WEB BASED TIMESHEET MANAGEMENT AND INVOICE GENERATION SYSTEM FOR RELYCOM INC.

Pragnya Aradhyula
B.Tech., Jawaharlal Nehru Technological University, India, 2007

PROJECT

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

SOFTWARE ENGINEERING

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

FALL
2010
WEB BASED TIME SHEET MANAGEMENT AND INVOICE GENERATION SYSTEM FOR RELYCOM INC

A Project

by

Pragnya Aradhyula

Approved by:

__________________________, Committee Chair
Jinsong Ouyang, Ph.D.

__________________________, Second Reader
Scott Gordon, Ph.D.

__________________________
Date
Student: Pragnya Aradhyula

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the Project.

__________________________, Graduate Coordinator  ________________
Nikrouz Faroughi, Ph.D.                    Date

Department of Computer Science
Abstract

of

WEB BASED TIMESHEET MANAGEMENT AND INVOICE GENERATION SYSTEM FOR RELYCOM INC.

by

Pragnya Aradhyula

Relycom Inc is a Software Consulting firm, which specializes in providing Enterprise wide solutions to their clients spread over different states in the US. Many of their employees visit their clients and work at client locations as software consultants. Each of these consultants needs to get a weekly timesheet signed and approved by their managers at client locations and these need to be sent to Relycom for invoicing purposes. Currently this process is being done manually using MS Excel and Fax. The proposed web based application aims to automate this process of tracking time and the approval process as well as generating invoices to the clients for the consulting services provided.

_______________________, Committee Chair
Jinsong Ouyang, Ph.D.

_______________________
Date

iv
ACKNOWLEDGMENTS

Many people helped me in the completion of my Master’s Project in various ways. I would like to acknowledge my gratitude to all of them. I am very thankful to my project advisor, Dr. Jinsong Ouyang, for his support, advice and creative ideas, which helped me gain knowledge and exposure to many new areas in technology. I would also like to thank Dr. Scott Gordon, for accepting to be the second reader and taking time to review my project.

I would also like to thank Mr. Prasad Dorbala, President Relycom Inc., who gave me an opportunity to design and develop this project. Developing this project gave me valuable experience and exposure to industry standard technologies. I would also like to thank Mr. Harish Vishnuvajjala of Relycom Inc, for his valuable time, advice and feedback.

I extend my sincere appreciation to my husband for always being there for me and in helping me accomplish my goals. I am grateful to my parents who have aroused in me an ambition to pursue a Master’s degree abroad and have provided unending encouragement. I would like to dedicate this project to my husband and parents.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>ix</td>
</tr>
<tr>
<td>List of Figures</td>
<td>x</td>
</tr>
</tbody>
</table>

Chapter

1. INTRODUCTION ................................................................. 1
   1.1 Background Information ........................................... 1
   1.2 Purpose of this Project-eTime ................................... 1
   1.3 Overview of this Document ....................................... 2

2. TECHNOLOGIES AND TOOLS USED ........................................ 3
   2.1 Overview of the Technologies Used .......................... 3
   2.2 Java ........................................................................ 3
   2.3 Hibernate .................................................................. 4
   2.4 Struts Framework ...................................................... 5
   2.5 Java Server Pages (JSP’s) .......................................... 5
   2.6 JUnit ........................................................................ 6
   2.7 Apache Tomcat .......................................................... 6
   2.8 MySql ....................................................................... 6
   2.9 Eclipse IDE .............................................................. 7

3. SYSTEM REQUIREMENTS SPECIFICATION .......................... 8
   3.1 Functional Requirements ......................................... 8
   3. Non-Functional Requirements ................................... 10

4. DESIGN AND IMPLEMENTATION ........................................ 12
4.1 Introduction

4.2 Domain of eTime (Timesheet Management System)

4.3 Smart UI Anti Pattern and Why We Need to Avoid this Pattern

4.4 Domain Driven Design - What is it and Why is it Important?

4.4.1 Ubiquitous Language

4.4.2 Layered Architecture: What is it and Why is it Important?

4.4.3 Building Blocks of DDD - Domain Layer Patterns

4.4.3.1 Entities

4.4.3.2 Value Objects

4.4.3.3 Services

4.4.3.4 Aggregates

4.4.3.5 Factories

4.4.3.6 Repositories

4.4.4 Modules

4.5 Building the Domain Layer Using DDD

4.5.1 Requirements Gathering

4.5.2 Identifying Domain Layer Objects

4.5.3 Identify Aggregate Roots

4.5.4 Identify Repositories

4.6 Infrastructure Layer

4.7 Application Layer

4.8 Presentation Layer

4.9 Architecture of eTime

4.10 Processing of a User Request from End to End in eTime

4.11 Database Model

4.11.1 Organization Table

4.11.2 Manager Table

4.11.3 Project Table

4.11.4 User_Info Table
4.11.5 Employee Table ................................................................. 53
4.11.6 Timesheet Table............................................................... 54
4.11.7 AuditLog Table ................................................................. 55

5. WEB INTERFACE DESIGN ................................................................. 56

5.1 User Interfaces for Admin ......................................................... 56
  5.1.1 Login Page ........................................................................ 56
  5.1.2 Organization Details Page .................................................. 57
  5.1.3 Manager Details Page ....................................................... 58
  5.1.4 Project Details Page ......................................................... 59
  5.1.5 Employee Details Page ...................................................... 60
  5.1.6 Search Timesheet Page ...................................................... 61
  5.1.7 Search Based Timesheet Details Page ............................... 62
  5.1.8 Reports Home Page ........................................................ 63
  5.1.9 Invoice Generation .......................................................... 64

5.2 User Interfaces for Manager ...................................................... 67
  5.2.1 Search Timesheet Page for Manager ................................. 67
  5.2.2 Search Based Timesheet Details Page ............................... 68
  5.2.3 Timesheet Detail Page After Manager Approval ................. 69

5.3 User Interfaces for Employee .................................................. 70
  5.3.1 Timesheet Page ............................................................... 70
  5.3.2 Timesheet Detail Page After Employee Submission .......... 71

6. CONCLUSION AND FUTURE WORK .............................................. 72

Appendix Database Create Scripts .................................................. 74
Bibliography ...................................................................................... 78
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1 Organization Table</td>
<td>50</td>
</tr>
<tr>
<td>Table 2 Manager Table</td>
<td>51</td>
</tr>
<tr>
<td>Table 3 Project Table</td>
<td>51</td>
</tr>
<tr>
<td>Table 4 User Info Table</td>
<td>52</td>
</tr>
<tr>
<td>Table 5 Employee Table</td>
<td>53</td>
</tr>
<tr>
<td>Table 6 Timesheet Table</td>
<td>54</td>
</tr>
<tr>
<td>Table 7 Audit Log Table</td>
<td>55</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>Use Case Diagram for the System</td>
</tr>
<tr>
<td>2</td>
<td>A Navigation Map of the Language of Model Driven Design</td>
</tr>
<tr>
<td>3</td>
<td>Organization - Manager – Project</td>
</tr>
<tr>
<td>4</td>
<td>Address Value Object</td>
</tr>
<tr>
<td>5</td>
<td>Employee – Billing Info</td>
</tr>
<tr>
<td>6</td>
<td>Timesheet – AuditLog – Invoice</td>
</tr>
<tr>
<td>7</td>
<td>Domain Layer Objects and Their Relationships Represented in UML</td>
</tr>
<tr>
<td>8</td>
<td>Organization Domain Object</td>
</tr>
<tr>
<td>9</td>
<td>Manager Domain Object</td>
</tr>
<tr>
<td>10</td>
<td>Organization Class - Business Method to Add a Manager</td>
</tr>
<tr>
<td>11</td>
<td>TimeSheet Domain Object - Method to Add a New Audit Log</td>
</tr>
<tr>
<td>12</td>
<td>Organization Repository’s Save Method</td>
</tr>
<tr>
<td>13</td>
<td>Organization Service’s Method</td>
</tr>
<tr>
<td>14</td>
<td>An Employee DTO class</td>
</tr>
<tr>
<td>15</td>
<td>eTime Architecture</td>
</tr>
<tr>
<td>16</td>
<td>Manager Controller’s Save Manager Details Method</td>
</tr>
<tr>
<td>17</td>
<td>Organization Service’s Method to Save Manager Details</td>
</tr>
<tr>
<td>18</td>
<td>Organization Class - Business Method to Add a Manager</td>
</tr>
<tr>
<td>19</td>
<td>Organization Repository’s AttachDirty Method</td>
</tr>
<tr>
<td>20</td>
<td>Database Schema Diagram of eTime</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

1.1 Background Information

Relycom Inc is a Software Consulting firm which specializes in providing enterprise wide software solutions to their clients spread over different states in the US. Many of their employees visit their clients and work at client locations as software consultants. Each consultant has to submit a timesheet every week that contains the details of the number of hours he/she worked each day in that week to his manager at the client location. Once the manager approves the timesheet, it has to be sent to Relycom’s address. An invoice has to be generated for the client based on the hours worked by the employee as well as the consultant’s billing rate. The information about client organization, project, consultant, client manager also need to be maintained.

