Software Design

Specification

BLEND

Clients Management System

Trajanov Tosho

June, 2017

1. Introduction

1.1 Purpose

The purpose of this document is to present a detailed description of the Clients Management System. It will explain the design of the features within the system. It contains specific information about the expected input, output, classes, and functions.

This Design Specification is to be used by Software Engineering and Software Quality Engineering as a definition of the design to be used to implement the Clients Management System.

1.2 Scope

The Software design document would demonstrate how the design will accomplish the functional and non - functional requirements captured in the Software Requirement specification (SRS). The document will provide a framework to the programmers through describing the high level components and architecture, sub systems, interfaces, database design and algorithm design. This is achieved through the use of architectural patterns, design patterns, sequence diagrams, class diagrams, relational models and user interfaces.

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition
Employee	A person that is using the system for tracking his daily hours.
Software Requirements Specification	A document that completely describes all of the functions of a proposed system and the constraints under which it must operate. For example, this document.
Stakeholder	Any person with an interest in the project who is not a developer.

User	A person that is using the system.
SDS	System Design Specification
SRS	Software Requirements Specification
Database	A collection of stored related data
Architectural Design	Establishing the overall structure of software system
CRUD	Create, Read, Update, Delete

1.4 References

- *Software Requirements Specification BLEND Clients Management System, Trajanov Tosho, February 2017*
- *A Software Design Specification Template, Brad Appleton <brad@bradapp.net> <http://www.bradapp.com/docs/sdd.html>*

2. System Overview

This software system is a low cost web tool for managing clients for a software development company. The system is designed to maximize the manager's productivity by providing tools for generating quotes, invoices as well as reviewing team's productivity.

The system allows the managers to create projects the employees are working on, use the logged time by the employees on a similar projects to make better estimations and planning in the future. Additionally, the system allows the managers to be able to prepare quotes and invoices without too much effort.

3. Design Considerations

3.1. Assumptions and Dependencies

- Blend shall support a minimum of 50 users.
- The company will be testing the system with their own test data sets.

3.2. General Constraints

- All server-side scripts shall be written in PHP 5.6 or above.
- All HTML code shall conform to the HTML 4.0 standard
- Blend shall use the MySQL database engine.

3.3. Goals and Guidelines

- Emphasis shall be placed on Usability as the User Interface will be used by users without much training.
- The system must be fully functional within the scheduled time frame.

3.4. Development Methods

This project is being conducted using a modified waterfall paradigm. The development process is formal with document and code reviews.

4. Architectural Strategies

The architecture and design has been influenced by the following design decisions and strategies:

- The design uses Object Oriented Principles (OOP). The trade-off of increased code overhead and object message passing is considered justified by the gain of modularization of functionality, data encapsulation, communication through interfaces and re-use through polymorphism.
- The overall system is designed upon a client-server approach. This is an established and well-understood architecture.
- The system uses the MVC architectural pattern.
- Authentication will be limited to password checking on initial login and a session ID subsequently.
- Communication with the users will be by the HTML protocol as this is well-supported by the selected browsers and PHP.
- PHP and MySQL were selected because they had the necessary capabilities to provide the needed services to the user and their GNU licenses will reduce product cost.
- The system is built on top of PHP framework Laravel 5.2.
- The system uses few external packages installed through PHP composer.

