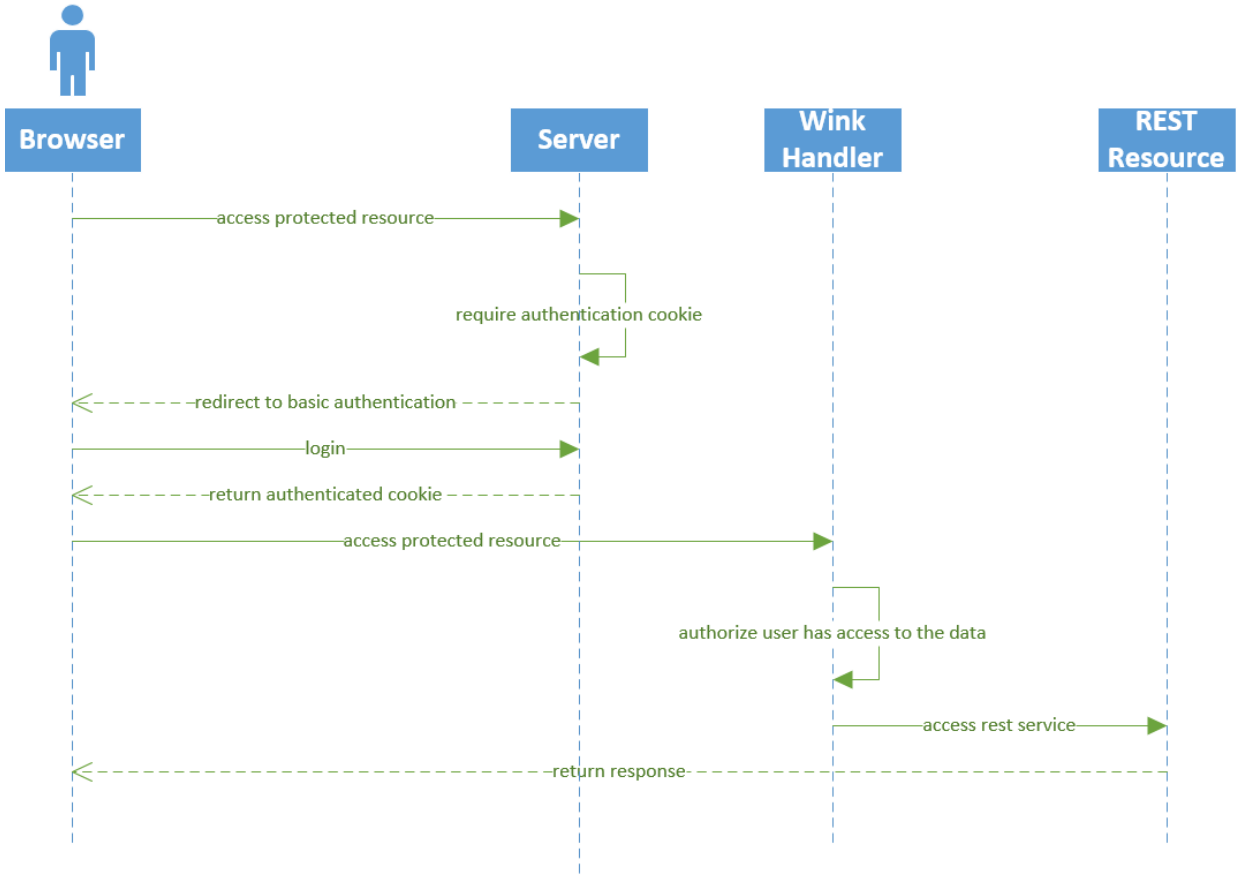


ProManage application uses a tiered application model. The application uses three tiers each with one project to represent the corresponding iter.