1.2 Purpose of this Project - eTime

Currently, the process of timesheet submission and approval is being done manually at Relycom and the company wishes to automate this manual workflow. The current process of manually sending the excel timesheets and maintaining them to calculate invoice is burdensome, error prone and time consuming. eTime aims to automate this process by providing a web interface that can be accessed from anywhere and anyone who is a valid user in eTime. The employee of Relycom can now enter his timesheet information into eTime and submit it to his manager via eTime. The managers can access
eTime, review the timesheets submitted to them for approval, and approve them. The administrators at Relycom can then review the timesheets and generate invoices using eTime once the timesheets have been approved. In addition to the above, eTime also enables Relycom to maintain information about their clients and employees.

1.3 Overview of this Document

The remainder of this document has been organized in the following way. Chapter 2 deals with the Technologies and tools used in this project. Chapter 3 deals with the system requirements specifications. Chapter 4 includes the design specifications and a design methodology called Domain Driven Design and how it is used in this project. Chapter 5 includes the various user interfaces that were developed as part of this project. Chapter 6 includes the conclusion and future work. Appendix lists the database table creation scripts.
Chapter 2
TECHNOLOGIES AND TOOLS USED

2.1 Overview of the Technologies Used

The core business logic was written in Java 5.0 programming language. For persistence, Hibernate Object-Relational mapping framework was used. For building the user interface, Struts 1.3, a presentation layer framework and Java Server Pages (JSP’s) were used. For writing unit tests JUnit 4.0 was used. Apache Tomcat 6.0, a web server was used for deployment of the web application. The database used by eTime was MySql. Eclipse IDE was used for development.

2.2 Java

Java is a popular Object Oriented programming language for developing web applications. One of the main advantages of Java is its platform independence. Platform independence comes with the concept of ‘Byte Code’. Once a program written in Java is compiled, it transforms into bytecode instructions. This bytecode runs on a Java Virtual Machine, which executes the bytecode instructions. Once you have the bytecode and a JVM, the program can run on any machine [4]. Java Servlets is a technology that helps in implementing a concept of client-server type architecture. A servlet is a server side component that can receive HTTP Requests, and retrieve the data sent by the client. The request can then be processed on the server and a response sent back to the client [4]. Java version 5 was used for developing eTime.
2.3 Hibernate

Hibernate is an open source Object – Relational (O-R) mapping framework used in the persistence layer. The main advantage of O-R mapping frameworks is that the developer does not need to write SQL queries and map the result sets back to objects. Hibernate provides an object based query language called HQL (Hibernate Query Language). Using an O-R framework like Hibernate, we can switch databases very easily, for example from Oracle to MySql by changing one single property in a configuration file. Another advantage is Transitive Persistence. We can configure Hibernate to cascade operations from parent to child, for example – save, update, delete. We can also load child objects automatically when a parent is loaded.

For example, when a couple of objects have a parent child relationship, if we persist the parent object, Hibernate can keep track of any changes made to the child and save both the objects. Instead, with another persistence technology like JDBC we would need to keep track of the relationships between the objects ourselves and it gets hard to keep track of which child object got updated in the object hierarchy and update only that specific object. Hibernate 3 was used in the persistence layer of eTime.
2.4 Struts Framework

Struts is an open source, java based presentation layer framework that helps in developing web layer. Struts follows MVC – Model View Controller pattern. This pattern emphasizes on clear separation of the front-end html/JSP logic from the actual server side controller logic that handles the user request. The data submitted by the users in the web page is retrieved in a clean fashion by the Struts framework using a Form Bean. Struts provides an Action Servlet, which is used for processing user requests and sending response back to the client. Also, Struts framework provides Tiles framework which helps in creating reusable templates for the user interface. A common template/layout can be applied for all the web pages easily using Tiles. Any changes to the styles/layers can be made in a centralized location and can be propagated to all pages easily. Struts 1.3 framework was used in eTime.

2.5 Java Server Pages (JSP’s)

Java Server Pages technology helps in creating web pages with both static and dynamic content using HTML and JSP constructs [1]. The dynamic content in the JSP page is executed at runtime on the web server and a HTML page is sent back to the web browser. Main advantage of JSP is its ability to create dynamic web pages, whereas pure HTML document can only generate static web pages. I used JSP 1.2 and HTML for developing the web pages in eTime.
2.6 JUnit

JUnit is an open source unit testing framework that helps a developer in writing unit tests for code written in Java programming language. With the help of Unit tests, we can test individual methods in code and verify that they work as expected. Without unit tests, there are greater chances of bugs being discovered later on and fixing them will be costly. Unit tests can be written in JUnit 4.0 by using an @Test annotation on top of your method. We can write the test code inside the test method and test the code we want to test. JUnit provides Assert statements like AssertNotNull etc. can be used to assert that the result of a program is as expected.

2.7 Apache Tomcat

Tomcat is an open-source web server/servlet container for deploying and running web applications. As a web server, Tomcat can receive HTTP Requests and send HTTP Responses back to the client. It is free and easy to setup and use. Apache Tomcat 6.0 version was used in eTime. A web application is packaged into the format of a web–archive file, also known as a ‘.war’ file. The war file is copied into Tomcat’s ‘webapps’ folder. This is the deployment step. Once a war file is deployed, starting up Tomcat server starts up the web application.

2.8 MySql

MySql is an open source relational database. MySql provides many storage engines also known as table types. Each storage engine provides different options related to
transaction management, locking scheme, data storage space and so on. eTime uses the InnoDB storage engine. This engine has aggressive memory caching and thus provides high-speed data retrieval capabilities [3]. MySql version 5.1 was used for storing data in eTime.

2.9 Eclipse IDE

Eclipse is an Integrated Development Environment and it is open source. Usage of Eclipse IDE makes development and compilation easy. It has plugins for JUnit, Tomcat which help in developing and running applications, JUnit tests easily. Eclipse 3.5 Galileo was used for developing eTime.
3.1 Functional Requirements

The eTime system should support 3 kinds of users – Admin, Manager, Employee.

Administrator should have the ability to do the following

- Create a new client Organization and add its details.
- Add a new Manager associated with a client Organization into the system
- Add a new Project into the system along with the Project details.
- Add a new Employee into the system which includes his personal information and billing information
- Associate an Employee to a Project
- Search for timesheets based on certain search criteria like the start date of a timesheet, end date of a timesheet, the status of a timesheet or based on a particular Employee.
- Override or update timesheets when a Manager or an Employee makes errors while submitting the timesheets.
- Generate Invoice for each timesheet based on employee’s billing rate and hours worked.
- Modify/Update any details related to an Employee, Manager, client Organization, Project and Timesheet.
- Login and logout of the application.
An Employee of Relycom should have the ability to do the following

- Save and Submit Timesheet for a specific week. Add notes for a timesheet.
- Search for Employee’s own timesheets based on certain search criteria like the start date of a timesheet, end date of a timesheet, the status of a timesheet.
- View timesheet submission, approval history.
- Login and Logout of the application.

A Manager of Relycom should have the ability to do the following

- View a list of Employees working under his Projects.
- Approve or Reject Timesheets for only the Employees reporting to him.
- Add notes if needed for a timesheet.
- Search for timesheets of all Employees working in Projects under him based on certain search criteria like the start date of a timesheet, end date of a timesheet, the status of a timesheet or the Employee name.
- View timesheet history.
- Login and Logout of the application.

Figure 1 shows the Use Case diagram of the system.
3.2 Non-Functional Requirements

- Number of concurrent users - The system should be able to handle at least 50 concurrent users.
- Response time - Response time of the System should be less then 5 sec.
- Security – Provide role based Authentication and Authorization
- Usability – System must be ‘usable’ implying that the system should provide an easy to use user interface that doesn’t involve a big learning curve.
- Performance – System should allow a user to perform his/her task as efficiently as possible and the tasks should take as little time as possible.
- Cost – Cost of building and maintaining the application should be low. Usage of open source technologies is recommended to keep the cost low.
4.1 Introduction

There are many ways to design and develop a software/product. In order to create a good product we need to have a very good understanding of the domain of the product. For example to develop a Health Insurance system, we need to know about the domain of Insurance as well as Health Care. Without having proper knowledge of the domain, the software that is developed would be very difficult to understand, maintain and might not meet the requirements of the desired product. The first step in building a software application is to get a good understanding of the domain by talking to the domain experts and then, create a domain model from those discussions.

Domain Driven Design (DDD) is a set of principles and guidelines on creating a software design that is closely tied to the domain model. In the following sections I am going to discuss step by step about the advantages of using DDD and how I designed eTime using DDD.

4.2 Domain of eTime (TimeSheet Management System)

This software application is related to the domain of ‘Software Consulting’. Relycom provides Software consulting services to its various clients. Software consultants/(Employees) of Relycom, work under a single Project at a Client
Organization. Each Project is managed by one single Manager at the client Organization. eTime needs to store various client Organization details, client Manager details and client Project details. When an Employee of Relycom is placed at the client location, a Project needs to be assigned to the Employee in the eTime system. A client Manager supervises the Project and is responsible for approving the weekly Timesheet submitted by the Employee.

Every Employee needs to enter a weekly Timesheet into eTime. After an Employee submits his Timesheet, his Manager when he logs in, would be able to view it and approve or deny it. After Manager’s approval the Admin at Relycom needs to view the client Manager approved timesheet and approve it from his side also. Each Employee has his Billing information also stored in the system. The admin would generate a weekly Invoice per Timesheet, and send it to the client Organization.