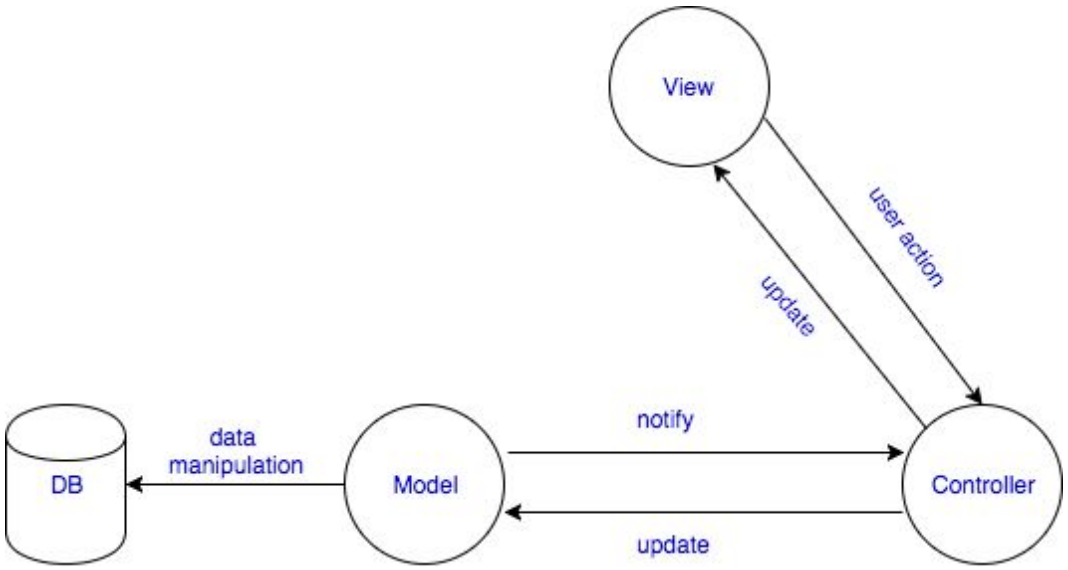
5. System Architecture

5.1 High level components and interfaces

5.1.1 Components

The system is designed to be easily extendable and organized in modules, where every module follows the MVC itself. Extending the system in future can be easily done without altering the existing structure, i.e. by adding additional module to the current implementation.

Being based on the MVC architectural pattern, the system has four major components: Database, Model, View and Controller.



5.1.1.1 Database

The system shall use a relational database to satisfy the need of keeping track of object relations. The selected database engine is MySQL InnoDB.

5.1.1.2 Model

The model component is an object representation of a database table. It provides methods for CRUD operations to the database that can be easily called through the business logic of the system, i.e. the Controller.

5.1.1.3 Controller

The controller component covers the business logic of the application. It's responsible for getting the data from the Model and passing it to the View, and getting the user interactions made through the View and sending an update to the model accordingly.

5.1.1.4 View

The view component is the presentational layer of the system. It provides the user with an interface to interact with the system.

5.1.2 User Interfaces

5.1.2.1 User authentication

This is the screen that appears whenever an unauthenticated user tries to access the Client Management System. It handles the part responsible for the security of the system.

5.1.2.2 Dashboard

This is the screen that appears right after login. It shall give the admin an graphical overview of the time log activities for all users.

5.1.2.3 Time Log

This is the page where the admin can list all time log entries, add a new entry, edit an existing one or check its details, and delete a time log when appropriate. When adding a new entry, or editing an existing one, the admin should be able to easily select the project the log is referring to from a dropdown list.

5.1.2.4 Clients

This is the page where all clients records are listed, together with a general overview of their current balance. From here, the admin can open a detailed view of the client profile and check their main details, the past and pending invoices and the quotes issued to that particular client.

Both through the detailed view, and the list of all clients, the admin can easily access the edit page and update the client details in the database, as well as delete the client if appropriate.

5.1.2.5 Invoices

All invoices are added through this interface, no matter what client they refer to. The main page of the Invoices interface is the list of all invoices, showing their status, balance and allowing for quick access to both the invoice and client details.

From the invoice list, the admin has the options to edit the invoice, delete it, or download it in PDF.

Both the add and the edit feature are based on the same functionality and allow the user to add separate invoice items easily, setup the status, currency, due date and other similar details.

5.1.2.6 Quotes

Similarly to the invoices, this is the page where all quotes are managed no matter what client they refer to. The main page is a list of all Quotes showing the status, client and the amount of the proposal. Again, the available options for each quote are edit, delete, and PDF download.