ProMangeUI - presentation tier - this is the front-end of the application. It is HTML5 and JQuery/AngularJS front-end using RESTful services and JSON to retrieve and update data on the back-end. The ui uses ajax to make http-service requests to the rest application living on the Tomcat server. This used to be its own project but was removed and all of the relevant ui files have been moved under the WebContent dir in ProManageREST. This was to simplify the deploy process and solve the cross-site scripting issue that occurs when making ajax calls to an external domain.

ProManageREST - business logic tier - the RESTful interface exposes the business logic and persistence layer to the ui and acts as the broker between presentation and persistence. Any business logic will live in this tier. The REST project is also responsible for mapping between the DTO objects of the persistence layer into the presentation tier model. The project can be decomposed into three basic packages. The RESTful interfaces are secured using a combination of Tomcat container-managed security as well as wink handlers for data authorization.

ProManageDAO - persistence tier - this project is strictly responsible for database access. These classes perform the basic CRUD (Create, Retrieve, Update and Delete) operations to a MySQL database. The database access is facilitated using mybatis framework.



Application security can be broken out into two components authentication and authorization. The ProManage application leverages container-managed security in order to perform authentication and basic authorization. We use the term “basic” to mean distinguishing users from system admins and customers. Any user that registers for the application will automatically be entered in as a customer. This strictly enables the user past the “basic” authentication and authorization performed by tomcat. In order to perform data authorization for customers based on their projects the application uses wink-handlers. This is the authorization the application uses to control who can modify projects, stories, and tasks.

A wink-handler is basically just a servlet filter. It intercepts an HTTP request prior to reaching the REST resource (web-service call) and after performing some data validations, it either forwards the request to the intended destination or returns an unauthorized access error to the browser.

The filter is responsible for making sure that users are only performing actions against projects/stories/tasks to which they are associated to. This authorization takes place in the handlers and security.impl packages.

3. Design Patterns

Singleton - singleton pattern is used in order to retrieve the database session (connection). The SqlSessionFactory is recommended to be generated by a singleton through the mybatis framework.
DAO - the data access layer uses data access objects in order to control all of the database related operations.

4. Key Algorithms

NA. There is one algorithm we took from a website for hashing the users password using SHA-512. This seemed trivial compared to the rest of the project and the code has the website for reference (EncryptUtil.java).

5. Classes and Methods

bu.met.cs.cs673.pm.jaxrs.model - this package contains all of the REST model objects. These are the model objects that the REST tier interfaces with and that the javascript ui leverages. These objects are seamlessly converted from pojo to json and back using the wink-json-provider. The jaxb annotation at the top of each class is what tells the apache-wink framework that this is a jaxb pojo to be converted to and from json.

bu.met.cs.cs673.pm.jaxrs.mapper - this package contains classes responsible for mapping between the dto (dao model) objects and the rest model objects.

bu.met.cs.cs673.pm.jaxrs.resource - this package contains all of the restful interfaces. This is the entry point to the web-service based application and each resource supports basic crud operations mapped to the http-verbs, as well as additional methods to support day to day application usage.

bu.met.cs.cs673.pm.dao - the dao package performs the majority of the work. It houses all of the basic CRUD sql operations. It leverages the mybatis framework in order to simplify and organize the coding. In essence mybatis uses the SqlSessionFactory (SessionFactorySingleton.java) in order to read a basic configuration file (mybatis-local-config.xml) to establish a database connection. The same config file also contains references to the independent sql map files. These sqlmap files (project.xml, user.xml, etc.) basically serve to decouple and organize the sql queries, which would otherwise have been prepared statements buried in the code, into their own independent xml configuration files. The framework then leverages the resultmaps (also config within the sql-map files) in order to convert from a result-set

into a pojo, by explicitly mapping columns to java class variables. While the framework can reflectively perform this column to java variable mapping, we've intentionally defined them expicitly for ease of maintenance.

bu.met.cs.cs673.pm.dto - these are the dto objects, basically Java object representations of database tables.

6. References

7. Glossary

mybatis - dao persistence framework similar to hibernate (formerly known as ibatis).
dto - data transfer object

apache-wink - REST implementation used in the project for the front-end services to the back-end db.