4.3 Smart UI Anti Pattern and Why We Need to Avoid this Pattern [5 pp. 76-79] [6]
A web application typically consists of User Interface (UI), Database logic, Business logic as well as logic related to Security, Messaging and Logging. A bad way of designing a system is to write the business logic, UI and the database logic all in one Class/Layer resulting in highly complex spaghetti code. This is known as ‘Smart UI anti pattern’. This might seem ok for small applications, but any future upgrade might require rewriting the whole application. The code developed this way is not reusable and is hard to maintain. When all the code is mixed up in one place without clear separation, any
change to the system would be costly because of the complexity of code. Tight coupling would exist between UI, database and business logic. For example consider the scenario where we need to fetch the details of an employee based on the employee’s name. If we were to follow the Smart UI pattern, the pseudo code for a class might look like this.

Sample Psuedo Code: User enters employee name and clicks a submit button in the webpage.

EmployeeDetailsClass {

    Parse user’s HTTPRequest
    Authenticate/Authorize User

    Open Sql Connection
    Query for User details
    Analyze Result set
    Build UserDetails Object

    Convert to UI representation format
    Send HTTPResponse back to user

}

As we can see, all the code is written in one layer. So, code reuse is not possible and there is no separation of business logic.
DDD emphasizes on using a layered architecture to separate different concerns into different layers and especially separating domain business logic out into its own layer - the domain layer.

4.4 Domain Driven Design – What is it and Why is it Important? [5]

Initial Requirements Phase and coming up with Design:

Similar to doing a blue print for a building in Civil Engineering, we need to develop a Design Model for a software product based on the domain knowledge acquired from domain experts.

Typically, domain experts discuss the requirements with the business analysts and then business analysts document the requirements and pass it to the design/development team. The design/development team then comes up with a design-model, which they think corresponds to the requirements and the project is developed. There is no emphasis on communication between the domain experts and the developers in these approaches.

In the transition of the knowledge from domain experts to business analysts and then from business analysts to the developers, the essence of business requirements is lost. So to avoid these problems, the domain experts should directly collaborate with the design team and come up with the design.
4.4.1 Ubiquitous Language [5 pp. 23 -25]

The domain experts speak in terms regularly used in their daily business. The developers on the other hand talk in terms of object oriented language to come up with the design. The problem is about how to make these two groups communicate. Even if these two groups did try to communicate they would have to translate between each other’s languages since they don’t share a common language. During this translation the essence of the domain is lost. When the developers finally come up with a design model and try to get feedback from the domain experts, the domain experts cannot understand/evaluate the model because the model is full of technical terminology. This problem can be solved by building a common language referred to as Ubiquitous language. [5 pp. 23 -25]

The developers should communicate with the domain experts and understand their jargon and start using that in their design. Thus developers and the domain experts can agree upon common usage terms and this leads to a Ubiquitous language. The design model developed will now contain terminology from the Ubiquitous language and thus makes it easy for the domain experts to understand and evaluate the design.

Initially, I held discussions with the domain expert at Relycom and understood their business domain. I made sure I used the same terminology in the domain model and source code.
4.4.2 Layered Architecture: What is it and Why is it Important? [5 pp. 68 -75]

We have seen the disadvantages with Smart UI Pattern – which involves writing all the Database, UI, Business logic, Security and logging code into one layer. A good approach for organizing code is to use a Layered Architecture. The whole project is organized into multiple layers depending on the nature of the code. UI code, Database code, business logic, security and logging are separated into individual layers. This way each layer is highly cohesive and responsible for specific functionality and thereby achieve loose coupling.

The most common layers in a layered architecture include: Presentation Layer, Application Layer, Domain Layer, Infrastructure layer, though each project might have additional layers based on their needs. Responsibilities of each layer-

- **Presentation layer** is responsible for all the User Interface related code and handles the requests from user. For e.g parsing the HTTPRequest etc and sending back the HTTPResponse back to the client.

- **Application layer** services deal with coordinating the overall activity in the application. For e.g if we were to modify a TimeSheet in eTime. The application layer service object first uses infrastructure layer objects to get the TimeSheet domain object from the database and then calls the appropriate business methods on the timesheet domain object. It could then use the infrastructure layer objects for persisting the updated domain object. Application layer objects are stateless
and maintaining the state and business logic is the responsibility of domain layer objects.

- **Domain Layer** is where all the business logic is present and domain objects maintain state. Persistence of Domain objects is not the responsibility of this layer, it is delegated to infrastructure layer.

- **Infrastructure layer** is responsible for dealing with infrastructure aspects of an application like persistence, e-mail messaging, security and logging.

There are many flavors of the layered architecture. However, *DDD’s main concern is to isolate the domain logic into its own layer so that the domain objects do not bother about anything other than pure business logic.* The burden of persisting the domain objects and displaying them should be taken care of by the other layers. For eTime, I identified the Domain objects and the Business Rules that are part of the domain layer. Then I split the architecture of eTime into 4 layers giving each layer a specific responsibility.

4.4.3 Building Blocks of DDD - Domain Layer Patterns [5 pp. 81 -161]

DDD categorizes the various objects in Domain layer into – Entities, Value Objects, Domain Services, Factories, Repositories and Aggregates.

Figure 2 shows the relationship between various patterns in domain driven design [5 pp. 65]. Let us see more about each pattern in detail in the following sections.
4.4.3.1 Entities:

*Objects with an Identity.*

In our domain, there would be some objects that need to have unique identities and these identities should be maintained over time. Such objects are called *Entities.* When such objects are created we need to generate a unique identifier for each object to distinguish one from the other. We should be able to persist these objects to the database and be able to reconstitute them back into an object after retrieval from the Database.
An id can be generated by either giving a unique system generated Id (number) or by taking a unique attribute of the entity generated by the database, usually the Primary Key. The attributes associated with an entity should also be identified.

For example, Employee is an entity in eTime. An Employee has to be able to be distinguished from other Employee. So this makes it an entity and therefore has a unique identifier – Employe ID. The attributes of the Employee are: EmployeeId, firstname, lastname, home address (address, street, city, zipcode), billing information, username, and password.

4.4.3.2 Value Objects:

*Objects without an Identity.*

In our domain there could be some objects that do not need to have a unique identity and be able to be used interchangeably. Such objects are called *Value Objects.* Some attributes of an Entity can form a value object. For example, in eTime, Employee is an Entity with address1, street, city, zip as some of its attributes. We can place these attributes into an Address Object, which becomes a value object. If two Employees are residing at the same address, we can use an address object interchangeably for both of them. So the identity of the address object is not as important as its attributes. Hence this is a value object. In order to be able to reuse these objects safely we need to either make these objects immutable, which means that once this object is constructed this cannot be
updated, or we need to pass a new copy of the object every time it is requested, instead of a reference to the same object.

It is important to identify Value objects in a given domain. If we classify all the objects in our domain as Entities, it becomes costly to maintain them, because unique identity needs to be generated and maintained for each Entity.

4.4.3.3 Services:

*Logic that doesn’t belong to a specific Entity or Value Object.*

There might be some operations/behavior in our domain which does not belong to any entity object. We should not attempt to put the functionality that doesn’t belong to an Entity or a Value object into it just because that functionality needs to be in some class. Instead we need to create a separate category of Classes called Services. Services do not maintain state and the operations are stateless. That way, we can keep the Entity and Value objects clean and avoid polluting them with code that doesn’t belong to them.

For e.g. in eTime, we need to generate an Invoice for an Employee for a specific Timesheet period. We need to know who the Employee’s Manager is, and which Project he is working under and then based on his billing rate we need to calculate the Invoice amount. This logic neither belongs to an Employee object, nor Manager object nor a Project object. It would be unjust if we put this logic in any of these classes. Instead we need to create a separate Service class called TimeSheetService and a method inside that
to write this logic. This method would call the necessary methods in each of these domain classes and coordinate the flow.

Services could be in both Domain layer and in Application layer. Domain layer services only deal with domain objects based interface and are present in Domain layer, where as Application layer services present in Application layer could act as an interface between the Presentation layer and Domain layer to shield the domain layer objects, and could have an interface other than domain objects. I would discuss more below when I present eTime as a Case Study.

4.4.3.4 Aggregates:
A domain might have many objects with multiple relationships existing between them like one-to-one, one-to-many. For example in the eTime application, Project could have multiple Employees assigned to it and an Employee could belong to a single Project. This is a bidirectional relationship and we need to reduce them as much as possible to reduce complexity. In the eTime domain it makes sense to assign a Project to an Employee rather than adding an Employee to a Project, because there could be instances when Employee does not belong to any Project and is on bench. So the one to many relationship between Project and Employee is not maintained. Only the relationship from Employee side is maintained.
We need to shield some of the relationships inside a particular Entity, to avoid complexity. We designate one Entity as the Aggregate root and any access to the member Entities inside this root object need to go through the Aggregate root only. That way, the outside world is shielded from what happens in this Aggregate. The Aggregate root can hold references to any of the aggregate’s objects, and the other objects within a root can hold references to each other, but an outside object can hold references only to the root object. It is possible however to pass the objects inside an aggregate outside as transient objects instead of references. That way any change to passed object doesn’t affect the original object inside the root.