The interface for adding/editing a quote is similar to the one for adding/editing an invoice. The admin has the option to add separate items, set a currency and other general details for the quote.

5.1.2.7 Settings

The settings interface allows the admin to setup the basic details needed for the system to function. It includes four sections:

1. **Projects**
   This is the interface for setting up the projects that will be used for adding time logs in the section described above. The projects interface is similar to all of the other interfaces in the system and includes a list of all projects, and options to add, edit or delete a project.
2. **Employees**
   This is the interface for setting up the employee references used again in the time log section for presenting which employee worked on the respective task. It also follows the general structure of the system and has a list of all employees and the options to add, edit or delete an employee.
3. **Taxes**
   This interface shows a list of taxes that can be used when creating an invoice or a quote.
4. **Currencies**
   This interface shows a list of the available currencies that can be used throughout the system.

7. Detailed System Design

7.1 Database

Type	Database
Purpose	Provides the means of data management and storage for the system.
Interfaces	The initial setup shall be done by an administrative user and shall include the creation of the database. The creation of the database tables and all modifications of the database shall be performed using Laravel's migrations.
Description	The chosen database is MySQL, with InnoDB storage engine for relational databases. The database will be on the same server with the system in order to set up a quick local connection and serve the user properly. The database receives an SQL query from the system and based on it performs the CRUD operations to the data.

7.2 Model

Type	Class
Purpose	Provides the object oriented interface for communicating with the database and executing operations requested by the user.
Interfaces	All model classes will extend Laravel's Model class described in their documentation (for more information follow this link https://laravel.com/api/5.2/Illuminate/Database/Eloquent/Model.html). The general structure of a model class will look something like this: `namespace App\Models;`

	`use Illuminate\Database\Eloquent\Model;`
	
	`class Client extends Model`
	`{`
	`protected $table = 'clients';`
	
	`public function projects()`
	`{`
	`return $this->belongsTo(Project::class);`
	`}`
	`}`

Description	The model class automatically maps the table based on its name, but it can also be manually set by the user using the protected attribute "table". It has a timestamp log working in the background which automatically picks up the time a record is created or updated and alters the database accordingly. Additionally, it allows for easy implementation of soft deletes, by simply adding a column in the database table and using the `SoftDeletingTrait` in the model. The relations with other tables can be easily represented in the model using Laravel's `belongsTo`, `hasOne`, `hasMany`, and `belongsToMany` functions. The model then makes it extremely easy to access all the related data just by calling one function.

7.3 Controller

Type	Class
Purpose	Holds the business logic of the system that is responsible for interpreting user interactions, updating the database accordingly, and passing initial data to the user based on their selection.
Interfaces	All controller classes will extend Laravel's Controller class described in their documentation (for more information follow this link https://laravel.com/api/5.2/Illuminate/Routing/Controller.html). The general structure of a controllers should follow the RESTful architecture.

```
namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;

class ClientController extends Controller
{
    public function __construct()
    {
        //
    }
    public function index()
    {
        //
    }
    public function create()
    {
        //
    }
    public function store($id, Request
$request)
    {
        //
    }
    public function show($id)
    {
        //
    }
    public function edit($id)
    {
        //
    }
    public function update($id, Request
$request)
    {
        //
    }
}
```

Description	The controller class will have all the needed methods for providing the

	wanted functionality to the user. To prevent user errors, the controller will use Laravel's `FormRequest` class for validating user input. Additionally, for database actions, the Controller will work with the Repository pattern and communicated with classes that handle the actual interaction with the Models. That is done in order to provide extra flexibility and extendability of the system.

7.4 View

Type	User Interface
Purpose	The view is the presentational layer of the system.
Interfaces	The view shall use the Blade templating engine for presenting dynamic data.
Description	The view provides the interface the user needs for interacting with the system. One general layout shall be created where all shared data will be appropriately sent, and all views used throughout the system shall extend that layout.