Retrieving or saving the child entity objects to the database, can only be allowed by an Aggregate root. We should not provide the ability to save child entity objects within an aggregate root directly. References to objects within an Aggregate should be available only by retrieving the Aggregate root from database and then navigating to appropriate object within that.

For example in the eTime project consider the scenario where a Client Organization needs to be added along with a Manager and a Project. There is no meaning for a Project or a Manager to exist in eTime System without a Client Organization. So, I made Organization – Manager – Project as an Aggregate and Organization as the Aggregate root. Any access to a Project or Manager should go through the Client Organization root.
The Organization root can return transient instances of Manager and Project if needed outside the aggregate boundary.

4.4.3.5 Factories:

*For a complex object creation.*

Sometimes a complex aggregate object might need to be constructed. That would mean that the client who is creating the object needs to know the internal details within that domain object. This is not a good idea. So instead a Factory class is written and this class would deal with creation of these complex objects. But sometimes creating a Factory class could be overkill. We should use factories mainly when construction of object using a constructor is complex and when inheritance is used to create subtype objects.

4.4.3.6 Repositories:

*To persist and retrieve Entity objects from database or any other persistence storage.*

Once the domain objects have been created in memory, they need to be persisted to a storage mechanism for e.g a database. Storing /retrieving from a database usually involved writing some Sql code or using an Object-Relational mapping Framework. We should separate the actual domain logic from the persistence logic, otherwise our domain layer would be cluttered with infrastructure persistence code which doesn’t belong in the domain layer. This could be achieved by using another pattern called Repositories.
We should create a Repository Interface, which only deals with domain objects. A client should be able to request for domain objects via a Repository method and the repository method should return appropriate domain objects based on the request. The client doesn’t need to know how the repository gets the required domain object from the database. The Repository could use Infrastructure layer and construct the domain object and return them back to the client.

Another important point is that we need to provide repository interfaces only for aggregate root objects. A client which needs a domain object needs to access a repository corresponding to an aggregate root and the repository should only save or retrieve aggregate roots. Otherwise it would break the rule of encapsulation and why we chose to create aggregates in the first place. The implementation of a repository can be closely tied to the Infrastructure layer, but the repository interface should only deal with pure domain model objects.

In the eTime project, I created an Organization Repository for the Organization – Manager -Project Aggregate. This repository can fetch or save an Organization Domain object and give it to the client. The client can then traverse through this object to obtain a Project or a Manager. There is no separate Repository for Project or Manager and they are not directly accessible from the database by the client.
4.4.4 Modules [5 pp. 109-116]

Organization of the Model/Code into Modules to have high cohesion and low coupling.

To reduce complexity, we should group together related components of the model into their own modules, so that by looking at the modules we get a high level understanding of the Model. These modules should also be named as per the Ubiquitous language. We can think of modules as the concept of Packages in Java. Classes that logically belong together and which perform related functions are grouped together into a single module, thereby giving a module high cohesion. In the eTime project, I have different modules like Organization, Employee and TimeSheet. Each module has its related Entities and Value Objects and Repository Interface inside them. I have seen some projects where they grouped all domain objects into one package, all DTOs inside one package, and all services into one package. This is not a good idea. They were not logically grouped. Instead we should group all classes, which perform a single logical function into the same module/package.

4.5 Building the Domain Layer Using DDD

4.5.1 Requirements Gathering

I held discussions with a domain expert at Relycom to understand the Software Consulting domain and the features required in the eTime Application. During these discussions, I identified the important concepts and terminology used in the Consulting domain like Client-Organization, Project, Manager, Employee, Billing Information,
Timesheet, Invoice, and Audit Log. This terminology became part of the project’s Ubiquitous language and part of the Domain Model.

4.5.2 Identifying the Domain Layer Objects

After understanding the domain, I started to identify the various objects inside the domain like Entities, Value Objects and Aggregates as well as relationship between them.

a. Organization, Manager, Project:

Relycom’s Employees get placed at client Organizations as software consultants. Organization has many Managers and each Manager could manage multiple Projects. Each of these needs to have a unique identity, so they are all Entity objects.

![Organization - Manager - Project](image)

**Figure 3: Organization - Manager - Project**

Address:

I moved the address related attributes into its own Address object. Since multiple people could share the same Address, it is a Value object.
b. Employee, Billing Information:

Each Employee object needs an identity of its own. So, Employee becomes an Entity. Billing Information attributes belong to a separate class and it is a Value object.

![EmployeeBilling](Image)

Figure 5: Employee – Billing Info

c. Timesheet, Audit Log, Invoice:

When an Employee from Relycom starts working at a client Organization on a Project, he submits weekly timesheets to the Project’s Manager. So, each timesheet’s details have to be maintained and uniquely identified from other timesheets. All actions being performed on a timesheet have to be tracked. So, whenever a Timesheet is created or modified an
AuditLog that keeps track of how the timesheet was modified and who modified it, needs to be saved. So, Timesheet and AuditLog are Entity Objects. Each Timesheet can have many AuditLogs. The admin at Relycom generates an Invoice for every Timesheet weekly. Each Timesheet will have one Invoice.

![Diagram](image)

Figure 6: Timesheet – AuditLog – Invoice
4.5.3: Identify the Aggregate Roots

I identified three Aggregates

- Organization(Root) – Manager – Project
- Employee(Root) – Billing Info
- Timesheet (Root)– Invoice –AuditLog

Figure 7: Domain Layer Objects and Their Relationships Represented in UML
a. Organization(Root) – Manager – Project Aggregate:

The Admin of eTime system needs to be able to create new Organizations in the system. Organization has many client Managers and each Manager could manage multiple Projects.

An Admin cannot add a Manager or a Project without first creating an Organization. Based on this scenario, I identified Organization as the Aggregate root for the Manager, Project and Organization Entity objects.

Only an aggregate root can have a Repository. Organization repository takes care of the persistence logic for Manager, Project and Organization entities. Even in case of accessing the children, for example a Manager, we have to retrieve the Organization object, and then access the Manager via Organization.
public class Organization implements java.io.Serializable {

    // Fields
    private Integer oid;
    private String name;
    private Address address;
    private String email;
    private String phone;
    private Long version;
    private Set<Manager> managers = new HashSet<Manager>(0);

    public Organization(String name, String email, String phone,
                         Address address, Set managers) {
        this.name = name;
        this.address = address;
        this.email = email;
        this.phone = phone;
        this.managers = managers;
    }

    Figure 8: Organization Domain Object

    As we can see the Organization class has a set of Managers inside it. Similarly, a
    Manager class has a set of Projects inside it.
public class Manager implements java.io.Serializable {
    // Fields
    private Integer mgrid;
    private Long version;
    private UserInfo userInfo;
    private Set projects = new HashSet();
    private Integer oid;
    private String title;

    public Manager(UserInfo userInfo, Set projects) {
        this.userInfo = userInfo;
        this.projects = projects;
    }

    Figure 9: Manager Domain Object

    Consider an example to update Manager for a given Organization as shown in Figure 10.

    public void addOrUpdateManagerUserInfo(Manager mgr) {
        boolean isUpdate = mgr.getMgrid().equals(0) ? false : true;

        if(isUpdate) {
            for(Manager manager : this.managers) {
                if(manager.getMgrid().equals(mgr.getMgrid())) {
                    manager.setTitle(mgr.getTitle());
                    copyUserInfo( manager.getUserInfo(), mgr.getUserInfo());
                }
            }
        } else {
            this.managers.add(mgr);
        }
    }

    Figure 10: Organization Class - Business Method to Add a Manager
Once we add a Manager to an Organization Domain object, we persist the Organization object by calling `OrganizationRepository.save(OrganizationObj)` method. This will take care of saving Manager object also.

b. Employee(Root) – Billing Info Aggregate:

The admin of Relycom needs to create an Employee in the eTime system and also provide Billing Information for each Employee. An Employee could be assigned a single Project at the client location or if he is on bench he does not have any Project assigned. I created an Employee Repository to deal with storing, retrieving Employees and their Billing Information

c. Timesheet(Root) – Invoice –AuditLog Aggregate:

The Employees need to be able to submit Timesheets, client Manager needs to be able to approve timesheets and also the Admin needs to approve them. The Admin needs to be able to query for various timesheets without regard to a specific employee. He can for example access the various pending timesheets for all the employees on all the projects. So, Timesheet needs to be its own Aggregate root and have its own Timesheet repository.

Each timesheet entity can have one or more Audit Log entity objects and one Invoice Value object.

Whenever a user (Admin/Employee/Manager) makes an update to a specific Timesheet entity object, an Audit Log object is created and stored in the database. This business
logic of adding an Audit Log is present in the Timesheet object. Audit Log has no meaning without a specific timesheet. So, it is made part of the timesheet Aggregate root and so, in order to access a specific Audit Log you have to go through the Timesheet root.

Various users could update Timesheet status. This business logic of changing/updating status is also part of the TimeSheet domain object.

Example: TimeSheet Business Method to add a new Audit Log:

```java
public void updateAuditLogs(UserInfo modifiedByUser, String updatedTSStatus, String userComment)
{
    AuditLog al = new AuditLog();
    al.setNotesEntered(userComment);
    al.setSystemComment(updatedTSStatus);
    al.setUserInfo(modifiedByUser);
    this.auditLogs.add(al);
}
```

Figure 11: TimeSheet Domain Object – Method to Add a New Audit Log

4.5.4 Identify Repositories

To decouple the persistence logic from the domain objects I used the repository pattern. Repository Interfaces are part of the Domain Layer whereas Repository Implementations
are part of Infrastructure layer. For example, we can use one persistence technology like JPA and then later on plan to change to Hibernate or Ibatis or JDBC. Then we can change the implementation in the Infrastructure layer without affecting the domain layer objects at all.

eTime has the following Repositories:

1. Organization Repository
2. Employee Repository
3. Timesheet Repository

Organization Repository:
For the Organization – Manager – Project Aggregate there is an Organization Repository for the organization aggregate root, which takes care of the persistence logic for manager, project, and organization entities. There are no separate repositories for Manager and Project. When you save the organization it should transparently also save Manager, Project.

Employee Repository:
For the Employee – Billing Info Aggregate, EmployeeRepository takes care of the persistence.
Timesheet Repository:

Timesheet Repository takes care persistence for Timesheet – Invoice – AuditLog Aggregate.

4.6 Infrastructure Layer

Repository Implementation is part of the Infrastructure Layer. One of the concerns of infrastructure layer is persistence. I used Object-Relational Mapping framework – Hibernate. One of the main advantages of using O-R Mapping framework is Transitive persistence. When you want to persist a child object, the aggregate needs to be persisted and O-R Mapping framework uses transitive persistence, which takes care of persisting the child object automatically. This is important for DDD. I observed that with JDBC it is hard to keep track of which child object got updated in the object hierarchy and update only that specific object. So it would not be easy or clean to implement Repositories with simple JDBC code and still follow DDD principles. O-R mapping framework maintains the same structure of Domain objects and Aggregates, whereas with JDBC we would have to save each child object separately.

Sample code from Organization Repository to save an Organization is shown in Figure 12.
Figure 12: Organization Repository’s Save Method

Database schema and tables are explained in section 4.11.

4.7 Application Layer

Application layer services decouple the domain layer and the presentation layer and control the flow. They take the presentation layer DTO objects and use Assembler classes for converting them to domain objects and vice versa. They can get the required domain objects from the repository and call some business methods on the domain objects and then persist the modified domain objects using a repository. The Services
don’t maintain any state. They just coordinate the flow and get things done by domain objects and repositories.

The services in eTime application are:

Organization Service – Deals with requests from the presentation layer for adding/updating/ retrieving an Organization or Manager or Project.

Employee Service – Deals with requests from the presentation layer for adding/updating/ retrieving an Employee or Billing Information

Timesheet Service – Deals with requests from the presentation layer for adding/updating/ retrieving a Timesheet, Audit log or Invoice information.

Login Service – Deals with requests from the presentation layer for authorizing and authenticating a user.

A sample method to retrieve Managers based on OrganizationId in Organization service is shown in Figure 13. This method takes OrganizationId as input and gives a list of Manager DTO’s as output.
4.8 Presentation Layer

Presentation Layer deals with writing JSP/Html pages as well as Controller Action classes that handle the server side logic of handling the user requests. eTime has several JSP, Html, Controller Action classes, FormBeans, DTO’s.

Shown in Figure 14 is an Employee DTO class. As we can see the DTO’s are flattened objects compared to Domain Objects which maintain the Object relationships structure. DTO’s are useful in displaying data in JSP pages, and by using them we don’t have to compromise on modifying the Domain layer objects to accommodate Presentation layer constraints.

```java
public List<ManagerDTO> getManagersByOrgId(int orgId){
    Organization org = orgRepository.findById(orgId);
    Set<Manager> mgrDOs = org.getManagers();
    ManagerDTOAssembler assembler = new ManagerDTOAssembler();
    List<ManagerDTO> mgrDTOs = new ArrayList<ManagerDTO>();

    for(Manager mgr:mgrDOs){
        mgrDTOs.add(assembler.toDTO(mgr));
    }
    return mgrDTOs;
}
```

Figure 13: Organization Service’s Method
More information about Presentation Layer follows in next sections.

4.9 Architecture of eTime

I used a multi-tiered architecture for the eTime project. One of the core principles of DDD is that the business logic should be isolated into its own domain layer and the domain objects should not deal with the presentation logic or the persistence logic. Based
on that I came up with a layered architecture with each layer having its own responsibility.

eTime architecture consists of Presentation Layer, Application Layer, Domain Layer and Infrastructure Layer.

a. Domain Layer:
This layer consists of domain objects like Entities, Value Objects, Factories and Aggregates and Repository interfaces. Business logic is present in the Entity objects. Repository implementations are part of Infrastructure layer.

b. Infrastructure Layer:
This layer consists of Repository Implementations. Repositories deal with retrieving the Domain objects and saving them. I used Hibernate for object persistence, so the Hibernate Api’s are used in the Repository implementations. Each of the Aggregates uses a specific Repository that performs various Create, Read, Update and Delete operations on the appropriate tables in the databases.
c. Application Layer:

This layer has Application Services. These perform stateless operations. Each Service method receives input in the form of Data Transfer objects (DTO) from the Presentation layer. The DTO is mapped to a Domain Object. This mapping between Domain and DTO objects are done in a separate Assembler class. A Domain object could be retrieved from...
Repository based on the Entity identity or saved to database by calling Repository methods.

The Domain Layer Objects methods are invoked to do some business logic as per the user request. Once all the processing is done the results are sent back to presentation layer in the form of a DTO object. This Domain to DTO mapping is needed to clearly separate the Presentation layer from Domain layer.

d. Presentation Layer:
All the JSP, HTML, Controller code will be present in this layer. The HTTPRequest from JSP page is sent to a Controller class. The Controller is responsible for handling various requests from the user and invoking Application Layer Services and based on the results forward to another JSP Page. Since I am using Java, Struts, the Controller is implemented as an Action Servlet. It contains no business logic. The user request is not processed here but just handed over to the appropriate Application Layer Service. This layer does not deal with Domain Objects at all. Domain objects are shielded from this layer. It only deals with Data Transfer Objects (DTO’s), which are flattened objects, easier to use in the JSPs for presentation.

4.10 Processing of a User Request from End to End in eTime:
Consider a scenario of adding a Manager to an Organization explained in the following sections.
a. User Clicks Add Button:

Admin enters the Manager details and selects the Organization for which he would belong from a drop down on the front end UI screen (a jsp page), and then clicks on ‘Add Manager’ button.

b. Request comes to Server Side Controller Class:

The submitted request is handled by the ‘ManagerController’ in the Presentation Layer. A method called saveManagerDetails() inside the ManagerController handles the request from the user.

```java
public class ManagerController extends DispatchAction{
    public ActionForward saveManagerDetails(ActionMapping mapping,
                                           ActionForm form, HttpServletRequest request, HttpServletResponse response) {
        try {
            OrganizationService orgService = new OrganizationService();
            ManagerForm mgrForm = (ManagerForm) form;
            int orgId = mgrForm.getOid();

            orgService.saveOrUpdateManagerDetails(mgrForm.getMgr(), orgId);

            List<ManagerDTO> mgrs = orgService.getManagersByOrgId(orgId);
            request.setAttribute("mgrs", mgrs);
            request.setAttribute("orgId", orgId);
        } catch (Exception e) {
            return mapping.findForward("error");
        }
        return mapping.findForward("showManagerAdminHomePage");
    }
}
```

Figure 16: Manager Controller’s Save Manager Details Method
The Controller method obtains the data submitted by user in form of ManagerForm object. The Controller then calls the Organization Service’s SaveOrUpdateManagerDetails method passing in the values submitted by the user.

c. Application Layer Service Method delegates work to Domain Object:

The Organization Service then makes use of Organization Repository to obtain the Organization domain object and then invokes the Organization Domain object’s addOrUpdateManagerUserInfo business method.

```java
public void saveOrUpdateManagerDetails(ManagerDTO mgr, int orgId) {
    Organization org = orgRepository.findById(orgId);
    ManagerDTOAssembler assembler = new ManagerDTOAssembler();
    org.addOrUpdateManagerUserInfo(assembler.toDO(mgr));

    orgRepository.attachDirty(org);
}
```

Figure 17: Organization Service's Method to Save Manager Details

Organization Domain object's business method contains the actual logic to add a Manager to an Organization is shown in Figure 18.
public void addOrUpdateManagerUserInfo(Manager mgr) {
    boolean isUpdate = mgr.getMgrid().equals(0) ? false : true;

    if (isUpdate) {
        for (Manager manager : this.managers) {
            if (manager.getMgrid().equals(mgr.getMgrid())) {
                manager.setTitle(mgr.getTitle());
                copyUserInfo(manager.getUserInfo(), mgr.getUserInfo());
            }
        }
    } else {
        this.managers.add(mgr);
    }
}

Figure 18: Organization Class - Business Method to Add a Manager

The OrganizationService then calls the OrganizationRepository in the Infrastructure
Layer to save the updated Organization object, which internally takes care of also saving
the Manager Object.

Organization Repository’s attachDirty method which takes care of saving an updated
Organization object is shown in Figure 19.
d. Controller sends back response to JSP page

The Controller makes another call to the Organization Service to retrieve list of all Managers for that particular Organization. The Objects obtained from Service is a list of Manager DTOs which are used by JSP page.

Controller then sends back response to the JSP view page to display the updated list of Managers, with the newly added Manager in that list.

As we can see, each layer is separated cleanly and each layer only deals with its appropriate logic.
4.11 Database Model

Figure 20: Database Schema Diagram of eTime
### 4.11.1 Organization Table

The Organization table contains the details about the clients of Relycom Inc.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Datatype</th>
<th>Size</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oid</td>
<td>int</td>
<td>10</td>
<td>PK</td>
<td>Auto incremental ID – Primary Key</td>
</tr>
<tr>
<td>name</td>
<td>varchar</td>
<td>45</td>
<td>Yes</td>
<td>Name of the client organization. This name is unique</td>
</tr>
<tr>
<td>address</td>
<td>varchar</td>
<td>45</td>
<td>No</td>
<td>Address of the client organization</td>
</tr>
<tr>
<td>address2</td>
<td>varchar</td>
<td>45</td>
<td>No</td>
<td>Address of the client organization</td>
</tr>
<tr>
<td>city</td>
<td>varchar</td>
<td>45</td>
<td>No</td>
<td>City</td>
</tr>
<tr>
<td>state</td>
<td>varchar</td>
<td>45</td>
<td>No</td>
<td>State</td>
</tr>
<tr>
<td>zip</td>
<td>varchar</td>
<td>25</td>
<td>No</td>
<td>Zipcode</td>
</tr>
<tr>
<td>email</td>
<td>varchar</td>
<td>45</td>
<td>No</td>
<td>Email address of the contact at client organization</td>
</tr>
<tr>
<td>phone</td>
<td>varchar</td>
<td>45</td>
<td>No</td>
<td>Phone number</td>
</tr>
<tr>
<td>version</td>
<td>bigint</td>
<td>20</td>
<td>No</td>
<td>Version number for Hibernate</td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp</td>
<td></td>
<td>No</td>
<td>Last updated timestamp</td>
</tr>
</tbody>
</table>

Table 1: Organization Table
4.11.2 Manager Table

The Manager table holds the details of all the Managers at the client Organization. It has a foreign key pointing to the Organization table’s Primary Key – OrganizationID.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Datatype</th>
<th>Size</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mgrid</td>
<td>int</td>
<td>10</td>
<td>PK</td>
<td>Auto incremental ID – Primary Key</td>
</tr>
<tr>
<td>organization_fk</td>
<td>int</td>
<td>10</td>
<td>Yes</td>
<td>Foreign key pointing to the OrganizationId in the Organization table</td>
</tr>
<tr>
<td>uid_fk</td>
<td>int</td>
<td>10</td>
<td>Yes</td>
<td>Foreign Key pointing to the user_id in the User_info table</td>
</tr>
<tr>
<td>version</td>
<td>bigint</td>
<td>45</td>
<td>No</td>
<td>Version number for Hibernate</td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp</td>
<td></td>
<td>Yes</td>
<td>Last updated timestamp</td>
</tr>
</tbody>
</table>

Table 2: Manager Table

4.11.3 Project Table:

The Project table contains the Project details.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Datatype</th>
<th>Size</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid</td>
<td>int</td>
<td>10</td>
<td>PK</td>
<td>Auto incremental ID – Primary Key</td>
</tr>
<tr>
<td>project_name</td>
<td>varchar</td>
<td>45</td>
<td>Yes</td>
<td>Name of the project</td>
</tr>
<tr>
<td>mgrid_fk</td>
<td>integer</td>
<td>10</td>
<td>Yes</td>
<td>The Manager of this project at client site</td>
</tr>
<tr>
<td>description</td>
<td>varchar</td>
<td>512</td>
<td>Yes</td>
<td>Project description</td>
</tr>
<tr>
<td>start_date</td>
<td>date</td>
<td></td>
<td>Yes</td>
<td>Start date of the project</td>
</tr>
<tr>
<td>end_date</td>
<td>date</td>
<td></td>
<td>Yes</td>
<td>Proposed end date of the project</td>
</tr>
<tr>
<td>version</td>
<td>bigint</td>
<td>20</td>
<td>No</td>
<td>Version number for Hibernate</td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp</td>
<td></td>
<td>Yes</td>
<td>Last updated timestamp</td>
</tr>
</tbody>
</table>

Table 3: Project Table
4.11.4 User_Info Table:

The User table holds personal information of all the users – Administrators, Managers, Employees.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Datatype</th>
<th>Size</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid</td>
<td>integer</td>
<td>PK</td>
<td></td>
<td>Auto incremental ID – Primary Key</td>
</tr>
<tr>
<td>username</td>
<td>varchar</td>
<td>30</td>
<td>Yes</td>
<td>Username for login purposes. This name is unique</td>
</tr>
<tr>
<td>password</td>
<td>varchar</td>
<td>20</td>
<td>Yes</td>
<td>Password field</td>
</tr>
<tr>
<td>first_name</td>
<td>varchar</td>
<td>45</td>
<td>Yes</td>
<td>First name of the user</td>
</tr>
<tr>
<td>middle_name</td>
<td>varchar</td>
<td>45</td>
<td>No</td>
<td>Middle name of the user</td>
</tr>
<tr>
<td>last_name</td>
<td>varchar</td>
<td>45</td>
<td>Yes</td>
<td>Last name of the user</td>
</tr>
<tr>
<td>address1</td>
<td>varchar</td>
<td>45</td>
<td>No</td>
<td>Address1 of the user</td>
</tr>
<tr>
<td>address2</td>
<td>varchar</td>
<td></td>
<td></td>
<td>Address2 of the user</td>
</tr>
<tr>
<td>city</td>
<td>varchar</td>
<td>45</td>
<td>No</td>
<td>City</td>
</tr>
<tr>
<td>state</td>
<td>varchar</td>
<td>45</td>
<td>No</td>
<td>State</td>
</tr>
<tr>
<td>zip</td>
<td>varchar</td>
<td>25</td>
<td>No</td>
<td>Zip code</td>
</tr>
<tr>
<td>email</td>
<td>varchar</td>
<td>45</td>
<td>No</td>
<td>Email address of the user.</td>
</tr>
<tr>
<td>cell_phone</td>
<td>varchar</td>
<td>20</td>
<td>No</td>
<td>Cell phone number</td>
</tr>
<tr>
<td>home_phone</td>
<td>varchar</td>
<td>20</td>
<td>No</td>
<td>Home phone number</td>
</tr>
<tr>
<td>work_phone</td>
<td>varchar</td>
<td>20</td>
<td>No</td>
<td>Work phone number</td>
</tr>
<tr>
<td>role</td>
<td>varchar</td>
<td>45</td>
<td>No</td>
<td>Role – Manager or Admin or Employee</td>
</tr>
<tr>
<td>active</td>
<td>tinyint</td>
<td>1</td>
<td>No</td>
<td>Status of the user – active or inactive</td>
</tr>
<tr>
<td>version</td>
<td>bigint</td>
<td>20</td>
<td>No</td>
<td>Version number for Hibernate</td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp</td>
<td></td>
<td>Yes</td>
<td>Last updated timestamp</td>
</tr>
</tbody>
</table>

Table 4: User Info Table
4.11.5 Employee Table

The Employee table holds the details of all the Employees working for Relycom.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Datatype</th>
<th>Size</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>empid</td>
<td>int</td>
<td>10</td>
<td>PK</td>
<td>Auto incremental ID – Primary Key</td>
</tr>
<tr>
<td>job_title</td>
<td>int</td>
<td>10</td>
<td>No</td>
<td>Job Title of the employee</td>
</tr>
<tr>
<td>billing_rate</td>
<td>int</td>
<td>10</td>
<td>No</td>
<td>Employee’s Billing Rate</td>
</tr>
<tr>
<td>skill_set</td>
<td>varchar</td>
<td>512</td>
<td>No</td>
<td>Skill set – like the technologies, Resume</td>
</tr>
<tr>
<td>uid_fk</td>
<td>int</td>
<td>10</td>
<td>No</td>
<td>Foreign key into the User_Info table where his information is stored</td>
</tr>
<tr>
<td>pid_fk</td>
<td>int</td>
<td>10</td>
<td>No</td>
<td>The project in which the Employee is currently working for. Field can be null.</td>
</tr>
<tr>
<td>version</td>
<td>bigint</td>
<td>20</td>
<td>No</td>
<td>Version number for Hibernate</td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp</td>
<td></td>
<td>Yes</td>
<td>Last updated timestamp</td>
</tr>
</tbody>
</table>

Table 5: Employee Table
4.11.6 Timesheet Table

This table holds all the timesheet information submitted by the various Employees of Relycom. It has a foreign key reference to the Employee table indicating to whom the timesheet belongs to. The timesheet table holds a list of timesheets saved by the users.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Datatype</th>
<th>Size</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tid</td>
<td>integer</td>
<td></td>
<td>PK</td>
<td>Auto incremental ID – Primary Key</td>
</tr>
<tr>
<td>empid_fk</td>
<td>varchar</td>
<td>45</td>
<td>Yes</td>
<td>Unique user id of the employee whose timesheet is being saved</td>
</tr>
<tr>
<td>start_date</td>
<td>varchar</td>
<td>45</td>
<td>Yes</td>
<td>Start date of the timesheet</td>
</tr>
<tr>
<td>end_date</td>
<td>varchar</td>
<td>45</td>
<td>Yes</td>
<td>End date of the timesheet</td>
</tr>
<tr>
<td>Monday_hours</td>
<td>float</td>
<td>45</td>
<td>Yes</td>
<td>Hours for Monday</td>
</tr>
<tr>
<td>Tuesday_hours</td>
<td>float</td>
<td>25</td>
<td>Yes</td>
<td>Hours for Tuesday</td>
</tr>
<tr>
<td>Wednesday_hours</td>
<td>float</td>
<td>45</td>
<td>Yes</td>
<td>Hours for Wednesday</td>
</tr>
<tr>
<td>Thursday_hours</td>
<td>float</td>
<td>20</td>
<td>Yes</td>
<td>Hours for Thursday</td>
</tr>
<tr>
<td>Friday_hours</td>
<td>float</td>
<td></td>
<td>Yes</td>
<td>Hours for Friday</td>
</tr>
<tr>
<td>Saturday_hours</td>
<td>float</td>
<td></td>
<td>Yes</td>
<td>Hours for Saturday</td>
</tr>
<tr>
<td>Sunday_hours</td>
<td>float</td>
<td></td>
<td>Yes</td>
<td>Hours for Sunday</td>
</tr>
<tr>
<td>total_hours</td>
<td>float</td>
<td></td>
<td></td>
<td>Total hours for a timesheet</td>
</tr>
<tr>
<td>status</td>
<td>varchar</td>
<td>45</td>
<td>Yes</td>
<td>Status of a time sheet – Saved, Submitted, Rejected, Approved, Processed</td>
</tr>
<tr>
<td>inv_amt</td>
<td>float</td>
<td></td>
<td>No</td>
<td>Invoice amount calculated based on the total hours and the billing rate of the Employee</td>
</tr>
<tr>
<td>inv_status</td>
<td>varchar</td>
<td>45</td>
<td>No</td>
<td>Status of the invoice – approved or not</td>
</tr>
<tr>
<td>version</td>
<td>bigint</td>
<td>20</td>
<td>No</td>
<td>Version number for Hibernate</td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp</td>
<td></td>
<td>No</td>
<td>Last updated timestamp</td>
</tr>
</tbody>
</table>

Table 6: Timesheet Table
4.11.7 AuditLog Table

Audit log table holds the details of any changes to the Timesheet by any user in the System.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Datatype</th>
<th>Size</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aid</td>
<td>integer</td>
<td>PK</td>
<td></td>
<td>Auto incremental ID – Primary Key</td>
</tr>
<tr>
<td>tid_fk</td>
<td>integer</td>
<td>10</td>
<td>Yes</td>
<td>Time sheet Id to which this log corresponds to</td>
</tr>
<tr>
<td>logged_by_uid_fk</td>
<td>integer</td>
<td>Yes</td>
<td></td>
<td>User ID of the user making change to the Timesheet</td>
</tr>
<tr>
<td>notes entered</td>
<td>varchar</td>
<td>50</td>
<td>No</td>
<td>Any comments entered by the user</td>
</tr>
<tr>
<td>system_comment</td>
<td>varchar</td>
<td>45</td>
<td>No</td>
<td>System comment is entered based on whether the timesheet got ‘SAVED,’ ‘SUBMITTED,’ ‘APPROVED,’ ‘REJECTED,’ ‘PROCESSED’</td>
</tr>
<tr>
<td>version</td>
<td>bigint</td>
<td>20</td>
<td>No</td>
<td>Version number for Hibernate</td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp</td>
<td></td>
<td>No</td>
<td>Last updated timestamp</td>
</tr>
</tbody>
</table>

Table 7: Audit Log Table
Chapter 5
WEB INTERFACE DESIGN

5.1 User Interfaces for Admin

5.1.1 Login Page

Figure 21: Login Page

The login page shown in Figure 21 is common to all the users of the application.
5.1.2 Organization Details Page

The Admin can either add a new Organization or choose from an existing list of Organizations and view/update the information.

![Organization Details Page](image)

Figure 22: Organization Details Page
5.1.3 Manager Details Page

The Admin can choose an Organization from a list and then add a new Manager or choose an existing Manager of that Organization, to view details.

Figure 23: Manager Details Page
5.1.4 Project Details Page

The Admin can choose an Organization and a Manager and add a Project to that Manager.

Figure 24: Project Details Page
5.1.5 Employee Details Page

The Admin can add a new Employee or view the details of an existing Employee.

Figure 25: Employee Details Page
5.1.6 Search Timesheet Page

The Admin can search for timesheets based on certain criteria. The Employee dropdown contains list of all Employees of Relycom.

Figure 26: Search Timesheet Page
5.1.7 Search Based Timesheet Detail Page

On clicking a link on search result, the page in Figure 27 is displayed. The Admin can either Approve it or Reject it or change it to Processed Status.

Figure 27: Search Based Timesheet Details Page
5.1.8 Reports Home Page

The Admin can choose to generate an invoice report. Currently, there is only one report – Invoice report, but in future there might be more reports.

Figure 28: Report Home Page
5.1.9 Invoice Generation

The Admin needs to choose a specific employee for which he plans to generate an invoice and submit the page.

Figure 29: Invoice - Select Employee Page
A list of client-approved timesheets are displayed for which the Admin needs to generate the Invoice. Admin can choose some timesheets and submit the page.

Figure 30: Invoice Timesheet Page
The invoice report page is shown in Figure 31. It lists various details like the client’s name, contact information, client manager, Relycom’s contact information, project name, consultant name, timesheet details, billing details and the total invoice amount.

Figure 31: Invoice Report Page
5.2 User Interfaces for Manager

5.2.1 Search Timesheet Page for Manager

A Manager can search for timesheets based on certain criteria. The Employee drop down here shows only the Employees reporting to the specific Manager who is logged in.

![Search Timesheet Page for Manager](image)

Figure 32: Search Timesheet Page for Manager
5.2.2 Search Based Timesheet Detail Page

On clicking a link on search result, the page in Figure 29 is displayed. Manager can then verify the timesheet details and can either ‘Approve’ the timesheet or ‘Reject’ the timesheet and also enter his comments in the timesheet page.

![Search Based Timesheet Details Page for Manager](image)

Figure 33: Search Based Timesheet Details Page for Manager
5.2.3 Timesheet Detail Page After Manager Approval

Once the Manager approves the timesheet the page in Figure 30 is shown.

![Timesheet Details Page After Manager Approval](image)

Figure 34: Timesheet Details Page After Manager Approval
5.3 User Interfaces for Employee

5.3.1 Timesheet Page

The Employee can enter the number of hours worked in a week in the timesheet page.

![Employee Timesheet Page](image)

Figure 35: Employee Timesheet Page
5.3.2 Timesheet Detail Page After Employee Submission

Once the Employee submits the timesheet the page in Figure 32 is shown.

Figure 36: Employee Timesheet Page After Submission
Chapter 6

CONCLUSION AND FUTURE WORK

This application eTime automates the timesheet approval workflow and reduces the burden on the administrator as well as the consultants of Relycom. This project was developed using Java, Hibernate, Struts frameworks using multi-layered architecture. These technologies are industry standard technologies, which are used to develop highly scalable and reliable applications. One of the main goals of this project was also to learn about Domain Driven Design (DDD) methodology and follow it to design the domain model of the project. Eric Evans is widely regarded as the father of Domain Driven Design. His book, ‘Domain-Driven Design: Tackling Complexity in the Heart of Software’, helped me a great deal in understanding DDD concepts.

One of the other concerns for the client, Relycom Inc. was cost involved in developing and maintaining the application in future. That was one of the main reasons for developing the application using various open source technologies and tools like Apache Tomcat server, MySql database, Java and Eclipse IDE.

This application could be further enhanced in future. Currently it only supports HTML based Invoice report, but in future multiple other reports could be added and exported to PDF. In addition to timesheet tracking and approval, we could add employee time off
functionality, employee submitting expense reports when working for a client. In addition we could also integrate this system with project management software or ERP software.
APPENDIX

Database Create Scripts

CREATE DATABASE `TimeSheetDB`;

DROP TABLE IF EXISTS `TimeSheetDB`.`ORGANIZATION`;
CREATE TABLE  `TimeSheetDB`.`ORGANIZATION` (
  `OID` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT 'unique id',
  `NAME` varchar(45) NOT NULL COMMENT 'name of client org',
  `ADDRESS1` varchar(45) DEFAULT NULL,
  `CITY` varchar(45) DEFAULT NULL,
  `STATE` varchar(45) DEFAULT NULL,
  `ZIP` varchar(25) DEFAULT NULL,
  `EMAIL` varchar(45) DEFAULT NULL,
  `PHONE` varchar(45) DEFAULT NULL,
  `VERSION` bigint(20) unsigned NOT NULL,
  `ADDRESS2` varchar(45) DEFAULT NULL,
  `TIMESTAMP` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
PRIMARY KEY (`OID`),
UNIQUE KEY `Index_2_NAME` (`NAME`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1
COMMENT='Org details';

DROP TABLE IF EXISTS `TimeSheetDB`.`MANAGER`;
CREATE TABLE  `TimeSheetDB`.`MANAGER` (  `MGRID` int(10) unsigned NOT NULL,
  `ORGANIZATION_FK` int(10) unsigned DEFAULT NULL,
  `VERSION` bigint(20) unsigned DEFAULT NULL,
  `UID_FK` int(10) unsigned DEFAULT NULL,
  `TIMESTAMP` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
PRIMARY KEY (`MGRID`),
UNIQUE KEY `Index_unique_uid_4` (`UID_FK`),
KEY `FK_MANAGER_ORG_ID_1` (`ORGANIZATION_FK`),
CONSTRAINT `FK_MANAGER_2` FOREIGN KEY (‘UID_FK’) REFERENCES ‘USER_INFO’ (‘UID’),
CONSTRAINT `FK_MANAGER_ORG_ID_1` FOREIGN KEY (‘ORGANIZATION_FK’) REFERENCES ‘ORGANIZATION’ (‘OID’)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT= 'Manager Info';

DROP TABLE IF EXISTS `TimeSheetDB`.‘PROJECT’;
CREATE TABLE  `TimeSheetDB`.‘PROJECT’ (
   `PID` int(10) unsigned NOT NULL AUTO_INCREMENT,
   `PROJECT_NAME` varchar(45) NOT NULL,
   `DESCRIPTION` varchar(512) DEFAULT NULL,
   `START_DATE` date NOT NULL,
   `END_DATE` date DEFAULT NULL,
   `MGRID_FK` int(10) unsigned DEFAULT NULL,
   `VERSION` bigint(20) unsigned DEFAULT NULL,
   `TIMESTAMP` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
   PRIMARY KEY (`PID`),
   KEY `FK_PROJECT_MGR_ID_1` (‘MGRID_FK’) USING BTREE,
   CONSTRAINT `FK_PROJECT_MGRID_1` FOREIGN KEY (‘MGRID_FK’)
REFERENCES ‘MANAGER’ (‘MGRID’)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=latin1
COMMENT=' Project Info';

DROP TABLE IF EXISTS `TimeSheetDB`.‘EMPLOYEE’;
CREATE TABLE  `TimeSheetDB`.‘EMPLOYEE’ (  
   `EMPID` int(10) unsigned NOT NULL AUTO_INCREMENT,
   `JOB_TITLE` varchar(45) DEFAULT NULL,
   `BILLING_RATE` float DEFAULT '0',
   `SKILL_SET` varchar(512) DEFAULT NULL,
   `PID_FK` int(10) unsigned DEFAULT NULL,
   `VERSION` bigint(20) unsigned DEFAULT NULL,
   `UID_FK` int(10) unsigned DEFAULT NULL,
   `TIMESTAMP` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
   PRIMARY KEY (‘EMPID’),
   UNIQUE KEY `Index_unique_uid_4` (‘UID_FK’),
   KEY `FK_EMPLOYEE_PID_2` (‘PID_FK’),
   ) ENGINE=InnoDB DEFAULT CHARSET=latin1
CONSTRAINT `FK_EMPLOYEE_PID_2` FOREIGN KEY (`PID_FK`) REFERENCES `PROJECT` (`PID`),
CONSTRAINT `FK_EMPLOYEE_UID_3` FOREIGN KEY (`UID_FK`) REFERENCES `USER_INFO` (`UID`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1
COMMENT='Employee info';

DROP TABLE IF EXISTS `TimeSheetDB`.'TIMESHEET';
CREATE TABLE  `TimeSheetDB`.'TIMESHEET` (  
    `TID` int(10) unsigned NOT NULL AUTO_INCREMENT,  
    `EMPID_FK` int(10) unsigned DEFAULT NULL,  
    `START_DATE` date NOT NULL,  
    `END_DATE` date NOT NULL,  
    `MONDAY_HOURS` float DEFAULT '0',  
    `TUESDAY_HOURS` float DEFAULT '0',  
    `WEDNESDAY_HOURS` float DEFAULT '0',  
    `THURSDAY_HOURS` float DEFAULT '0',  
    `FRIDAY_HOURS` float DEFAULT '0',  
    `SATURDAY_HOURS` float DEFAULT '0',  
    `SUNDAY_HOURS` float DEFAULT '0',  
    `TOTAL_HOURS` float DEFAULT NULL,  
    `STATUS` varchar(50) NOT NULL,  
    `VERSION` bigint(20) unsigned DEFAULT NULL,  
    `INV_AMT` float DEFAULT NULL,  
    `INV_STATUS` varchar(45) DEFAULT NULL,  
    `TIMESTAMP` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
    PRIMARY KEY (`TID`),  
    UNIQUE KEY `composite_1` (`EMPID_FK`, `START_DATE`, `END_DATE`),  
    CONSTRAINT `FK_TIMESHEET_EMPID_1` FOREIGN KEY (`EMPID_FK`) REFERENCES `EMPLOYEE` (`EMPID`)  
) ENGINE=InnoDB AUTO_INCREMENT=13 DEFAULT CHARSET=latin1
COMMENT='Timesheet Info';

DROP TABLE IF EXISTS `TimeSheetDB`.'AUDIT_LOG';
CREATE TABLE  `TimeSheetDB`.'AUDIT_LOG` (  
    `AID` int(10) unsigned NOT NULL AUTO_INCREMENT,  
    `TID_FK` int(10) unsigned DEFAULT NULL,  
    `LOGGED_BY_UID_FK` int(10) unsigned DEFAULT NULL,  
    `LOGGED_TIMESTAMP` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
    PRIMARY KEY (`AID`),  
    CONSTRAINT `FK_AUDIT_LOG_TID` FOREIGN KEY (`TID_FK`) REFERENCES `TIMESHEET` (`TID`)  
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1
COMMENT='Audit Log Info';
`NOTES_ENTERED` varchar(1000) DEFAULT NULL,
`SYSTEM_COMMENT` varchar(100) NOT NULL,
`VERSION` bigint(20) unsigned DEFAULT NULL,
`TIMESTAMP` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

PRIMARY KEY ("AID"),
KEY `FK_AUDIT_LOG_TID_FK1` ("TID_FK"),
KEY `FK_LoggedBy_UID_2` ("LOGGED_BY_UID_FK"),
CONSTRAINT `FK_AUDIT_LOG_TID_FK1` FOREIGN KEY ("TID_FK")
REFERENCES 'TIMESHEET' ("TID"),
CONSTRAINT `FK_LoggedBy_UID_2` FOREIGN KEY ("LOGGED_BY_UID_FK")
REFERENCES `USER_INFO` ("UID")
) ENGINE=InnoDB AUTO_INCREMENT=50 DEFAULT CHARSET=latin1
COMMENT='Timesheet history';

DROP TABLE IF EXISTS 'TimeSheetDB'.`USER_INFO';
CREATE TABLE  `TimeSheetDB`.`USER_INFO` (  
`UID` int(10) unsigned NOT NULL AUTO_INCREMENT,
`USERNAME` varchar(45) NOT NULL,
`PASSWORD` varchar(45) NOT NULL,
`ROLE` varchar(45) NOT NULL,
`ACTIVE` tinyint(3) unsigned DEFAULT '1',
`FIRST_NAME` varchar(45) NOT NULL,
`MIDDLE_NAME` varchar(45) DEFAULT NULL,
`LAST_NAME` varchar(45) NOT NULL,
`ADDRESS1` varchar(45) DEFAULT NULL,
`ADDRESS2` varchar(45) DEFAULT NULL,
`CITY` varchar(45) DEFAULT NULL,
`STATE` varchar(45) DEFAULT NULL,
`ZIP` varchar(15) DEFAULT NULL,
`CELL_PHONE` varchar(12) DEFAULT NULL,
`WORK_PHONE` varchar(12) DEFAULT NULL,
`HOME_PHONE` varchar(12) DEFAULT NULL,
`EMAIL` varchar(45) DEFAULT NULL,
`VERSION` bigint(20) unsigned DEFAULT NULL,
`TIMESTAMP` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

PRIMARY KEY ("UID") USING BTREE,
UNIQUE KEY 'Index_Unique_Username_2' ('USERNAME')
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=latin1
COMMENT='User personal info';
BIBLIOGRAPHY


Available:

http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/JSPIntro.html


Available:

http://en.wikipedia.org/wiki/HTML


Available:

http://dev.mysql.com/tech-resources/articles/storage-engine/part_1.html


Available:

http://download.oracle.com/javaee/6/tutorial/doc/


Available:

http://www.infoq.com/minibooks/domain-driven-design-quickly


Available:

http://dddsample.sourceforge.net/
[8] InfoQ: Domain Driven Design and Development In Practice [Online]

Available:

http://www.infoq.com/articles/ddd-in-practice


Available:


[10] Struts 1 Welcome [Online]

Available:

http://struts.apache.org/1.x/


Available:


[12] JUnit [Online]

Available:

http://junit.sourceforge.net/


Available:

http://www.roseindia.net/struts/StrutsArchitecture.shtml


Available: