payday

Team Rocket
Payday, Release 00.2 Alpha Web

Software System Plan Documentation
Revision 01

Team Lead: *Andrew Rutherford*

Suraj Sequeira, Justin Patel, Sheehan Toufiq, Benjamin Byrd

Specification Version: 1.0
Published: 2012
Author: Team Rocket

Version Control History

Version	Primary Author(s)	Description of Version	Date Completed
Draft	Sheehan Toufiq	Initial Commit	2014NOV14

Requirements Analysis Document

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Scope of the system
   - 1.3 Objectives and success criteria of the project
   - 1.4 Definitions, acronyms, and abbreviations
   - 1.5 References
   - 1.6 Overview
2. Current system
3. Proposed system
   - 3.1 Overview
   - 3.2 Functional requirements
   - 3.3 Nonfunctional requirements
     - 3.3.1 Usability
     - 3.3.2 Reliability
     - 3.3.3 Performance
     - 3.3.4 Supportability
     - 3.3.5 Implementation
     - 3.3.6 Interface
     - 3.3.7 Packaging
     - 3.3.8 Legal
   - 3.4 System models
     - 3.4.1 Scenarios
     - 3.4.2 Use case model
     - *3.4.3 Object model*
     - *3.4.4 Dynamic model*
     - 3.4.5 User interface—navigational paths and screen mock-ups
4. Glossary

Requirements Analysis Document

1. Introduction
2. Current system
3. Proposed system
    - 3.1 Overview
    - 3.2 Functional requirements
    - 3.3 Nonfunctional requirements
    - 3.4 System models
        - 3.4.1 Scenarios
        - 3.4.2 Use case model
        - 3.4.3 Object model
            - 3.4.3.1 Data dictionary
            - 3.4.3.2 Class diagrams
        - 3.4.4 Dynamic models
        - 3.4.5 User interface—navigational paths and screen mock-ups
4. Glossary

System Design Document

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Design goals
   - 1.3 Definitions, acronyms, and abbreviations
   - 1.4 References
   - 1.5 Overview
2. Current software architecture
3. Proposed software architecture
   - 3.1 Overview
   - 3.2 Subsystem decomposition
   - 3.3 Hardware/software mapping
   - 3.4 Persistent data management
   - 3.5 Access control and security
   - 3.6 Global software control
   - 3.7 Boundary conditions
4. Subsystem services

   Glossary

Object Design Document

1. Introduction
   - 1.1 Object design trade-offs
   - 1.2 Interface documentation guidelines
   - 1.3 Definitions, acronyms, and abbreviations
   - 1.4 References
2. Packages
3. Class interfaces

Glossary

Test Case Specification

1. Test case specification identifier
2. Test items
3. Input specifications
4. Output specifications
5. Environmental needs
6. Special procedural requirements
7. Intercase dependencies

Test Plan

1. Introduction
2. Relationship to other documents
3. System overview
4. Features to be tested/not to be tested
5. Pass/Fail criteria
6. Approach
7. Suspension and resumption
8. Testing materials (hardware/software requirements)
9. Test cases
10. Testing schedule

PayDay

Payday is a web-based accounting and employee management system developed for small businesses. The implementation of this system is expected to be completed by December 3, 2014.

Payday shall perform common accounting functions and keep track of employees. The application will provide a login page for users. Upon user request Payday shall generate expense reports based upon transactions stored in the database. Payday shall provide functions for maintaining employee records. Payday shall provide functions for adding new employees, viewing/editing employee information, and deleting employee records. Payday shall provide the managing user functions for managing employee payment. Payday shall provide a dashboard pane for performing employment pay functions. Payday shall provide functions for creating employee checks/pay stubs, tracking employee hours, editing employee history, and employee record viewing. Payday shall provide the process of gathering the total amount of hours each employee has worked in a shift by getting the difference between their clock in and clock out time.

Payday shall allow managers to be given access to alter/edit employee previous clock in and clock out times, number of hours worked by each employee, and quarterly employee pay stubs. Payday shall allow managers to view daily revenues. Payday shall allow the manager and employees to change and edit their credentials. If an employee or manager want to change their password, user ID, or image, Payday will allow the user to alter the information he wants to change. Payday shall allow managing users to view a breakdown of year-to-date taxes to be paid from past transactions, including expenses and revenue. Payday shall allow employees and managers to view a breakdown of each employee's hourly work and payment history.

Payday system requires google chrome web browser to run properly.

Rationale:

Currently there are some 5.6 million small businesses in the United States. Collectively, small business spent over $1 billion on accounting services. Small business accounting is a cumbersome task. We think we can make this task both easy and affordable. Payday will be a web-based small business accounting and employee management system.

Andrew Rutherford

8/26/14

CSC 4350

Homework1

Background:

I am an undergraduate student at Georgia State University with a major in Computer Science, and a concentration in software systems.

Programming Knowledge:

Through my coursework in computer science I have become well-versed in object oriented programming languages. I am most proficient with Java, but I can also use C++ and C. In high school and college I have taken web-programming course, so I am familiar with HTML, PHP, Javascript, and Ruby.

Software Development:

I have yet to attain an internship in the field of software development. Through my web programming course I was exposed to the various steps needed to successfully complete a computer project.

Interest:

My primary interest is in the field of software development. My next two areas of interest would be web programming and database management respectively.

Take aways:

I would like to gain experience developing an application, and I want to improve my programming skills

Sheehan Toufiq

220 Brady Walk
Lawrenceville, GA 30046
(770) 298-0791
sheehantoufiq@gmail.com

Education

Georgia State University
Bachelor of Science, Computer Science: *Currently Enrolled*
Expected Graduation Date: May 2015

Employment History

Peach Belle, Atlanta, GA - *http://peachbelleboutique.com*
Advisor, August 2014 – Present

- Advised Founder on product branding, fashion merchandising, and importing.
- Developed ecommerce solution to handle online orders.

Eventluv, Atlanta, GA - *http://eventluv.com*
Founder, September 2013 – Present

- Built a minimum viable SaaS product to help event organizers with ticketing, promotion, and analytics.
- Ran marketing campaigns on Google Adwords to test validity of product.
- Worked with alpha testers to refine product.
- Helped clients set up prototype for their events.

Redwood Brands, Atlanta, GA
Founder & CEO, September 2012 – Present

- Sourced products and negotiated with foreign suppliers.
- Used arbitrage to take advantage of product price differences in various markets.
- Branded, imported, and sold products wholesale and direct to consumer by white labeling.
- Brought multiple ecommerce brands to profitability using affiliate marketing and white labeling.
- Set up, managed, and marketed multiple ecommerce platforms for generic and white label products.
- Developed in-house software and APIs to help streamline certain processes.

Face the Fans, Atlanta, GA
Development Intern, October 2011 – August 2012

- Assisted in the development of an online social game.
- Used open social APIs to build incentivized gamification techniques.
- Designed graphics, resources, and user experiences/interfaces.

Toufiq Media, LLC, Atlanta, GA
Founder & CEO, December 2008 – January 2012

- Freelanced graphics, web development, and custom software to local businesses, startups, and other clients.
- Helped clients build a following and make sales through social media marketing.
- Handled all accounting, preparing invoices, and service to clients.

Professional Skills

JavaScript: *Intermediate*, **Java**: *Intermediate*, **C++**: *Beginner*, **Linux & Unix Command Line**: *Intermediate*, **PHP**: *Intermediate*, **HTML**: *Advanced*, **CSS**: *Intermediate*, **SQL**: *Intermediate*, **Photoshop**: *Intermediate*, **Python**: *Intermediate,* **Spreadsheets**: *Intermediate*

Hobbies & Interests

- Learning
- Reading
- Programming
- Entrepreneurship
- Testing various business ideas.

References

Musaddeq Khan
Spoor
(404) 429-8177
musaddeqkhan@gmail.com

Nafisa Ahmed
Peach Belle
(770) 905-4644
nafisa.ahmed92@gmail.com

Troy Presley
Akamai Technologies, Face the Fans
(404) 583-7124
troy@facethefans.com

Dean Gebert
Face the Fans
(404) 931-6111
dean@facethefans.com

Suraj Sequeira

8580 Merion Drive
Duluth, GA 30097
ssequeira17@gmail.com
(678) 392-6595

EDUCATION

Georgia State University, Atlanta, GA
Bachelor of Science, Computer Science May 2015

- Hope Scholarship recipient

COMPUTER SCIENCE EXPERIENCE

Fulton County IT Department, Atlanta, GA May 2012- August 2012
Intern/PC Assistant

- Assisted in sysprep desktops with Windows 7/XP
- Aided in clearing viruses from infected computers.
- Imaged and re-imaged PC's
- Backed up data for the departments within the Fulton County Building

Omega Bio-Tek, Norcross, GA May 2011- July 2011
Intern/Assistant to the Assistant Manager

- Collaborated with the shipping and logistics teams for the organization.
- Tracked and accounted for inventory in the warehouse by utilizing software including Microsoft Excel.

ADDITIONAL EXPERIENCE

OCA, Atlanta, GA January 2012 - May 2012
Intern/Legislative Aide

- Shadow lobbyist Larry Pelligrini.
- Serve as a Legislative Aide to GA State Representative Pedro Marin.
- Learned how legislation is created from a concept stage to becoming law.
- Interacted with several lawmakers and their aides, lobbyist, and citizens.

SKILLS

- Proficient in utilizing MS Office.
- Highly skilled at utilizing the web as a research tool; knowledge of social media tools.
- Ability to troubleshoot hardware infrastructure.

COMMUNITY INVOLVEMENT

Youth Leadership Forsyth, Cumming GA August 2009 –May 2011
Mentor Me, Cumming GA January 2011-May 2011
Mission Atlanta, Cumming GA May 2010

Benjamin Edward Byrd

(770) 856-6214
bebyrd@gmail.com

Present Address:
440 McBride Road
Fayetteville, GA 30215

EDUCATION

Georgia State University, Atlanta GA **June 2013-Present**
College of Arts and Sciences
Bachelor of Science, Computer Science

The University of Georgia, Athens, GA **May 2010**
College of Family and Consumer Sciences
Bachelor of Science in Family and Consumer Sciences

RELATED INVOLVEMENT

Member, Griffin Campus Technology Committee **August 2010 – Present**
The University of Georgia, Griffin Campus *Griffin, GA*

- Provided direction on most efficiently utilizing limited funds and resources to maximize technological advancement of the Griffin Campus
- Implemented technology changes across the campus and assisted other departments with adapting current business processes.

EXPERIENCE

Program Specialist II **April 2011 – Present**
The University of Georgia, Griffin Campus *Griffin, GA*

- Advised and oversaw the progress of 20+ students in the Consumer Economics degree program
- Recruited new students; Served as the representative for the department in public settings
- Purchased and maintained software licenses for faculty and staff in department; Served in IT support role for department
- Debugged interface and compiled technical manuals for in-house student records database
- Scheduled courses, reserved classrooms, and acquired textbooks and teaching materials for faculty

Utility Worker **April 2008 – April 2010**
The University of Georgia, Griffin Campus *Griffin, GA*

- Maintained and oversaw the maintenance and operation of research equipment
- Installed, tested, and performed upgrades to equipment related to research purposes
- Prepared facilities and projects for presentations and demonstrations
- Oversaw and Assisted an individual taking part in a summer internship including overseeing environmental research and the compiling of related reports

House Manager **October 2009 – May 2010**
The Phi Delta Theta Fraternity *Athens, GA*

- Managed the maintenance and upkeep of the fraternity house for the chapter
- Worked with individuals in the community and organization to improve the image of the house; Oversaw renovations in public areas of the building and coordinated with outside contractors
- Chapter Executive Board; Made key decisions involving the direction and growth of the chapter
- Helped organize and run philanthropic events; Participated in both the acquiring and setup of equipment related to such events along with the staffing and scheduling of said equipment

COMMUNITY INVOLVEMENT

National Association of Homebuilders (NAHB) **September 2009 – May 2010**
Athens, GA

- Gained Knowledge and interacted with members of the Housing industry
- Worked with fellow members in volunteer projects around the Athens Metro-Area

PROFESSIONAL SKILLS

- Microsoft Office (Word, Excel, PowerPoint, and Access)
- Python
- ArcGIS
- SQL
- Java
- Adobe Suite

Justin Patel

3908 Kingsley Park Lane, Duluth, GA 30096
Home: 770-263-1051 - Cell: 770-367-6582 : Justinptl@gmail.com

Profile

Experience as a Customer Service Specialist, Cashier, Re-Stocker, Copy and Print Business Specialist, and Maintenance. Have hands on skill with Technology and Retail. I am a Student in a Bachelors Degree Program in Computer Science.

Core Qualifications

- Customer Satisfaction Experience
- Experience in Java and HTML coding
- Strong Analytic skills
- GUI and tools
- Document management
- Excellent problem solving skills

Technical Skills

Skills	Experience	Total Years	Last Used
Computer and Problem Solving Skills	Troubleshooting, Problem Solving, and Programming	3	December 2013

Professional Experience

Customer Service Specialist
February 2012 to April 2012
Office Depot - Johns Creek, GA
I directly dealt with assisting and interacting with customers. I held the role of cashier, handled money, handled copy and print services for businesses, restocked the store supplies, and experience in maintenance.

Education

High School Diploma : **Bachelor of Computer Science**, 2015
Georgia State University - Atlanta, GA, United States
Coursework in the field of data programming, high level math, and physics. Gained various skills through analytical courses that involved problem solving and proofs. Above average skill set in literature and writing.

Payday

Team Rocket

Team Lead: Andrew Rutherford

Suraj Sequeira
Justin Patel
Andrew Rutherford
Sheehan Toufiq
Benjamin Byrd

Team Role	Team Member(s)	Responsibilities
Project Manager	Andrew Rutherford	1.Coordinate Deliverables 2.Set up team meetings 3.Set up presentations
Client	Sheehan Toufiq Andrew Rutherford Ben Byrd Suraj Sequeira Justin Patel	1.Make Requirements
UI Developer	Ben Byrd	1. Design the user interface
	Sheehan Toufiq	
Database Developer	Sheehan Toufiq Andrew Rutherford Ben Byrd Suraj Sequeira Justin Patel	1. Design the database 2.Implement the abstract operations

Team Role	Team Member(s)	Responsibilities
Project Manager	Andrew Rutherford	1.Coordinate Deliverables 2.Set up team meetings 3.Set up presentations
Client	Sheehan Toufiq Andrew Rutherford Ben Byrd Suraj Sequeira Justin Patel	1.Make Requirements
UI Developer	Ben Byrd	1. Design the user interface
	Sheehan Toufiq	
Database Developer	Sheehan Toufiq Andrew Rutherford Ben Byrd Suraj Sequeira Justin Patel	1. Design the database 2.Implement the abstract operations
Technical Writer	Sheehan Toufiq	1. Write the various documents needed for the project
	Andrew Rutherford	
	Ben Byrd	
	Suraj Sequeira	
	Justin Patel	
Tester	**Sheehan Toufiq**	**1. Responsible for designing the test plan**
		2. Responsible for testing the functions of the prog

Requirement Rationale

Numerous businesses exist in the United States. According to statistics provided by the U.S. Census Bureau and cited by the Small Business & Entrepreneurship Council, 89.8% of businesses in the US have 20 or fewer employees. After a period of research into comparable products, we found a limited number of products focused toward the small business owner. Seeing a need in the market, we, Team Rocket, have decided to develop a simple-to-use, easy-to-access, small business accounting system.

To satisfy the needs of the modern small business owner, we consulted a potential client, Mr. Sheehan Toufiq, for a general assessment on the needs and requirements for a small business accounting system. Through a month long discussion and analysis, we determined the major requirements for a small business accounting system. These boiled down to five major categories: Transaction recording, expense reporting, revenue reporting, employee tracking, and payroll.

According to our client, one of the major difficulties in running his small business is maintaining an accurate reading of the cash flow of the business. Currently, our client tracks cash flow with an Excel workbook.. While this does allow for some basic management of cash flow, overall the

experience is cumbersome and limiting. To ease this burden for both Mr. Toufiq, as well as the rests of the small business world, we proposed that we design Payday to feature an easy-to-use, cash flow system. Using a simple interface with minimal interactions, Payday can keep track of the cash flow for the entire business.

Another difficulty encountered by our client was in the area of report generation. Mr. Toufiq found Excel's reporting functionality to be both complicated and limited. We sought to remedy that by implementing two major reporting functions in Payday: expense reporting and revenue reporting. Upon initial configuration, Payday is configured to produce reports specific to expenses and revenue with the minimal amount of inputs and interaction necessary. The reporting feature simplifies the process of report generation for the small business owner, and gives them a easy way to check their company's performance. Furthermore, web-based design allows the owner to access these reports from virtually anywhere with internet access.

Another area common between Mr. Toufiq's and small-business owner's needs is in the field of employee management. Mr. Toufiq, in his quest to expand his business, was looking into adding some employees to help distribute the workload. After some diligent research, he told us he

found the idea to be far less promising than what he initially expected. For starters, he would have keep track of hours, taxes, Medicare, unemployment insurance, and much more. We surveyed a couple other local business owners and the consensus was pretty similar: employee management can be a pretty big hassle for the small business owner. In Mr. Toufiq's case, the Excel spreadsheet he's been using for about a year was starting to sound a little primitive.

For us, we see an opportunity. By integrating and automating many of the small business management requirements into accounting software, we can reduce the workload on our clients and allow them to get back to what really matters: business. Providing systems for payment, personnel data, and more as well as integrating with transaction recording and report generation allows for a streamlined business process for payments that covers over 89.8% of the business world. Web-based design simplifies platform constraints and opens us up to the majority of all computing platforms, and allows us to reach an ever growing audience of web-connected individuals all over the US. By making the software easy to use, we provide people who may have never thought about accounting for their business on the computer a no-risk, hassle-free means by which they can simplify the requirements for operating a business.

Out of Scope Rationale

At The beginning of the project our group thought of different functions for our accounting system. We wanted our application to be as useful and innovative as possible for a small business. But as we progressed with the project we realized some of the functionalities were not possible because of time and schedule constraints. We decided to take out a few functionalities that were not seen beneficial to our application as other functions were to the accounting system. We brainstormed and decided the functions that should be removed were clock in and out for employees, filing and calculating taxes, and allowing our PDF generation to print.

We felt these functions were a great addition for the accounting system but were not necessary for the accounting system for a small business. These functionalities gave additional help to user but if not added it would not take away value from the application.

Out of Scope Rationale

Employee Clocking in and out Functionality

This functionality gave the employee the ability to record the time he starts work and the time he finishes. This functionality gave our accounting

system a payroll side for the employee. It would have a clock in button that creates a timestamp. It also has a clock out button, which also creates a timestamp. Payday would then calculate the amount of hours you worked that day and store information for viewing to the employee and manager. Payday would also take the time stamps from each day the employee has clocked in and out and calculate the number of hours for each week, month, or year. Payday would ask the employee or manager which one he wanted to view and once the desired option is selected the data would appear according to the information provided by the user.

We felt this functionality was not needed for an accounting system for a small business. We felt this system should give primary use to the manager or the person in charge of the finances for the company that is using payday. Therefore we let the manager have the authority in inputting the employee hours into the Payday application. The application now only will allow the employee to view the number of hours he has worked but they cannot enter in the data into the system. They still have the option to view the number of hours per day, week, month, and years. The employee will now report his hours to the manager, who will enter the hours into payday. Managers will be allowed to edit the hours how he sees fit. This

means if the employee gives the wrong hours, the manager can edit the data he has inputted earlier to make it accurate.

Out of Scope Rationale

Filing and Calculating Taxes Functionality

This functionality would allow the user to enter the data for the company's taxes and Payday would take that information and calculate the the amount that is due for taxes. The manager, therefore, would just simply enter in the correct data in Payday and Payday would take that information and start filing the company's taxes. The application would also take that information in account to calculate the exact amount the manager needed to pay for taxes. We felt this functionality gave an added component to this application in order to make it a more versatile accounting system. This function was thought up to help the user to have a more comfortable experience in regards to finance and accounting. Taxes have always been a sore and tedious task so we want to create a function that would ease the workload from the user and have the application do most of the work.

The main reason for cutting this feature out of the application was mainly because of time. This feature was thought of as an extra feature that would be a like a shiny button on a already new toy. We felt this functionality would benefit the user if added but not harm them if it was not included in the application. The exclusion of this function will just cause the user to continue dealing with the taxes of his company the same way he has been dealing with it in the past. We would like to include this feature but time has not allowed this to be apart of the project as much as we wanted it too.

Out of Scope Rationale

PDF generation Print

This functionality allows the user to print a PDF once it has been generated. The process of this function starts with a PDF generating to allow the user to view the data in a more concrete fashion. It then gives the users the option to print the PDF so the users can have a hard copy of the data the PDF contains. We first thought this function would beneficial for the company and users because it helps give out information in a more

physical way. We then realized this can also allow important information to be floating around outside the application which can be detrimental for the company's business.

The important factor of the threat of having confidential information of our clients getting out, caused us to decide to leave out this function. We still allow the users to view the PDF but it cannot be printed. This allows the users to still get his work done by seeing the information on the PDF, but there is no risk in users letting out the information. This function was thought as being a tool and help for the user to be more productive and efficient in the workplace. This function though after much thought seemed to be more of a potential harm for the user than a benefit.

Object Rationale

When an user comes to our site they will be directed to our login screen. The login screen will be comprised of two text fields, a field where the user can enter his username, and a field where the user can enter in his password. The username

and password the user enters must match a username and password stored in the database. This process is detailed in the first use case. We created several objects in our program to execute this process.

The entity objects for the login process are the userObject and the loginObject. The user object initiates the login sequence by inputting a username and a password. The login object is used by the payday system to authenticate the user. The redirect object takes the user to either their appropriate dashboard, or it will display an error message on the login page. When conceptualizing this process, I found that it would be convenient to make a single object to handle the redirection, rather than implementing redirection in the checkUsername and checkPassword objects. I will expound upon this decision more when I detail the checkUsername and checkPassword objects.

For this process I found that two control objects will be used. The control objects for this use case are the checkUsername and checkPassword objects. These 2 objects compare what the user inputted into each field, with what information is available in the database. These control objects also initialize the redirectObject. When I initially thought out the login process I thought that I could accomplish checking the username and the password with one object, but when it came time to model login process I found it slightly difficult to model a single checking object. When coding I may be able to condense the two objects into one, but during the

analysis phase I found that splitting the objects was easier. I may find that the control objects may become simple boolean functions, and if this is the case I will combine them. When the question of redirection came up I wanted to avoid redundancy, I handled this by making a redirect object, instead of implementing a redirection functionality in each of the control objects.

The boundary object for this process is the loginForm. This provides the user with a form to put their username and password into. The design of this form will be discussed in the layout section of our rationale.

Once a user has been successfully identified as a manager; the user will be presented with several options for the payday system to execute. One such option is the display an employee's history function. This function allows a manager to retrieve an employee's information; such as, their pay history, schedule, and address. In group discussions we found this functionality may not be essential for an accounting system, but when we thought of the real world application of our system, we agreed that the manager of a company should have access to their employee's information.

The entity objects for this use case are the managerObject and the employeeObject. The managerObject initiates the process of viewing an employee's information. The managerObject also designates which employee's information needs to be retrieved. The employeeObject holds the employee's

information to be retrieved. When conceptualizing this process I visualed our system database as several different tables. I then reasoned that the employeeObject would work function as a row of the table, where the row holds the aforementioned information for an employee of the company. The managerObject will be used throughout our program, and it will serve several different purposes. The managerObject will function as a database query creator. The manager will specify which employee's information to retrieve, and what specific information the manager wants from the employee object.

The control object for this use case is the retrieveObject. Once the managerObject initiates the process of showing an employee's information the retrieveObject will search the database, and it will return the information from the employeeObject. For clarification, the managerObject is a query creator, the retrieveObject executes the query, and the employeeObject holds the information to be retrieved.

The boundary object for this process is the showEmployeeObject. This is the button the user will see on the manager pane. Once this button is pressed the show employee information process will begin.

A key functionality of any accounting system should be the ability to produce expense reports. Our system will allow managers to keep track of their accrued expenses, and when the system is prompted, it will display a concise

expense report for a given amount of time. The managerObject and the retrieveObject will be reused for this function.

The entity objects for this use case are the managerObject and the expenseObject. I stated previously that the manager object would serve several different purposes, and this is the first example of that. The managerObject initiates the process of receiving the expense report. The manager will specify which expenses he would like to see, and designates a time frame for those expenses. The expenseObject holds all of the expenses accumulated in the payday system. I found that this functionality would function very similarly to the viewing employee information. The managerObject is a query creator, the retrieveObject executes the query, and the expenseObject holds the information to be retrieved.

The boundary object for this use case is the showExpensesObject. This is the button the user will see on the manager pane. Once this button is pressed the generate expense report process will begin.

Alternative Flow Rationale

UC_1:

This is the use case for the login into our accounting system software. The login consists of the entity objects userObject and loginObject. The userObject is necessary in the implementation of our program because it allows the user to enter their personal login credentials that allow access

into the system. Since the software consists of either a Managing User or an Employee type user, the login stands as a alternate flow type use case. Where the login can be either an Employee or Manager. The loginObject is necessary because it is the authorization stage of the login where the user, based upon their enter credentials, is redirected to either the Manager Pane of the Employee Pane. loginObject has an alternative flow where if the user inputs an unrecognized or incorrect credential, he/she shall be presented with an error message, and redirected to the login again. The control objects checkUsername and checkPassword are in charge of the recognition process of the loginObject. These are the main objects that control the path or flow you are directed in. The boundary object loginForm is the front end of this user case where upon the user enters their information. In other words, without any of these objects mentioned, the program will not have the right authoritative functions to manage the type of user, which is very vital in a company software.

UC_13

This use case is for giving the user access or ability to change their personal credentials. Both the Employee and Manager are given this access, but with different constraints, which make this use case vital to the core functioning of our software. The entity objects for the use case are

userManagerModel and userEmployeeModel. The boundary objects that are represented in the front end are editUserManagerBoundary, readUserManagerBoundary, editUserEmployeeBoundary, and readUserEmployeeBoundary. The control objects are updateUserManagerController and updateUserEmployeeController. All of these objects are vital to the core functioning of the software because of risk that may come in company loss of control with credentials. For instance, if a Manager's credentials were to be accidentally leaked, the Manager will be able to edit and change their information in order to prevent risk of company harm. The readUser boundary objects of the use case will be initialized when the user enters into the Edit User Settings view. Dependent upon whether the user is an Employee or Manager, the entity object userManagerModel or userEmployeeModel will be retrieved. This alternative flow of the use case to lead the type of user to the correct Pane in order to avoid unpermitted access to an Employee or Manager into each others information. Once the Employee is done editing the information, he/she will hit the save button, which will initialize the editUser boundary objects that will update the information in the database record. Each step and object is vital to ensure that the Employee stays on their proper Pane and the Manager stays in their proper Pane.

Single Flow Object Rationale

UC_5 Manager Pane

This use case is provides the Manager user to have a dashboard Pane that allows editing of Employee, where the Employee is listed by name. This is a constraint rather than an actual process function. This use case is non-functional, but was included within our software because it adds a simplistic method for the Manager to carry out editing the Employee's information. The Employee can also be listed with their SSN, or anything else, but to keep things simple, we decided to use their names and a listing.

UC_6 New Employee Information

This use case deals with the core function of adding a new employee. It is a non-functional use case that acts as more of a constraint, rather than a required function to process data. The constraints deal with the type of information that is requested and entered in by the user. The only object involved in this use case is the insufficientData Object, which acts as the return error for the constraints specified. The constraints specified deal with whether or not all the data in the fields for Add an Employee are filled out or not. If all the fields are not filled completely, then the insufficientData Object is initialized and the user is presented with an error message. This was not

necessary, but was a great feature we thought should be added in order to notify the user that is adding the Employee of any error in their lack of input.

UC_7 Manage Employee Data

This use case for dealing with giving the Manager the ability to create and Employee Checks/Pay Stubs, Track hours worked by each Employee, Editing Employee History, and Viewing Employee Records. Being able to create a Check and Pay Stub is vital to our software because it allows the Manager to manage the Employee information and create a check all in one software making it convenient. This convenience is what we think make our software stick out above others. Tracking hours is vital to the software because it allows the proper calculation of pay by determining the salary earned on an hourly basis. Without this function, the checks and pay stubs would be unusable. Editing Employee history allows the ability to prevent misinterpretation of information on from a system or user error. If an Employees information in the system is incorrect in comparison to actual events, the Manager is able to alter the system in order to enter correct information. This is absolutely vital in the real-world environment of a company for risk management purposes. The View Employee Records is of

same importance of editing, but just strictly viewing for proper time management purposes.

UC_8

This use case is to calculate the total hours worked by each Employee. The function is very important and vital because it is of utmost importance for company to keep track of the Total Hours that have been worked by each employee so as to get a proper pay stub made and report the hours to any government agency that requires this knowledge. The function is carried out by pulling out the initial clock-in time and final clock out time of each Employee from the database, and getting the difference.

Function Point Analysis

To track our potential development costs, we developed a function point analysis document for the Payday online small business accounting system. To estimate the potential costs, we took into account all the requirements determined necessary by the client and compared them with the object design and system design requirements we developed. All three were used to determine the final estimate cost. First and foremost, we compiled our weighting factor estimate. Broken down into five different measurement parameters, we determined the estimate number of

parameters for each category as well as the difficulty of implementation for each category.

The first measurement parameter, Number of Inputs, was estimated to be 11. We came to this conclusion utilizing the RTM document as a guideline, compiling smaller inputs into “input categories.” Since the different types of input categories vary in number of specific inputs, types of inputs, and size of inputs we determined this input category to require a “complex” weighting factor of 6.

The second measurement parameter, Number of User Outputs, was estimated to be 13. Again, utilizing the RTM document as a guideline, we compiled a list of the major outputs required to satisfy the requirements of the client. Typically, outputs of the Payday system will be some type of accounting request, such as transactions and financial reports. Since each of these various outputs will require a considerable amount of individual data, we assume the “complex” weighting factor of 7 to be appropriate.

The third measurement parameter, Number of User Inquiries, was estimated to be 5. We found in our analysis of the RTM document that the actual number of inquiries by the user was limited. Essentially, Payday will provide four major functions requiring user intervention to operate: Employee reporting, revenue reporting, transaction accounting, and

employee payment. Additionally, basic user interface navigation and functionality will be required as well. Since the system is designed for the small business owner, we determined that the system will be designed around average weighting factor estimate of 4.

The fourth measurement parameter, Number of Internal Files, was estimated to be 3. For the Payday system to meet the requirements provided by the customer, we determined the system to require three separate database tables: One for user information/tracking, one for transaction data, and one for revenue reporting. Since there is potential for a significant amount of data recorded and the individual data will comprise of many differerent components, we estimated the weighting factor to be the most complex at a value of 15.

The last measurement parameter, Number of External Interfaces of Files, was estimated to be 2. Since Payday will be using industry standard encoding of data for the web page generation and using standard database calls, we determined that we would at most have to develop 2 original external facing interfaces. The complexity should be low since we will implement standard interface designs, thus we utilized a simple weighting factor of 5.

The second major component of the Function Point Analysis was the Rating Estimate of Categories. Utilizing the requirement document, the RTM, the system design and object design we gave an estimate difficulty/importance rating for each category.

Since Payday is an accounting software platform, we will need to have both accurate and reliable record keeping. We rate this as a 5 in number of importance. Payday will be a web based accounting system, thus will require communications with outside systems. Category two was awarded an importance rating of 4. On the backend, Payday will perform simple calculations of financial transactions. Since these transactions will not require heavy processing and complex computation, we award category 3 an importance rating of 2. With that being said, performance is a must for Payday. The system will need to output data in a very efficient manner. Thus, we award category 4 a rating of 4. We don't expect the system to be used in an existing environment as it is designed for small businesses. Thus, we award category 5 a rating of 3. Payday will be online for all functions, thus we will award category 6 a rating of 4 and a rating of 4 to category 8. Generally, most functionality will require inputs on individual screens for processing. Thus, we award a rating of 2 to category 7. All processing for Payday should not be complex, thus we award a 3 rating to

both categories 9 and 10. Payday will utilize an object oriented design and thus will feature reusable code. We award category 11 a rating of 4. Installation will not be included in the design, thus category 12 is awarded a rating of 2. Since Payday is designed to be a simple small business accounting system, it will be available to many different organizations. Ease of use is a priority for a varied user base, thus we award a 4 rating to category 13 and a 5 rating to category 14.

For the final state of the Function Point Analysis, we used a cost factor of $1000 for each Function Point based on our previous experiences and current project loads. Given the total FPC estimate of 264.48, we determined the estimate cost for developing the Payday system to be $264,480.

Rationale_Server_Technologies_Used

The following include rationales for the technologies used for the server development for Payday's client server architecture.

Amazon EC2 Cloud – Virtual Host

- Amazon EC2 Cloud is a virtual cloud host designed to provide computing power based on demand. The rationale for using Amazon EC2 Cloud as a virtual host for Payday is that there is familiarity with the technology by our developers, it has an easy to use console, it is highly

secure, the support and community is unparalleled in the industry, there is a highly active community, and it is free for one year with new accounts.

Ubuntu Server 12.04 LTS Linux – Operating System

- Ubuntu Server 12.04 LTS is a Linux distribution for a server environment. The rationale for using Ubuntu Server is that it is highly portable supports UNIX commands, there is familiarity with the technology by our developers, it is easy to set up, the documentation is unparalleled in the industry, there is a highly active community, and it is free.

Apache HTTP Web Server – HTTP File Server

- Apache HTTP Web Server is a file server designed to serve files from a directory tree structure. The rationale to use Apache HTTP File Server is that it works well out of the box with Ubuntu Server, it has a simple set up, there is familiarity with the technology by our developers, it is a mature technology, and it is free.

MySQL – Database

- MySQL is an open source relational database that is supported by Oracle. The rationale to use MySQL for Payday is that is a relational database, which handles account transactions well which is a requirement for Payday. Also it supports SQL commands, there is a familiarity with the technology by our developers, it is open source, it supported by Oracle, the

documentation is unparalleled in the industry, there is a highly active community, it is simple to set up and works well out of the box with Apache HTTP Web Server, the technology is highly mature, and it is free.

Gradle – Build Automation

- Gradle is a build automation tool for testing, packaging, and publishing Java and JVM for the web. The rationale for using Gradle for Payday is that it is supported by Spring.io, it is simple to use for publishing Java for the web, it is an industry standard for java testing automation, it has a simple set up, it is open source, the documentation is unparalleled in the space, and it is free.

Spring.io – Backend Logic

- Spring.io is an enterprise level java MVC web framework. The rationale for using Spring.io for Payday is that it is a modern Java MVC framework, it has the best and easiest to follow documentation out of all the enterprise level Java frameworks, it has great support, simple set up, and a highly active community. Spring.io is also open source and completely free to use.

Rationale_Design_Patterns_Used

The following include rationales for the design patterns used by Payday.

Server Side Model View Controller (MVC)

• A model view controller design pattern is a system that allows controller layers to serve and manipulate data from model layers, then serve the data and take inputs from view layers. Payday's server side MVC design includes models that contain data objects that are built from the database and controllers serving data to views that are Restful JSON endpoints. The rationale for using a model view controller design pattern within the server side logic for Payday is that it allows for modularization of code, it promotes better maintenance practices, minimizes spaghetti code and promotes DRY (Don't Repeat Yourself) principles.

Client Side Model View Controller (MVC)

• Payday's client side MVC design includes models that contain data objects that are built from GET, PUT, POST, and UPDATE HTTP requests from the server side logic that serves the Restful JSON endpoints as views. Controllers then serve data to views that are contained in the presentation client layer of the application. The rationale for using a model view controller design pattern within the client side logic for Payday is that it is a system that allows controller layers to serve and manipulate data from model layers, then serve the data and take inputs from view layers. It also allows for modularization of code, it promotes better maintenance practices, minimizes spaghetti code and promotes DRY (Don't Repeat Yourself) principles.

Singleton

• A singleton is a design pattern that allows a master object control logic that is required throughout the entire use of an application. The rationale for Payday to use a singleton design pattern is that it is the best implementation for application wide login sessions, interactions, and manipulation of data by a current logged in user object.

Rationale_Server_Technologies_Used

The following include rationales for the technologies used for the server development for Payday's client server architecture.

Amazon EC2 Cloud – Virtual Host

• Amazon EC2 Cloud is a virtual cloud host designed to provide computing power based on demand. The rationale for using Amazon EC2 Cloud as a virtual host for Payday is that there is familiarity with the technology by our developers, it has an easy to use console, it is highly secure, the support and community is unparalleled in the industry, there is a highly active community, and it is free for one year with new accounts.

Ubuntu Server 12.04 LTS Linux – Operating System

• Ubuntu Server 12.04 LTS is a Linux distribution for a server environment. The

rationale for using Ubuntu Server is that it is highly portable supports UNIX commands, there is familiarity with the technology by our developers, it is easy to set up, the documentation is unparalleled in the industry, there is a highly active community, and it is free.

Apache HTTP Web Server – HTTP File Server

• Apache HTTP Web Server is a file server designed to serve files from a directory tree structure. The rationale to use Apache HTTP File Server is that it works well out of the box with Ubuntu Server, it has a simple set up, there is familiarity with the technology by our developers, it is a mature technology, and it is free.

MySQL – Database

• MySQL is an open source relational database that is supported by Oracle. The rationale to use MySQL for Payday is that is a relational database, which handles account transactions well which is a requirement for Payday. Also it supports SQL commands, there is a familiarity with the technology by our developers, it is open source, it supported by Oracle, the documentation is unparalleled in the industry, there is a highly active community, it is simple to set up and works well out of the box with Apache HTTP Web Server, the technology is highly mature, and it is free.

Gradle – Build Automation

• Gradle is a build automation tool for testing, packaging, and publishing Java and JVM for the web. The rationale for using Gradle for Payday is that it is supported by Spring.io, it is simple to use for publishing Java for the web, it is an industry standard for java testing automation, it has a simple set up, it is open source, the documentation is unparalleled in the space, and it is free.

Spring.io – Backend Logic

• Spring.io is an enterprise level java MVC web framework. The rationale for using Spring.io for Payday is that it is a modern Java MVC framework, it has the best and easiest to follow documentation out of all the enterprise level Java frameworks, it has great support, simple set up, and a highly active community. Spring.io is also open source and completely free to use.

Rationale_System_Architectures_Used

The following include rationales for the system architectures used for Payday.

Client Server Architecture

 Payday's client server architecture includes a client side and a server

side where the client makes requests to the server, which carries out logic to manipulate the database. The rationale for using a client server architecture for Payday is that it allows multiple clients to be active across different sessions, it allows data to be stored within a centralized database location on the server, it allows the same data to be accessed by multiple clients, it allows support for multiple platforms, there is familiarity of the architecture by our developers, and it allows flexibility for building future platform agnostic clients.

Four Tier Architecture

 Payday's four tier architecture is a four tiered layered approach that allows abstraction between different system processes. The tiers include storage layer that consists of a database and a file server, an application logic layer that consists of the server side logic, a presentation server layer that consists client side logic, and a presentation client that consists of view templates. The rationale for using a four tier architecture for Payday is that it allows modularization of code, promotes good practices for security and it allows flexibility for building future platform agnostic clients. Also a four tiered layered approach allows abstractions between all system processes and promotes better maintenance.

Restful JSON Endpoints

 Payday's application programming interface includes endpoints serving data from the database as Restful JSON from the server side of the client server architecture. The rationale for using a Restful JSON application programming interface for Payday is that is allows simple parsing by frontend JavaScript, it is extremely fast, it has an object oriented data structure, and there is familiarity of the architecture by our developers.

Rationale_User_Interface_Decisions

The following includes rationales for each major user interface (UI) decision in Payday.

Having the menu navigation on the left side

 The rationale for Payday to have the application menu navigation on the left side on the client views are that it allows for a much friendlier overall user experience, and it is a modern approach to user interface design.

Each user has a has a unique dashboard

 The rationale for Payday to have the application serve a unique dashboard for each user is because it gives an overview of data that is

needed by the user. The manager requires sales and expenses data, while an employee requires work history and paystub data.

Dropdown for categories on manager transactions

 The rationale for Payday to have a dropdown for categories when creating a new transaction on the manager transactions view is that it will be a friendlier user experience to track transactions by categories. Also categories will be able to be created by the managing user then the data of existing categories will be read by the dropdown menu.

Dropdown for type of transaction (Debit vs. Credit)

 The rationale for Payday to have a dropdown for the type of transaction is that the system reads the input through a createTransaction() method that has an if/else statement to determine if the user input of a new transaction type is a "Debit" or "Credit". The method checks against the string "Debit", so any other value will credit the manager's account, or return an error if the logic is built out in the future.

Payday 1.0 - Introduction

Payday is a web-based accounting and employee management system developed for small businesses.

Payday 2.0 - Web-based accounting and employee management system

Payday shall perform common accounting functions and keep track of employees.

Payday 3.0 - Login

Payday shall provide a login page for users. If the user inputs correct login information, they shall be directed to the appropriate user dashboard. If users input incorrect login information, they shall be presented with an “incorrect login information” notification.

Payday 3.1 - Postlogin

Payday shall provide two categories of users. Payday will provide an employee user category and an employer user category. Depending on the user’s type, upon a successful login they shall be redirected to their designated dashboard.

Payday 4.0 - Expense functions

Payday shall query the database to verify that the login information for a manager is correct. If the login information does not match the stored data for a manager, the application shall query the database to verify that the login information for an employee is correct. If the login information does not match the stored data for an employee the application outputs an error message.

Payday 4.1 - Expense report functions

Upon user request Payday shall generate expense reports based upon transactions stored in the database

Payday 5.0 - Employee functions

Payday shall provide functions for maintaining employee records. Payday shall provide functions for adding new employees, viewing/editing employee information, and deleting employee records. Each employee shall contain appropriate identifying information. Payday will provide the managing user an employee management dashboard pane. Payday shall list employees by name.

Payday 5.1 - Adding New Employees

Payday shall provide the managing user a function to add new employees. Payday shall provide a dashboard that will allow the manager to submit required employee information. Payday shall require at minimum the following employee information: Name, Address, Date of Birth, Social Security Number, Current Employment Status, and Job Title. If the managing user fails to input any of this information, Payday shall prevent creation of the employee record and provide user with an “insufficient information” notification. Upon successful creation of employee, Payday shall notify managing user of successful creation, record date/time of employee creation, and list the employee under the main employee management dashboard pane.

Payday 5.2 - Viewing/Editing Employees

Payday shall provide the managing user functions for viewing and editing employee records. Payday shall provide functions for viewing employee information submitted upon job creation. Payday shall provide a function allowing the editing of all fields listed under the employees personal record. Payday shall allow the managing user a function to save the edited record in the form of a Yes or No question. Upon request to save, Payday will present the managing user a confirmation. Upon confirmation, the record changes will be saved. Otherwise, upon dismissal the changes shall be discarded.

Payday 5.3 - Deletion of Employees

Payday shall provide the managing user a function for deleting an employee. Payday shall provide a function on the employee dashboard for selecting an employee for deletion. Payday shall confirm with the managing user deletion via a Yes or No question. Upon confirmation of record deletion, Payday shall remove all records of the deleted employee. Upon dismissal, Payday shall do nothing.

Payday 6.0 - Employment Net Pay

Payday shall provide the managing user functions for managing employee payment. Payday shall provide a dashboard pane for performing employment pay functions. Payday shall provide functions for creating employee checks/pay stubs, tracking employee hours, editing employee history, and employee record viewing.

Payday 6.1 - Employment Checks/Pay Stubs

Payday shall provide the managing user functions for creation and viewing of employee checks and pay stubs. Payday shall provide the managing user functions to view employment pay for the requested pay period. Payday shall provide the managing user functions to generate a check and attached paystub. Payday shall list pay periods as defined by the managing user.

Payday 6.2 Clock in, Clock out for Employees(F)

Payday shall provide the process of gathering the total amount of hours each employee has worked in a shift by getting the difference between their clock in and clock out time. ***(Suggested: Expected vs. Arrived clock in/out time?)

Payday 6.3 Managers can edit employee history(NF)

Payday shall allow managers to be given access to alter/edit employee previous clock in and clock out times, number of hours worked by each employee, and quarterly employee pay stubs. ***(Suggested: Expected vs. Arrived clock in/out time? Customer/Client Assistance log(Which employee served which customer)?).

Payday 10.0 - Employee History

Payday shall allow employees and managers to view a breakdown of each employees hourly work and payment history. Managers shall have the ability to change the time of an employee's work history. The work history shall be able to be viewed as a past week, past 2 weeks, past month, or entire work history format.

Payday 11.0 - Revenue

Payday shall allow managers to view daily revenues. Payday shall also calculate and allow the manager to view monthly and yearly revenues. If the manager inputs daily revenues, Payday will take the data and output the yearly and monthly revenue. Payday shall also take the data and allow the manager to view the information in a chart or graph format.

Payday 12.0 - Editing Personal Settings

Payday shall allow the manager and employees to change and edit their credentials. If an employee or manager want to change their password, user ID, or image, Payday will allow the user to alter the information he wants to change. Payday shall first make certain user changing the information previously inputted is the correct user.

Payday 13.0 - View Tax Breakdown

Payday shall allow managing users to view a breakdown of year-to-date taxes to be paid from past transactions, including expenses and revenue. Payday shall generate the net profit from the Profit & Loss statement of the business and use specific parameters including if the business is a sole proprietorship, partnership, s-corporation, c-corporation, or limited liability corporation, and what state the business is located to accurately calculate the federal and state taxes.

Payday 14.0 - User Requirements

Payday shall require the use of the Google Chrome internet web browser Greater than Version 37. Payday will use standard internet user authentication protocols.

ID #	Paragraph	Requirement Traceability Matrix	Type	Functional vs Nonfunctional	Use Case Name
1	3	Payday shall provide a login page for users, which shall query the database to check for correct login information	SW	Functional	UC_1_Login
2	2	Payday shall keep track of employee information	SW	Functional	UC_2_Employee_Information
3	4.1	Upon user request Payday shall generate expense reports based upon transactions stored in the database	SW	Functional	UC_3_Expense_Reports
4	5	Payday shall provide functions for adding and deleting employee records. Payday will also provide functions to delete all records of a former employee. Payday will also notify the managing user of every creation, record date/time of employee creation, and list the employee under the main employee management dashboard pane.	SW	Functional	UC_4_Edit_Employee_Functions
5	5	Payday shall provide the managing user an employee management dashboard pane with Employee listed by name	SW,SWC,DR	Non-Functional	UC_5_Manager_Pane
6	5.1	Payday shall require at minimum the following employee information: Name, Address, Date of Birth, Social Seurity Number, Current Employment Status, and Job Title. Payday will prevent the creation of the employee record if the information asked has not been filled and provide the user with an "insufficient information" notification.	DWC,SW,DR	Non-Functional	UC_6_New_Employee_Information
7	6	Payday shall provide functions for creating employee checks/pay stubs, tracking employee hours, editing employee history, and employee record viewing.	SW	Functional	UC_7_Manage_Employee_Data
8	6.2	Payday shall provide the process of gathering the total amount of hours each employee has worked in a shift by getting the difference between their clock in and clock out time.	SW	Functional	UC_8_Employee_Hours
9	6.3	Payday shall allow managers to be given access to alter/edit employee previous clock in and clock out times, number of hours worked by each employee, and quartly employee pay stubs. Payday will also allow employees and managers to view a breakdown of each employees hourly work and payment history.	SW	Functional	UC_9_Manager_Pay_Control
10	10	The work history shall be able to be viewed as a past week, past 2 weeks, past month, or entire work history format	SWC	Functional	UC_10_Pay_History
11	11	Payday shall also calculate and allow managers to add revenue transaction and view daily,weekly, monthly, and yearly revenues	SW	Functional	UC_11_Revenue
12	11	Payday shall also take the revenue and expenses and allow the manager to view the information in a chart or graph format	SW,DR,NTW	Functional	UC_12_RetrieveGraphData
13	12	Payday shall allow the Managers and employees to change edit their personal user settings, including password, user ID, or profile image	SW	Functional	UC_13_EditUserSettings
14	13	Payday shall generate the net profit from the Profit & Loss statement of the business and use the specific parameters including if the business is a sole proprietorship, partnership, s-corporation, c-corporation, or limited liability corporation, and where the business is located to accurately calculate the federal and state taxes.	SW,SWC	Functional	UC_14_ProfitLossStatement
15	14	Payday shall require the use of the Google Chrome internet web browser Greater than Version 37	SWC	Non-Functional	
16	14	Payday will use standard internet user authentication protocols. (OAuth)	SWC	Non-Functional	

ID #	Paragraph	Requirement Traceability Matrix	Type
1	2	Payday shall perform common accounting functions.	SW
2	2	Payday shall keep track of employee information	SW
3	3	Payday shall provide a login page for users.	SW
4	3.1	If the user inputs correct login information, they shall be directed to the appropriate user dashboard	SW
5	3.1	If users input incorrect login information, they shall be presented with an “incorrect login information” notification.	SW
6	4	Payday shall query the database to verify that the login information for a manager is correct.	SW
7	4	for a manager, the application shall query the database	SW,DR
8	4	for an employee the application outputs an error	SW,DR
9	4.1	reports based upon transactions stored in the database	SW
10	5	records	SW
11	5	employees, viewing/editing employee information, and	SW
12	5	Each employee shall contain appropriate identifying	SWC
13	5	management dashboard pane.	SW
14	5	Payday shall list employees by name.	SWC
15	5.1	add new employees.	SW,DR
16	5.1	Payday shall provide a dashboard that will allow the manager	SW,DR
17	5.1	information: Name, Address, Date of Birth, Social	DWC
18	5.1	Payday shall prevent creation of the employee record	SW,DR
19	5.1	notify managing user of successful creation, record	SW,DR
20	5.2	viewing and editing employee records.	SW
21	5.2	information submitted upon job creation.	SW
22	5.2	fields listed under the employees personal record.	SW
23	5.2	the edited record in the form of a Yes or No question.	SW,DR
24	5.2	user a confirmation.	SW, DR
25	5.2	Upon confirmation, the record changes will be saved	SWC, DR
26	5.2	Upon dismissal the changes shall be discarded.	SWC, DR
27	5.3	deleting an employee.	SW
28	5.3	dashboard for selecting an employee for deletion.	SW
29	5.3	a Yes or No question.	SW,DR
30	5.3	remove all records of the deleted employee.	SWC,DR
31	5.3	Upon dismissal, Payday shall do nothing.	SWC,DR
32	6	managing employee payment.	SW
33	6	employment pay functions.	SW
34	6	checks/pay stubs, tracking employee hours, editing	SW
35	6.1	creation and viewing of employee checks and pay stubs.	SW,DR
36	6.1	view employment pay for the requested pay period.	SW,DR
37	6.1	generate a check and attached paystub.	NTW
38	6.1	user.	SWC,DR
39	6.2	amount of hours each employee has worked in a shift by	SW
40	6.3	alter/edit employee previous clock in and clock out	SW

51	10	breakdown of each employees hourly work and payment	SW
52	10	employee’s work history.	SW,DR
53	10	week, past 2 weeks, past month, or entire work history	SWC
54	11	Payday shall allow managers to view daily revenues.	SW
55	11	view monthly and yearly revenues.	SW
56	11	the data and output the yearly and monthly revenue.	SW,DR
57	11	to view the information in a chart or graph format.	NTW
58	12	change and edit their credentials	SW
59	12	password, user ID, or image, Payday will allow the user	SW,DR
60	12	information previously inputted is the correct user.	SW,DR
61	13	of year-to-date taxes to be paid from past transactions,	SW
62	13	Loss statement of the business and use specific	SW,SWC
63	14	internet web browser Greater than Version 37.	SWC
64	14	protocols.	SWC
65			
66			

Functional vs Nonfunctional

Functional

Functional

Functional

Functional

Functional

Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional

Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Functional
Non-Functional
Non-Functional

Requirement Rationale

Numerous businesses exist in the United States. According to statistics provided by the U.S. Census Bureau and cited by the Small Business & Entrepreneurship Council, 89.8% of businesses in the US have 20 or fewer employees. After a period of research into comparable products, we found a limited number of products focused toward the small business owner. Seeing a need in the market, we, Team Rocket, have decided to develop a simple-to-use, easy-to-access, small business accounting system.

To satisfy the needs of the modern small business owner, we consulted a potential client, Mr, Sheehan Toufiq, for a general assessment on the needs and requiements for a small business accounting system. Through a month long discussion and analysis, we determined the major requirements for a small business accounting system. These boiled down to five major categories: Transaction recording, expense reporting, revenue reporting, employee tracking, and payroll.

According to our client, one of the major difficulties in running his small business is maintaining an accurate reading of the cash flow of the business. Currently, our client tracks cash flow with an Excel workbook.. While this does allow for some basic management of cash flow, overall the experience is cumbersome and limiting. To ease this burden for both Mr. Toufiq, as well as the rests of the small business world, we proposed that we design Payday to feature an easy-to-use, cash flow system. Using a simple interface with minimal interactions, Payday can keep track of the cash flow for the entire business.

Another difficulty encountered by our client was in the area of report generation. Mr. Toufiq found Excel's reporting functionality to be both complicated and limited. We sought to remedy that by implementing two major reporting functions in Payday: expense reporting and revenue reporting. Upon initial configuration, Payday is configured to produce reports specific to expenses and revenue with the minimal amount of inputs and interaction necessary. The

reporting feature simplifies the process of report generation for the small business owner, and gives them a easy way to check their company's performance. Furthermore, web-based design allows the owner to access these reports from virtually anywhere with internet access.

Another area common between Mr. Toufiq's and small-business owner's needs is in the field of employee management. Mr. Toufiq, in his quest to expand his business, was looking into adding some employees to help distribute the workload. After some diligent research, he told us he found the idea to be far less promising than what he initially expected. For starters, he would have keep track of hours, taxes, Medicare, unemployment insurance, and much more. We surveyed a couple other local business owners and the consensus was pretty similar: employee management can be a pretty big hassle for the small business owner. In Mr. Toufiq's case, the Excel spreadsheet he's been using for about a year was starting to sound a little primitive.

For us, we see an opportunity. By integrating and automating many of the small business management requirements into accounting software, we can reduce the workload on our clients and allow them to get back to what really matters: business. Providing systems for payment, personnel data, and more as well as integrating with transaction recording and report generation allows for a streamlined business process for payments that covers over 89.8% of the business world. Web-based design simplifies platform constraints and opens us up to the majority of all computing platforms, and allows us to reach an ever growing audience of web-connected individuals all over the US. By making the software easy to use, we provide people who may have never thought about accounting for their business on the computer a no-risk, hassle-free means by which they can simplify the requirements for operating a business.

Out of Scope Rationale

At The beginning of the project our group thought of different functions for our accounting system. We wanted our application to be as useful and innovative as possible for a small business. But as we progressed with the project we realized some of the functionalities were not possible because of time and schedule constraints. We decided to take out a few functionalities that were not seen beneficial to our application as other functions were to the accounting system. We brainstormed and decided the functions that should be removed were clock in and out for employees, filing and calculating taxes, and allowing our PDF generation to print.

We felt these functions were a great addition for the accounting system but were not necessary for the accounting system for a small business. These functionalities gave additional help to user but if not added it would not take away value from the application.

Out of Scope Rationale

Employee Clocking in and out Functionality

This functionality gave the employee the ability to record the time he starts work and the time he finishes. This functionality gave our accounting system a payroll side for the employee. It would have a clock in button that creates a timestamp. It also has a clock out button, which also creates a timestamp. Payday would then calculate the amount of hours you worked that day and store information for viewing to the employee and manager. Payday would also take the time stamps from each day the employee has clocked in and out and calculate the number of hours for each week, month, or year. Payday would ask the employee or manager which one he wanted to view and once the desired option is selected the data would appear according to the information provided by the user.

We felt this functionality was not needed for an accounting system for a small business. We felt this system should give primary use to the manager or the person in charge of the finances for the company that is using payday. Therefore we let the manager have the authority in inputting the employee hours into the Payday application. The application now only will allow the employee to view the number of hours he has worked but they cannot enter in the data into the system. They still have the option to view the number of hours per day, week, month, and years. The employee will now report his hours to the manager, who will enter the hours into payday. Managers will be allowed to edit the hours how he sees fit. This means if the employee gives the wrong hours, the manager can edit the data he has inputted earlier to make it accurate.

Out of Scope Rationale

Filing and Calculating Taxes Functionality

This functionality would allow the user to enter the data for the company's taxes and Payday would take that information and calculate the the amount that is due for taxes. The manager, therefore, would just simply enter in the correct data in Payday and Payday would take that information and start filing the company's taxes. The application would also take that information in account to calculate the exact amount the manager needed to pay for taxes. We felt this functionality gave an added component to this application in order to make it a more versatile accounting system. This function was thought up to help the user to have a more comfortable experience in regards to finance and accounting. Taxes have always been a sore and tedious task so we want to create a function that would ease the workload from the user and have the application do most of the work.

The main reason for cutting this feature out of the application was mainly because of time. This feature was thought of as an extra feature that would be a like a shiny button on a already new toy. We felt this functionality would benefit the user if added but not harm them if it was not included in the application. The exclusion of this function will just cause the user to continue dealing with the taxes of his company the same way he has been dealing with it in the past. We would like to include this feature but time has not allowed this to be apart of the project as much as we wanted it too.

Out of Scope Rationale

PDF generation Print

This functionality allows the user to print a PDF once it has been generated. The process of this function starts with a PDF generating to allow the user to view the data in a more concrete fashion. It then gives the users the option to print the PDF so the users can have a hard copy of the data the PDF contains. We first thought this function would beneficial for the company and users because it helps give out information in a more physical way. We then realized this can also allow important information to be floating around outside the application which can be detrimental for the company's business.

The important factor of the threat of having confidential information of our clients getting out, caused us to decide to leave out this function. We still allow the users to view the PDF but it cannot be printed. This allows the users to still get his work done by seeing the information on the PDF, but there is no risk in users letting out the information. This function was thought as being a tool and help for the user to be more productive and efficient in the workplace. This function though after much thought seemed to be more of a potential harm for the user than a benefit.

Rationale_System_Architectures_Used

The following include rationales for the system architectures used for Payday.

Client Server Architecture

- Payday's client server architecture includes a client side and a server side where the client makes requests to the server, which carries out logic to manipulate the database. The rationale for using a client server architecture for Payday is that it allows multiple clients to be active across different sessions, it allows data to be stored within a centralized database location on the server, it allows the same data to be accessed by multiple clients, it allows support for multiple platforms, there is familiarity of the architecture by our developers, and it allows flexibility for building future platform agnostic clients.

Four Tier Architecture

- Payday's four tier architecture is a four tiered layered approach that allows abstraction between different system processes. The tiers include storage layer that consists of a database and a file server, an application logic layer that consists of the server side logic, a presentation server layer that consists client side logic, and a presentation client that consists of view templates. The rationale for using a four tier architecture for Payday is that it allows modularization of code, promotes good practices for security and it allows flexibility for building future platform agnostic clients. Also a four tiered layered approach allows abstractions between all system processes and promotes better maintenance.

Restful JSON Endpoints

- Payday's application programming interface includes endpoints serving data from the database as Restful JSON from the server side of the client server architecture. The rationale for using a Restful JSON application programming interface for Payday is that is allows simple parsing by frontend JavaScript, it is extremely fast, it has an object oriented data structure, and there is familiarity of the architecture by our developers.

Outline for the following rationales:

- A rationale for each technology used.
- A rationale for each system architecture used.
- A rationale for each design pattern used.
- A rationale for each major UI decision.

1. A rationale for each technology used.
    a. Server:
        i. Virtualhost: Amazon EC2 Cloud
            1. Familiarity with developers
            2. Free for 1 year with new accounts
            3. Easy to use console
            4. Highly secure
            5. Great support
            6. Great community
        ii. Operating System: Linux Ubuntu Server 12.04 LTS
            1. Highly Portable
            2. Supports UNIX commands
            3. Familiarity with developers
            4. Easy setup
            5. Great documentation
            6. Free
        iii. HTTP File Server: Apache HTTP Web Server
            1. Simple setup
            2. Familiarity with developers
            3. Works well with Ubuntu Server
            4. Mature
            5. Free
        iv. Database: MySQL
            1. Relational Database
            2. Supports SQL commands
            3. Familiarity with developers
            4. Open Source
            5. Free
            6. Supported by Oracle
            7. Great community
            8. Simple setup
            9. Mature
        v. Build Automation: Gradle
            1. Simple to use for publishing java for the web
            2. Supported by SpringIO
            3. Great for testing
            4. Great documentation

5. Simple setup
6. Open source
7. Free

vi. Backend Logic: SpringIO
1. Modern java MVC framework
2. Best documentation
3. Great support
4. Simple setup
5. Active community
6. Open source
7. Free

b. Client:
i. Client Logic: AngularJS
1. MVC Architecture for client side frontend
2. Familiarity with developers
3. Support for plenty of third party AngularJS modules
4. Allows real time data binding
5. Extremely fast
6. Allows fluid interactions that mimic native applications
7. Highly flexible
8. Open source
9. Free

ii. Library: jQuery
1. Need this library for Bootstrap
2. Lots of functionality from third party plugins
3. Great documentation
4. Familiarity with developers
5. Simple setup
6. Open source
7. Free

iii. Library: Bootstrap 3.0
1. Expedites frontend design
2. Expedites frontend UI interactions
3. Familiarity with developers
4. Simple setup
5. Great documentation
6. Highly flexible
7. Open source
8. Free

iv. Library: ChartsJS
1. Javascript library to build charts
2. Great documentation

3. Simple setup
4. Open source
5. Free

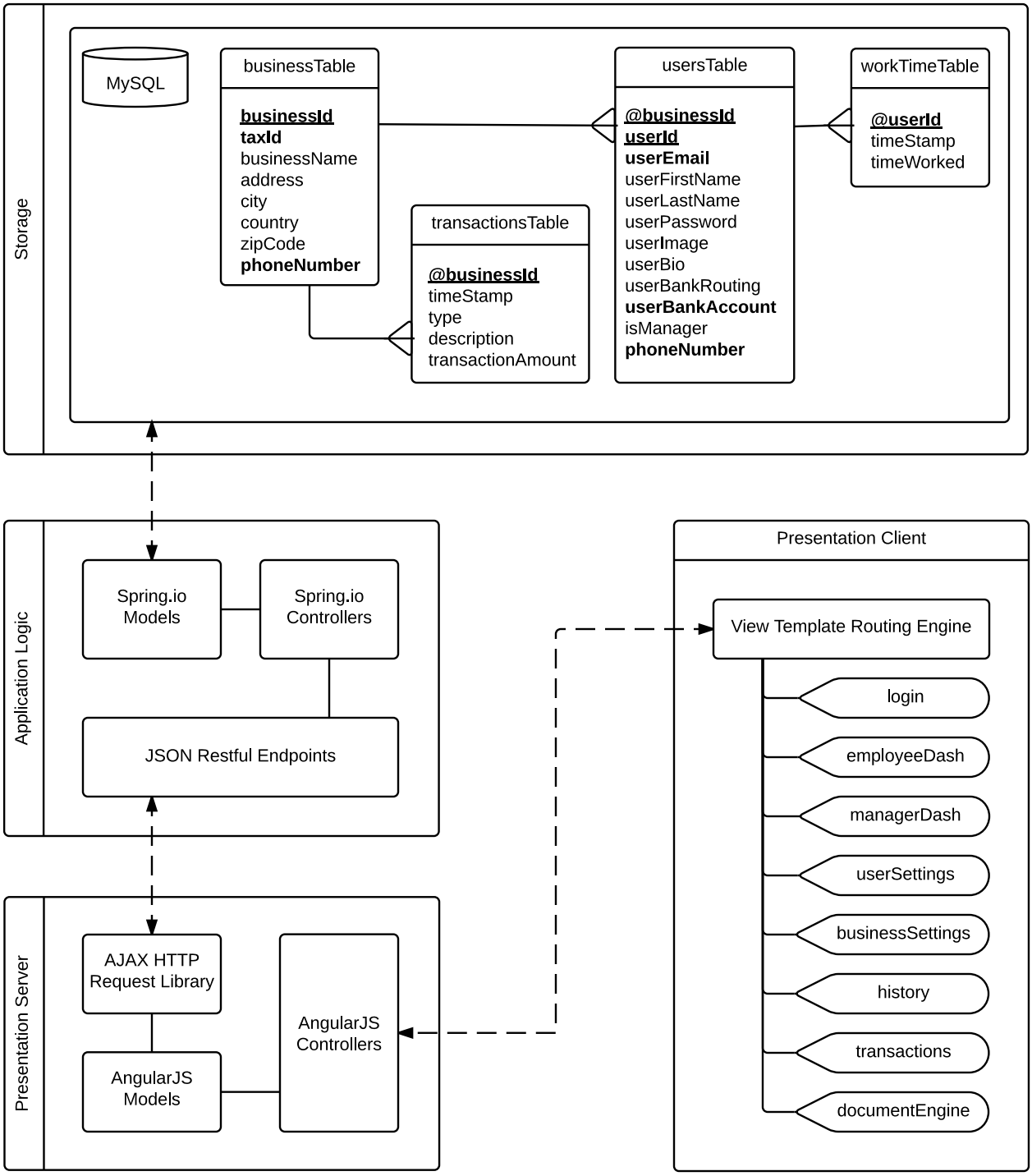
2. A rationale for each system architecture used.
    a. Client Server
        i. Allows multiple clients to be active across different sessions
        ii. Allows data to be stored within a centralized database
        iii. Allows the same data to be accessed by multiple clients
        iv. Allows support for multiple platforms
        v. Allows flexibility for building future platform agnostic clients
    b. Four Tier
        i. Allows modularization of code
        ii. Promotes good practices for security
        iii. Allows flexibility for building future platform agnostic clients
    c. Restful JSON
        i. Allows simple parsing by frontend javascript
        ii. Extremely fast
        iii. Object oriented data structure

3. A rationale for each design pattern used.
    a. Model View Controller
        i. Allows for modularization of code
        ii. Great for maintenance
        iii. Minimizes spaghetti code
        iv. Promotes DRY (Don’t repeat yourself) principles.
    b. Singleton
        i. Great pattern for login sessions
        ii. Great method for interactions for a current user

4. A rationale for each major UI decision.
    a. Having the menu navigation on the left side
    b. Each user has a has a unique dashboard
    c. A manager has charts from data as an overview
    d. Dropdown for categories on manager transactions
    e. Dropdown for type of transaction (Debit vs Credit)
    f.

Payday Software Architecture

Architectures used:
Client Server
4 Tier Architecture
Model View Controller



Rationale_Client_Technologies_Used

The following include rationales for the technologies used for the client side development for Payday's client server architecture.

AngularJS – Client Side Logic

- AngularJS is a client side MVC JavaScript framework. The rationale for using AngularJS for Payday is that it has an MVC architecture for client side frontend, there is familiarity with the technology by our developers, it is highly flexible and has support for plenty of third party client side modules, it allows real time data binding, it is supported by Google, it is extremely fast and allows fluid interactions that mimic native interactions. It is also mobile friendly and allows our developers to build rich mobile experiences. Finally it is open source, and free.

JQuery – Client Side Library

- JQuery is a client side library that allows functionality from plenty of third party plugins. The rationale for using JQuery for Payday is that the library is necessary for the third party library Bootstrap 3.0. Also it has great documentation, there is familiarity of the technology by our developers, it will make Payday flexible enough to be able to include third party plugins in the future, it has a simple setup, it is open source, and it is free.

Bootstrap 3.0 – Client Side Library

- Bootstrap 3.0 is a client side library that expedites frontend design and UI interactions. The rationale for using Bootstrap 3.0 is that there is familiarity with the technology by our developers, it has a simple setup, it has great documentation, it is highly flexible, it is supported by Twitter, it allows our developers to build rich mobile experiences, it expedites frontend client side design and user interface interactions. It is also open source and free.

ChartsJS – Client Side Library

- ChartsJS is a JavaScript library to build graphical charts. The rationale for using ChartsJS for Payday is that it has the best documentation out of competing chart building libraries, it has a simple setup, it allows for Payday to quickly and efficiently include build charts, it is open source, and it is free.

Rationale_Design_Patterns_Used

The following include rationales for the design patterns used by Payday.

Server Side Model View Controller (MVC)

- A model view controller design pattern is a system that allows controller layers to serve and manipulate data from model layers, then serve the data and take inputs from view layers. Payday's server side MVC design includes models that contain data objects that are built from the database and controllers serving data to views that are Restful JSON endpoints. The rationale for using a model view controller design pattern within the server side logic for Payday is that it allows for modularization of code, it promotes better maintenance practices, minimizes spaghetti code and promotes DRY (Don't Repeat Yourself) principles.

Client Side Model View Controller (MVC)

- Payday's client side MVC design includes models that contain data objects that are built from GET, PUT, POST, and UPDATE HTTP requests from the server side logic that serves the Restful JSON endpoints as views. Controllers then serve data to views that are contained in the presentation client layer of the application. The rationale for using a model view controller design pattern within the client side logic for Payday is that it is a system that allows controller layers to serve and manipulate data from model layers, then serve the data and take inputs from view layers. It also allows for modularization of code, it promotes better maintenance practices, minimizes spaghetti code and promotes DRY (Don't Repeat Yourself) principles.

Singleton

- A singleton is a design pattern that allows a master object control logic that is required throughout the entire use of an application. The rationale for Payday to use a singleton design pattern is that it is the best implementation for application wide login sessions, interactions, and manipulation of data by a current logged in user object.

Rationale_Server_Technologies_Used

The following include rationales for the technologies used for the server development for Payday's client server architecture.

Amazon EC2 Cloud – Virtual Host

- Amazon EC2 Cloud is a virtual cloud host designed to provide computing power based on demand. The rationale for using Amazon EC2 Cloud as a virtual host for Payday is that there is familiarity with the technology by our developers, it has an easy to use console, it is highly secure, the support and community is unparalleled in the industry, there is a highly active community, and it is free for one year with new accounts.

Ubuntu Server 12.04 LTS Linux – Operating System

- Ubuntu Server 12.04 LTS is a Linux distribution for a server environment. The rationale for using Ubuntu Server is that it is highly portable supports UNIX commands, there is familiarity with the technology by our developers, it is easy to set up, the documentation is unparalleled in the industry, there is a highly active community, and it is free.

Apache HTTP Web Server – HTTP File Server

- Apache HTTP Web Server is a file server designed to serve files from a directory tree structure. The rationale to use Apache HTTP File Server is that it works well out of the box with Ubuntu Server, it has a simple set up, there is familiarity with the technology by our developers, it is a mature technology, and it is free.

MySQL – Database

- MySQL is an open source relational database that is supported by Oracle. The rationale to use MySQL for Payday is that is a relational database, which handles account transactions well which is a requirement for Payday. Also it supports SQL commands, there is a familiarity with the technology by our developers, it is open source, it supported by Oracle, the documentation is unparalleled in the industry, there is a highly active community, it is simple to set up and works well out of the box with Apache HTTP Web Server, the technology is highly mature, and it is free.

Gradle – Build Automation

- Gradle is a build automation tool for testing, packaging, and publishing Java and JVM for the web. The rationale for using Gradle for Payday is that it is supported by Spring.io, it is simple to use for publishing Java for the web, it is an industry standard for java testing automation, it has a simple set up, it is open source, the documentation is unparalleled in the space, and it is free.

Spring.io – Backend Logic

- Spring.io is an enterprise level java MVC web framework. The rationale for using Spring.io for Payday is that it is a modern Java MVC framework, it has the best and easiest to follow documentation out of all the enterprise level Java frameworks, it has great support, simple set up, and a highly active community. Spring.io is also open source and completely free to use.

Use Case Details:

RTM Entry:

Use Case ID:

Use Case Name:

Use Case Primary Actors:

Preconditions:

Successful Postconditions:

Entry Condition

Exit Condition

Quality Constraints

Related Use Cases:

Step	Scenerio Action
1	Manager logs into website (See
2	Manager selects the Employee
3	Manager selects the desired Er
4	Manager will initiate View Empl
5	Payday will display the currently

Payday shall provide functions for viewing employee information.

UC_21_ViewEmployeeInformation

Manager

User must be logged into Payday system and have manager user privileg

The selected employee information will be displayed.

Manager activates the View Employee Information menu item.
When Manager closes the browser, presses the back button, or initiates the Log Off option

Will display requested information within 5 minutes

UC_11_EmployeeManagement, UC_13_EmployeeManagementDashboa

Flow of Events

e UC_1_UserLogin)

Tab (See UC_11_EmployeeManagement

mployee (See UC_13_EmployeeManagementDashboard)

oyee Information option

y selected employee's personel information

1

ges

ard

Expected Behavior

UC_ViewEmployeeInfo Rationale

Per our RAD, Payday must provide means for storing/retrieving employee personnel data. This is important in an accounting system since personnel info is required for tax/legal purposes and can also prevent confusion between employees with similar names. Furthermore, by having functions that allow the retrieval and display of individual employee info we can eliminate redundant hard copy storage and allow for retrieval by authorized parties in a variety of locations. Lastly, having functions for editing that data will allow for correction of errors and updating personnel records as personal information changes as well as changes in employment.

Use Case Details:	
RTM Entry:	
Use Case ID:	
Use Case Name:	
Use Case Primary Actors:	
Preconditions:	
Successful Postconditions:	
Entry Condition	
Exit Condition	
Quality Constraints	
Related Use Cases:	

Step	Scenerio Action
1	Manager logs into website (See UC_1_
2	Payday will determine if the login crede
3	Manager will initate view revenue optio
4	Manager will initiate witch view options
5.1	Payday will initiate view yearly revenue
5.2	Payday will initiate view monthly revenu
5.3	Payday will iniate view daily revenue

Payday shall allow managers to view input in daily, monthly, and yearly revenues. Payc
21
UC_21_ViewRevenue
Manager
User must be logged into Payday system and have manager user privileges
The selected type of revenue will be displayed.
Manager activates the View Revenue Information menu item. When Manager closes the browser, presses the back button, or initiates the Log Off option
Will display requested information within 5 minutes

Flow of Events

_UserLogin)

ntials are correct

ns

s he would like to see. (Yearly, Monthly, or Daily.)

ue

day will take the data and output the yearly and monthly revenue.

Expected Behavior

Per our RAD, Payday should provide the functions to allow any authorized user to view the yearly, monthly, and daily Revenues. This is a crucial part of an accounting system because the revenue data is what is used to figure out what the company of the user is actually making once all the expenses have been paid. Payday allows the user to view the revenue in a yearly, monthly, or daily stance from the data it receives from the transaction function.

Use Case Details:	
RTM Entry:	1
Use Case ID:	1
Use Case Name:	UC_1_Login
Use Case Primary Actors:	User, Login,redirect
Preconditions:	N/A
Successful Postconditions:	The user will be directed to either the manager dashboard or the employee dashboard
a da	N/A

Primary Scenario - Successful <UC_1_Login>		
Step	Scenerio Action	Expected Behavior
1	Display the login page	When the user enters the site, they will see a page that will prompt the user for a username and password.
2	Query the database	The login class will take the user input from the login page, and form a database query.
3	Check username	The query will seek out the inputted username in the database, and check if that username corresponds to a manager or an employee.
4	Check password	When the username is found the query will then check if the corresponding password matches the inputted password
5	Redirect to appropriate dashboard	The login class will take the user input from the login page, and form a database query.
	Quality Constraints:	
	Scenerio Actions 2-5 should take under 20 seconds	
	While Scenario Actions 2-5 are executed the user will remain on the login page	

Alternative Scenario - Failure <UC_1_login>		
1	Display the login page	When the user enters the site, they will see a page that will prompt the user for a username and password.
2	Query the database	The login class will take the user input from the login page, and form a database query.

3	Check username and password	The inputted username and password will be checked with data stored in the database
4	Display error message	An error will be displayed on the login page
	Quality Constraints:	
	Scenario Actions 2-4 should take under 20 seconds	
	The error message should appear below the username and password fields	
	The username and password fields should clear when the error message appears	

UC_1_Login

- This use case details the actions taken when a user attempts to log into their personal account on the payday system.

- The entity objects for this use case are the *userObject*, the *loginObject*, and the *redirectObject*. The user object initiates the login sequence by inputting a username and a password. The login object is used by the payday system to authenticate the user. The redirect object takes the user to either their appropriate dashboard, or it will display an error message on the login page.

- The control objects for this use case are the *checkUsername* and *checkPassword* objects. These 2 objects compare what the user inputted into each field, with what information is available in the database. These control objects also initialize the *redirectObject*.

Use Case Name:

RTM Entry:

Use Case ID:

Use Case Name:

Use Case Actors:

Entry Condition:
Flow of Events:

Exit Condition:
Quality Constraints:

Payday shall provide functions for adding new employees, viewing/editing employee information, and deleting employee records.

11

UC_11_EmployeeManagement

Admin(Manager)

Admin(Manager) Enters 'Employee Management' Dashboard from the 'Manager Dashboard'

1. Admin(Manager) will open the Payday Software
2. Admin(Manager) will be greeted by the login page where he/she shall enter their Username and Password
3. Payday shall determine from the username the appropriate dashboard to redirect the Admin(Manager) in.
4. Admin(Manager) shall be redirected into the 'Manager Dashboard'
5. Admin(Manager) then will click on the 'Employee Management' GUI Link
6. The Admin(Manager) will then be greeted by another dashboard consisting of Adding/Deleting Employees, Viewing/Editing Employee Information, and Adding/Deleting Employee Records.

When Admin(Manager) closes the browser, presses the back button, or initiates the Log Off option

1. All the information shall show within 5 seconds from the time the Admin(Manager)

) clicks.

Managing Employee

Per Payday's RAD, Payday shall greet actor with log in page. Depending upon the login credentials, the actor(Manager) will be directed to the manager dashboard. From the dashboard, there will be a Employee Management directory. The directory will open up a new page with the functions Employee View and Add an Employee. Employee View will open a page with View Employee History, Prepare Paycheck, View Hourly Pay, and Daily Hours to Date. View Employee History will allow Manager to check the history of the employee, such as transaction, and pay stubs. Prepare Paycheck allows Manager to create a paycheck for the employee from the employee shift hours times the pay. View Hourly Pay checks on the employees hourly pay. Daily Hours to Date shall show the employees hours worked each day, every week. The employee shall be listed by name in the Manage Employee page.

Use Case Details:

RTM Entry:	Payday shall allow the manager and employees to change and edit their personal user settings, including password, user ID, or profile image
Use Case ID:	23
Use Case Name:	UC_23_EditUserSettings
Use Case Primary Actors:	User (Employee), User (Manager),
Preconditions:	User (Employee) or (User) Manager must be logged into an account that matches the database. The user must have the Edit User Settings screen displayed.
Successful Postconditions:	The user's new information will replace the user's old information in the database. The user will be redirected to the User Settings screen and a success notification will display.
Related Use Cases:	N/A

Primary Scenario - Successful <UC_EditUserSettings>

Step	Scenerio Action	Expected Behavior
1	Display Edit User Settings Screen	Edit User Settings Screen displays when clicking on the Edit User Settings Button on the Navigation Menu.
2	User's Current Information Displays	Once on the Edit User Settings Screen displays, the user's current information shows along with an "Edit" button.
3	User Clicks on Edit Button	Once the User clicks on the edit button, the text with user's information will transform into text input boxes with the user's current information populated within the text input boxes. A "Save" button will appear.
4	User enters appropriate information within text boxes	The user clicks on the the text boxes he/she would like to change. The current information should be able to be deletable and replaced with the new information that the User wants to use.
5	User clicks on Profile Picture	If the image value is null, the profile picture will be a generic headshot outline avatar. The file upload prompt used by the respective browser will pop up prompting the User to upload an image.
6	User uploads image	The User's input will display a thumbnail for the image uploaded
7	User clicks save	The new information for the User will replace the User's old information within the database. The User will be redirected to the new User Settings screen with the User's new information.

Alternative Scenario - Failure 1 (Wrong file upload) <UC_EditUserSettings>

1	Display Edit User Settings Screen	Edit User Settings Screen displays when clicking on the Edit User Settings Button on the Navigation Menu.
2	User's Current Information Displays	Once on the Edit User Settings Screen displays, the user's current information shows along with an "Edit" button.
3	User Clicks on Edit Button	Once the User clicks on the edit button, the text with user's information will transform into text input boxes with the user's current information populated within the text input boxes. A "Save" button will appear.
4	User enters appropriate information within text boxes	The user clicks on the the text boxes he/she would like to change. The current information should be able to be deletable and replaced with the new information that the User wants to use.
5	User clicks on Profile Picture	If the image value is null, the profile picture will be a generic headshot outline avatar. The file upload prompt used by the respective browser will pop up prompting the User to upload an image.
6	User uploads a incompatible file	The notification will pop up describing the issue with the file. The User will not be able to save the information with an incompatible file.

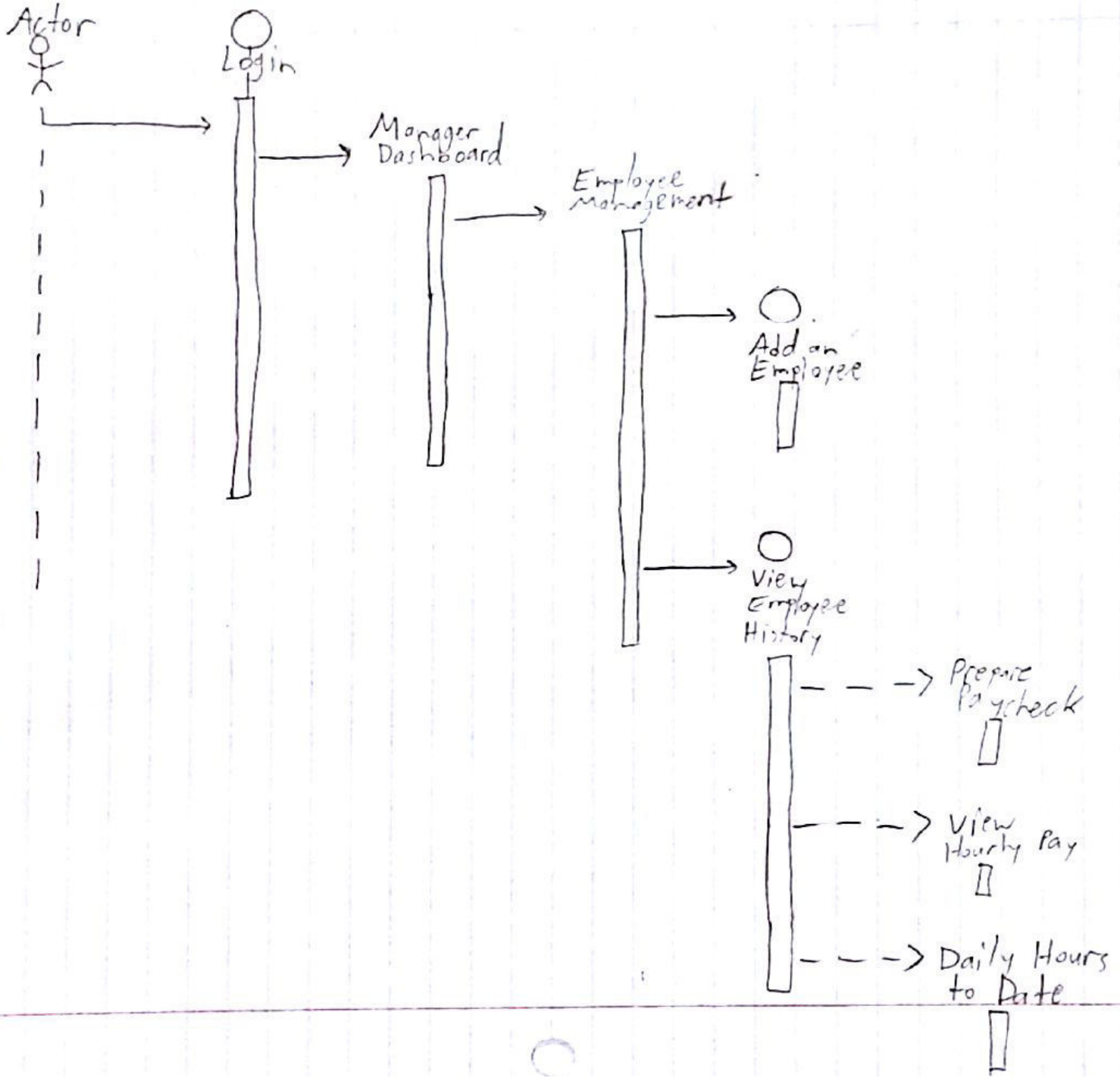
Alternative Scenario - Failure 2 (Incompatible Information) <UC_EditUserSettings>

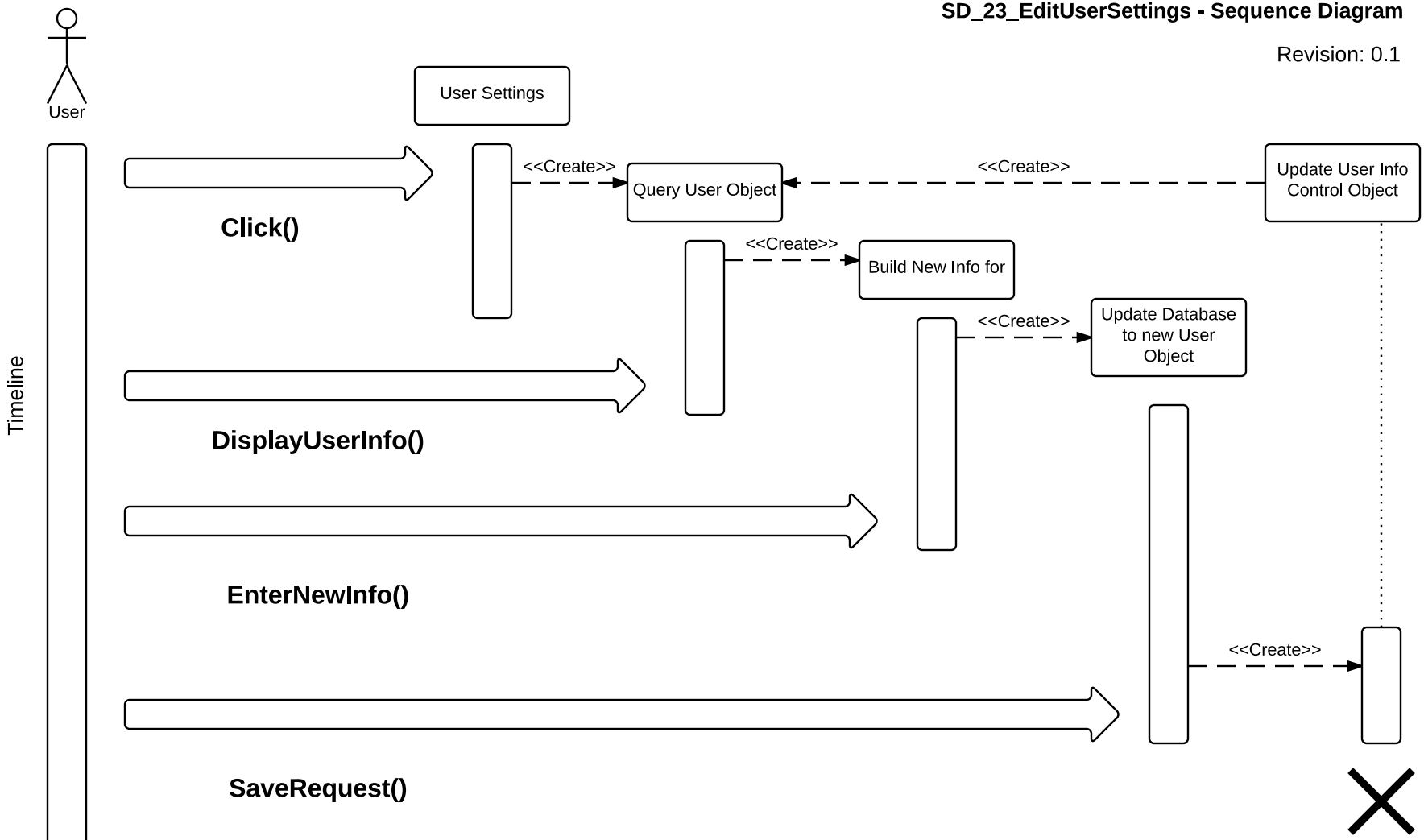
Step	Scenerio Action	Expected Behavior
1	Display Edit User Settings Screen	Edit User Settings Screen displays when clicking on the Edit User Settings Button on the Navigation Menu.
2	User's Current Information Displays	Once on the Edit User Settings Screen displays, the user's current information shows along with an "Edit" button.

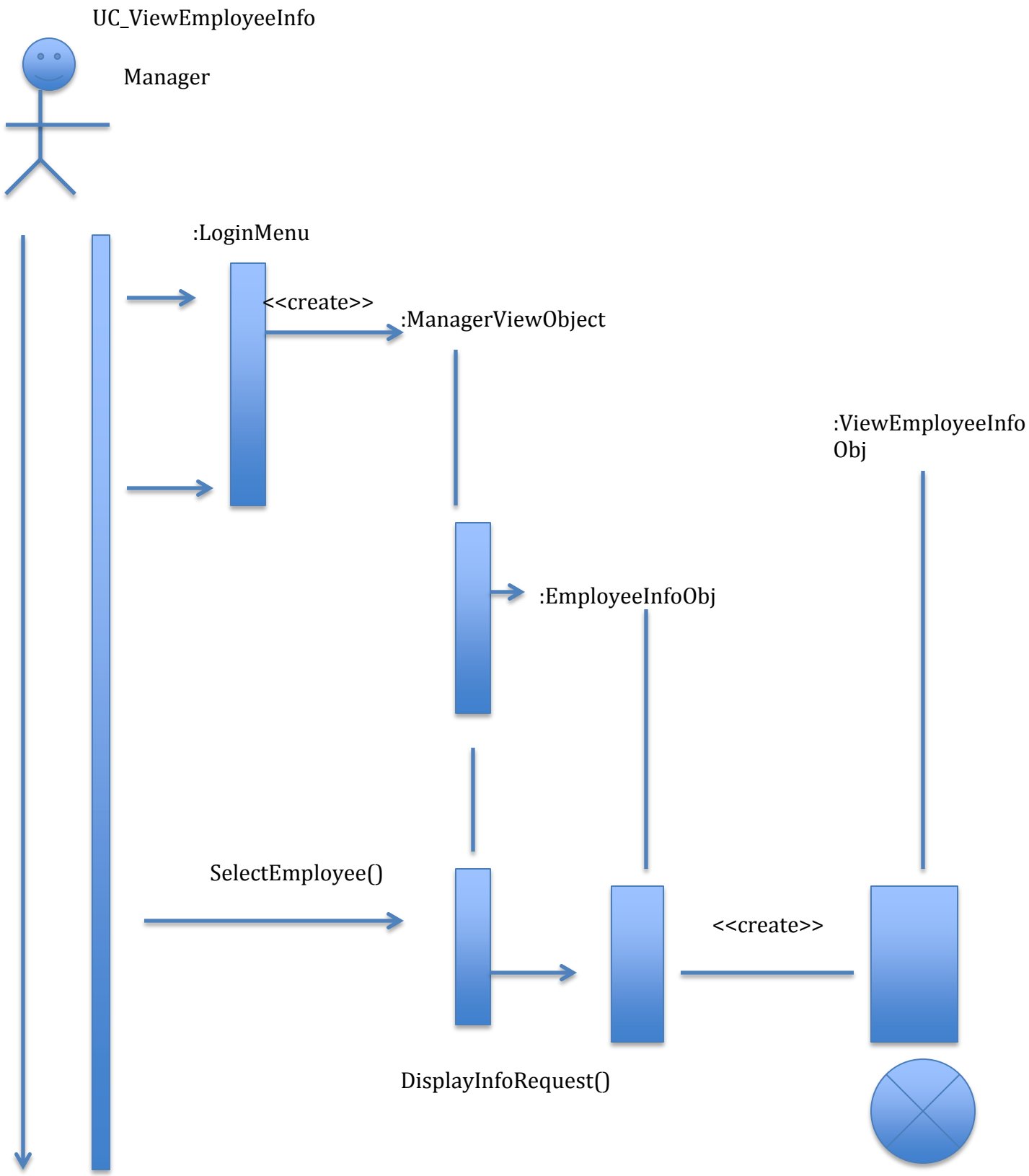
3	User Clicks on Edit Button	Once the User clicks on the edit button, the text with user's information will transform into text input boxes with the user's current information populated within the text input boxes. A "Save" button will appear.
4	User enters appropriate information within text boxes	If the information for the user is incompatible, (Address or Phone Number is in wrong format), a notification will show with a desription of the issue. The User will not be able to save until the issue is fixed.

UC_23_EditUserSettings Rationale

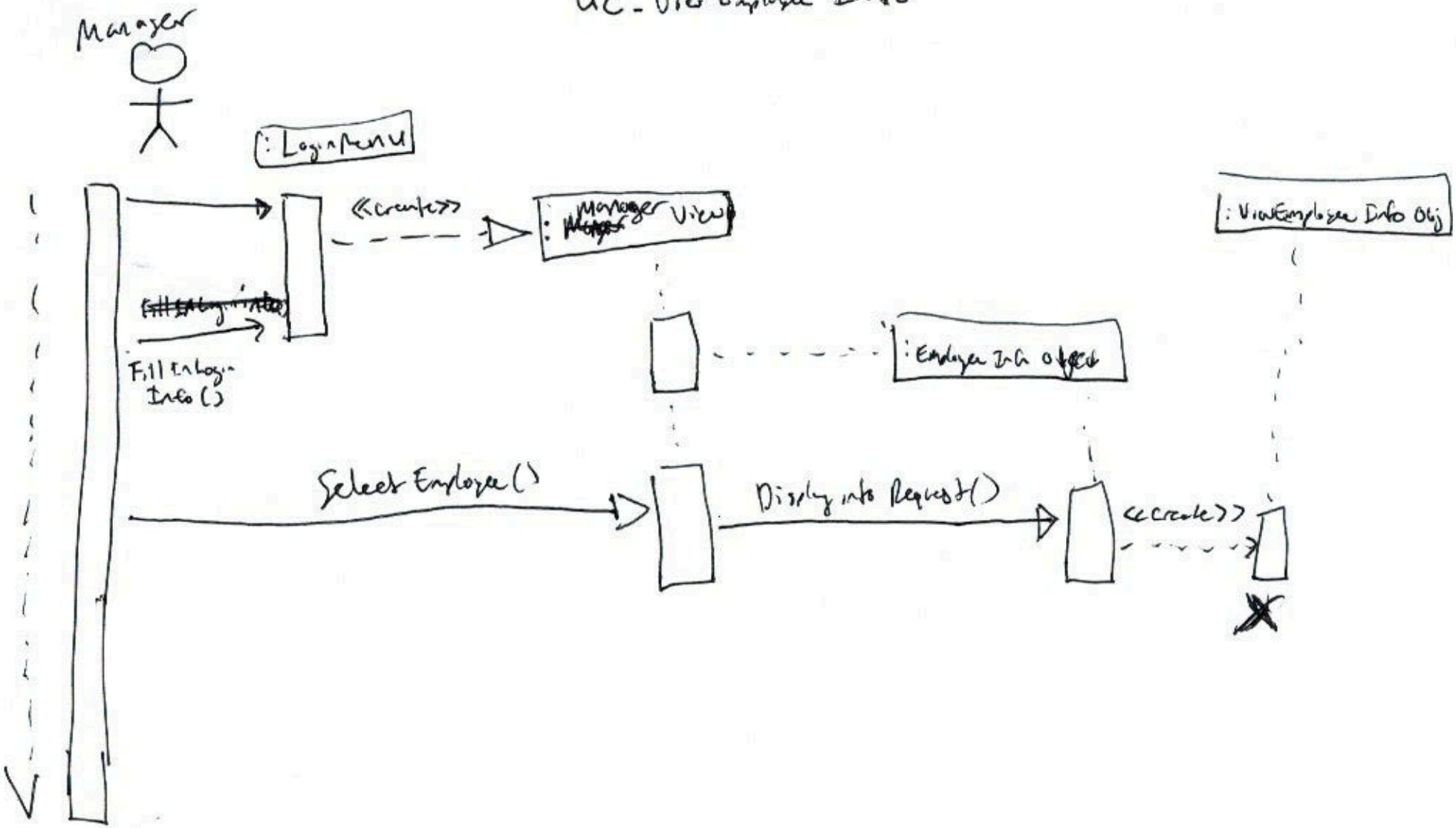
Per Payday's RAD, Payday shall allow the manager and employees to change and edit their credentials. If an employee or manager want to change their password, user ID, or image, Payday will allow the user to alter the information he wants to change. Payday shall first make certain user changing the information previously inputted is the correct user.

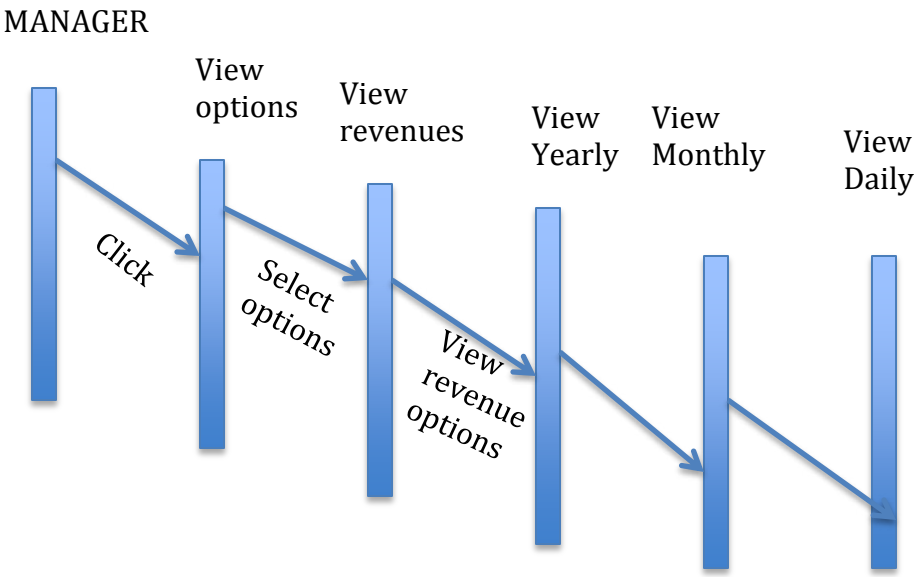


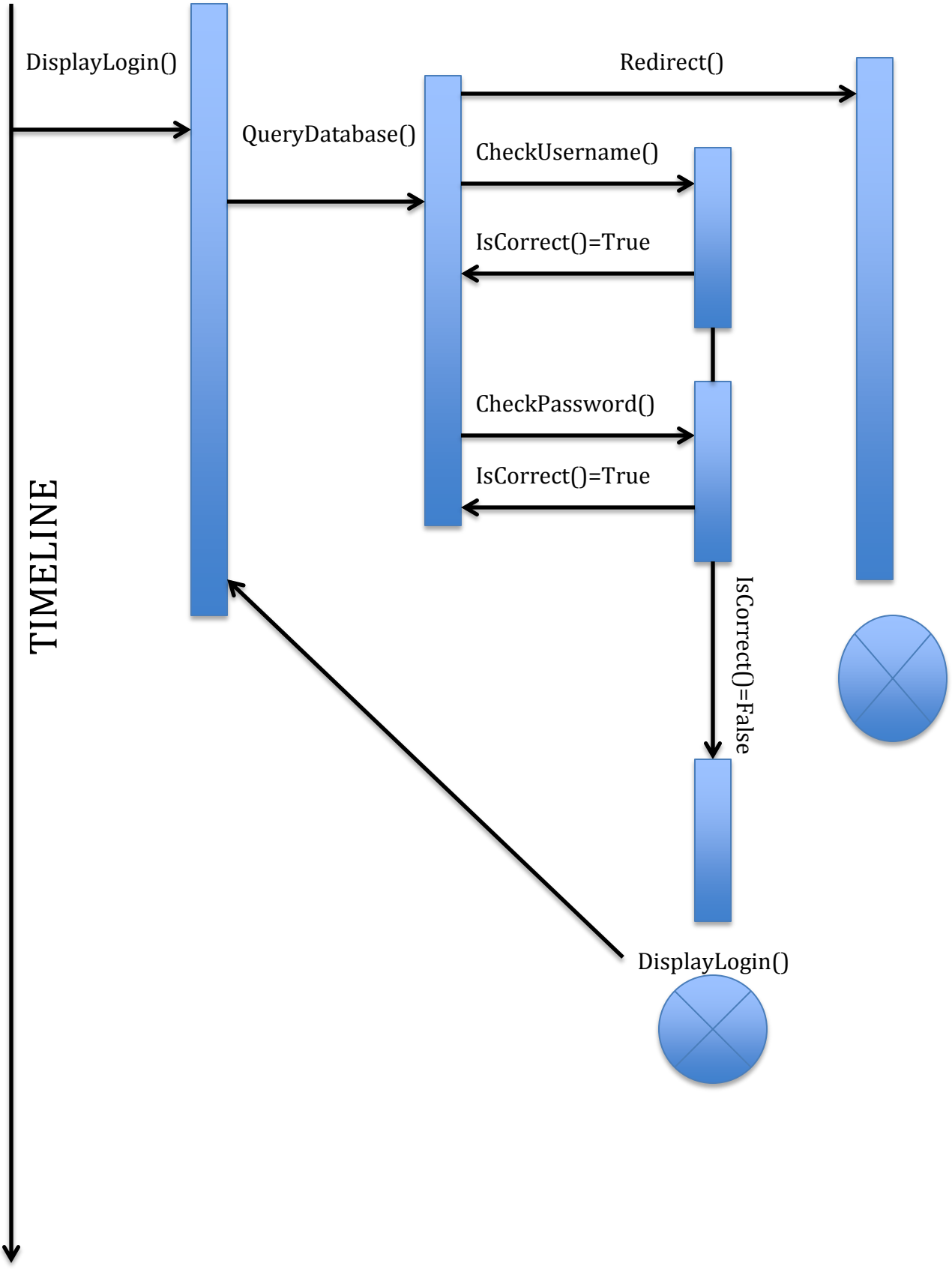

SD_23_EditUserSettings - Sequence Diagram
Revision: 0.1
User
User Settings
Query User Object
Update User Info Control Object
Build New Info for
Update Database to new User Object
<<Create>>
<<Create>>
<<Create>>
<<Create>>
<<Create>>
Timeline
Click()
DisplayUserInfo()
EnterNewInfo()
SaveRequest()


UC_ViewEmployeeInfo
Manager
:LoginMenu
<<create>>
:ManagerViewObject
:ViewEmployeeInfo Obj
:EmployeeInfoObj
SelectEmployee()
<<create>>
DisplayInfoRequest()

UC - View Employee Info




MANAGER
View options
View revenues
View Yearly
View Monthly
View Daily
Click
Select options
View revenue options


DisplayLogin()
QueryDatabase()
Redirect()
CheckUsername()
IsCorrect()=True
CheckPassword()
IsCorrect()=True
IsCorrect()=False
DisplayLogin()
TIMELINE

UC_12_RetrieveGraphData Object Design

Entity:
graphDataModel()

Methods:
- buildGraphDataController()
- readGraphDataBoundary()
- readTaxBoundary()
- readNetProfitBoundary()
- readRevenueTransactionBoundary()
- readExpenseTransactionBoundary()

Variables:
- expenses[]
- revenues[]
- taxes[]
- transactionDates[]
- categories[]
- descriptions[]
- dailyBalances[]

Entity:
netProfitModel()

Methods:
- readNetProfitBoundary()
- readRevenueTransactionBoundary()
- readExpenseTransactionBoundary()

Variables:
- expenses[]
- revenues[]
- taxes[]
- transactionDates[]

Entity:
taxModel()

Methods:
- readTaxBoundary()
- readBusinessInformationBoundary()
- readRevenueTransactionBoundary()
- readExpenseTransactionBoundary()

Variables:

- expenses[]
- revenues[]
- taxes[]
- transactionDates[]
- categories[]
- businessName
- taxId
- businessAddress

Entity:
profitLossStatementModel()

Methods:

- readNetProfitBoundary()
- readTaxBoundary()
- readBusinessInformationBoundary()
- readRevenueTransactionBoundary()
- readExpenseTransactionBoundary()

Variables:

- expenses[]
- revenues[]
- taxes[]
- transactionDates[]
- categories[]
- businessName

UC_13_EditUserSettings Object Design

Entity Objects:

Entity:
userManagerModel()

Methods:
- updateUserController()
- readUserSettingsBoundary()
- editUserSettingsBoundary()

Variables:
- userFirstName
- userLastName
- userEmail
- userPassword
- userImage
- userBio
- userBankRouting
- userBankAccount
- userPhoneNumber
- isManager=true

Entity:
userEmployeeModel()

Methods:
- updateUserController()
- readUserSettingsBoundary()
- editUserSettingsBoundary()

Variables:
- userFirstNames[]
- userLastNames[]
- userEmails[]
- userPasswords[]
- userImages[]
- userBios[]
- userBankRoutings[]
- userBankAccounts[]
- userPhoneNumbers[]
- isManager[]=[false]
- hourlyWages[]

Entity:
businessInformationModel()

Methods:

- updateBusinessInformationController()
- readBusinessInformationBoundary()
- editBusinessInformationBoundary()

Variables:

- businessName
- taxId
- businessAddress
- businessPhoneNumber
- address
- city
- state
- zipCode

Suraj Sequeira

Methods

GetProfitLossStatement
SelectSoleProprietorship
SelectPartnership
SelectCCorporation
SelectSCorporation
SelectedLimitedLiabilityCorporation
GetBusinessAddress

Entity Objects

Netprofitobject
TaxesObject

Control Objects

CheckProfitLossStatementObject
CheckTaxesObject

Boundary Objects

ViewProfitLossObjects
ViewTaxesObject

UC_11_Transactions Object Design

Entity:
revenueTransactionModel()

Methods:
- updateRevenueTransactionController()
- readRevenueTransactionBoundary()
- createRevenueTransactionBoundary()
- editRevenueTransactionBoundary()
- deleteRevenueTransactionBoundary()

Entity:
expenseTransactionModel()

Methods:
- updateExpenseTransactionController()
- createExpenseTransactionBoundary()
- readExpenseTransactionBoundary()
- editExpenseTransactionBoundary()
- deleteExpenseTransactionBoundary()

Variables:
- expenses[]
- revenues[]
- taxes[]
- transactionDates[]
- categories[]
- descriptions[]
- dailyBalances[]

UC_12_RetrieveGraphData Object Design

Entity:
graphDataModel()

Methods:
- buildGraphDataController()
- readGraphDataBoundary()
- readTaxBoundary()
- readNetProfitBoundary()
- readRevenueTransactionBoundary()

- readExpenseTransactionBoundary()

Variables:
- expenses[]
- revenues[]
- taxes[]
- transactionDates[]
- categories[]
- descriptions[]
- dailyBalances[]

Entity:
netProfitModel()

Methods:
- readNetProfitBoundary()
- readRevenueTransactionBoundary()
- readExpenseTransactionBoundary()

Variables:
- expenses[]
- revenues[]
- taxes[]
- transactionDates[]

Entity:
taxModel()

Methods:
- readTaxBoundary()
- readBusinessInformationBoundary()
- readRevenueTransactionBoundary()
- readExpenseTransactionBoundary()

Variables:
- expenses[]
- revenues[]
- taxes[]
- transactionDates[]
- categories[]
- businessName
- taxId
- businessAddress

Entity:
profitLossStatementModel()

Methods:
- readNetProfitBoundary()
- readTaxBoundary()
- readBusinessInformationBoundary()
- readRevenueTransactionBoundary()
- readExpenseTransactionBoundary()

Variables:
- expenses[]
- revenues[]
- taxes[]
- transactionDates[]
- categories[]
- businessName

UC_13_EditUserSettings Object Design

Entity Objects:

Entity:
userManagerModel()

Methods:
- updateUserController()
- readUserSettingsBoundary()
- editUserSettingsBoundary()

Variables:
- userFirstName
- userLastName
- userEmail
- userPassword
- userImage
- userBio
- userBankRouting
- userBankAccount
- userPhoneNumber
- isManager=true

Entity:
userEmployeeModel()

Methods:
- updateUserController()
- readUserSettingsBoundary()
- editUserSettingsBoundary()

Variables:
- userFirstNames[]
- userLastNames[]
- userEmails[]
- userPasswords[]
- userImages[]
- userBios[]
- userBankRoutings[]
- userBankAccounts[]
- userPhoneNumbers[]
- isManager[]=[false]
- hourlyWages[]

Entity:
businessInformationModel()

Methods:
- updateBusinessInformationController()
- readBusinessInformationBoundary()
- editBusinessInformationBoundary()

Variables:
- businessName
- taxId
- businessAddress
- businessPhoneNumber
- address
- city
- state
- zipCode

Object Design For UC #1

Redirect

Methods
- getCheckUsername()
- getCheckPassword()
- sendManagerpane()
- sendEmployeepane()

Check Username

Methods
- retrieveUserInfo()
- callRedirect()

Check Password

Methods
- retrieveUserInfo()
- callRedirect()

Object Design for UC #8

Redirect

Methods

- getemployeeName()
- getclockIn()
- getclockOut()
- callDifference()
- displayDifference()

Calculate Time Difference(callDifference calculates difference of Clock Times).

Methods

- RequestemployeeHours()

Values

- employeeName
- clockIn
- clockOut

UC_11_Transactions Object Design

Entity:
revenueTransactionModel()

Methods:
- updateRevenueTransactionController()
- readRevenueTransactionBoundary()
- createRevenueTransactionBoundary()
- editRevenueTransactionBoundary()
- deleteRevenueTransactionBoundary()

Entity:
expenseTransactionModel()

Methods:
- updateExpenseTransactionController()
- createExpenseTransactionBoundary()
- readExpenseTransactionBoundary()
- editExpenseTransactionBoundary()
- deleteExpenseTransactionBoundary()

Variables:
- expenses[]
- revenues[]
- taxes[]
- transactionDates[]
- categories[]
- descriptions[]
- dailyBalances[]

UC_8_Employee_Hours

The purpose of this use case is to provide the Managing User the ability and access to gather the total amount of hours that each Employee has worked in a shift, in a function. This Function works by getting the initial clock-in time and the clock-out time, the function will then calculate the difference between the times and display it back to the user.

The Entity Object used in this case is the employeeHours object. The employeeHours object will provide any User(Managing User or Employee) to access the Total Hours Worked function within their View Records Pane, which is located in all the Users Dashboard Panes.

The Control Object for this use case is the totalHours object. It is a simple function that the user clicks after clicking on an Employee Name and will calculate the difference between the clock in/out times by accessing the specific employees times from the database.

UC_9_ManagerPayControl

This use case provides functionality for managers to perform functions related to employee pay. Functions in this use case will allow users edit employee clock-in/clock-out times, change the number of weekly hours each employee worked, view employee pays stubs, as well as a breakdown of each employees hours worked and payment history

The entity object for this use case are the userObject . The user object confirms the current user is authorized to view employee records.

The control objects for this use case are the viewEmployeeRecord and editEmployeeRecord objects. The viewEmployeeRecord object will take the user's request, poll the database, and display the appropriate records. The editEmployeeRecord object will take the users request and perform the appropriate changes in the database.

The boundary objects for this use case are the employeeRecordSelectObject and employeeRecordChangeRequestObject. The EmployeeRecordSelectObject will provide the front-end functions allowing the user to select a specific employee's records for editing or display. This request will be passed to the viewEmployeeRecordObject. The employeeRecordChangeRequestObject will provide fields for the user to submit changes to the employee's work records. These changes will be passed to the editEmployeeRecord object.

UC_11_Revenue

This use case provides functionality for managers to view/add/edit/remove transactions and request revenue reports.

The entity object for this use case are the userObject . The user object confirms the current user is authorized to view employee records.

The boundary objects for this use case are the addTransactionObject, the editTransactionObject, and the viewRevenueRequestObject. The addTranactionObject will provide the necessary inputs for adding a transaction for a specific period. The editTransactionObject will provide the forms for selecting a previous transaction and modifying those tranactions, as well as deleting previously inputted transactions.. The viewRevenueRequestObject will provide the functions allowing the user to select the desired revenue period and request the report.

The control objects for this use case are the transactionEditObject and the revenueReportObject. The transactionEditObject will take the user request from both the addTransactionObject and editTransactionObject boundary objects, then create/select the appropriate records in the DB. The object will then make the changes requested by the boundary objects. The revenueReportObject will take the period specified by the user in the viewRevenueRequestObject, select the appropriate records from the DB from that period and generate a report for display to the user.

UC_12_RetrieveGraphData

This use case allows the retrieval revenue, expense, and profit data to be shown in a graph format.

The entity objects for this use case are revenueTransactionModel, expenseTransactionModel, graphDataModel, userManagerModel, netProfitModel.

The boundary objects for this use case are readGraphDataBoundary, readRevenueTransactionBoundary, readExpenseTransactionBoundary.

The control object for this use case is buildGraphDataController.

The userManagerModel initiates the boundary object readGraphDataBoundary whenever the managing user views the manager dashboard. That boundary object in turn initiates the control object buildGraphDataController, which retrieves the entity object graphDataModel which initiates the boundary objects readRevenueTransactionBoundary, readExpenseTransactionBoundary. Each of these boundary objects then retrieves the sub objects revenueTransactionModel, expenseTransactionModel, netProfitModel. Once the data is retrieved, the control object buildGraphDataController is responsible for building the data into graphs for the user's view.

UC_13_EditUserSettings

This use case allows users to change and edit their credentials if an employee or manager wants to change their password, user ID, or image.

The entity objects for this use case userManagerModel, userEmployeeModel.

The boundary objects for this use case editUserManagerBoundary, readUserManagerBoundary, editUserEmployeeBoundary, readUserEmployeeBoundary.

The control objects for this use case are updateUserManagerController, updateUserEmployeeController

When a managing user is on the Edit User Settings view, the boundary object readUserManagerBoundary will initialize. Once initialized it will retrieve the userManagerModel, which will display the information in the Edit user Settings view. Once appropriate information is edited and the manager clicks the save button, the boundary object editUserManagerBoundary will initialize, which in turn will initialize the control object updateUserManagerController. This control object will update the user data within the userManagerModel updating the database record for the manager.

When an employee user is on the Edit User Settings view, the boundary object readUserEmployeeBoundary will initialize. Once initialized it will retrieve the userEmployeeModel, which will display the information in the Edit user Settings view. Once appropriate information is edited and the employee clicks the save button, the boundary object editUserEmployeeBoundary will initialize, which in turn will initialize the control object updateUserEmployeeController. This control object will update the user data within the userEmployeeModel updating the database record for the employee.

UC_14_Net Profit

This use case details the actions that allow the manager to view all the attributes that affect the net profit.

The entity objects for this use case are the netprofitObject and taxesObject. The net profit object generates the data by reading the Profit & Loss statement. The taxes object generates it data by looking at the specific parameters of the sole proprietorship, partnership, s-corporation, c-corporation, or limited liability corporation and it also looks at the of the business in order to calculate the federal and state taxes.

The control objects for this use case are CheckProfitLossStatementObject and CheckTaxesObject. The CheckProfitLossStatementObject checking the data from the Profit & Loss Statement. The CheckTaxesObject check what the user has chosen between sole proprietorship, partnership, s-corporation, c-corporation, or limited liability corporation and also checks the location of the business.

The boundary objects are ViewProfitLossObject and ViewTaxesObject. The ViewProfitLossObject allows the user to view the business' profit and loss by viewing the Profit & Loss statement. The ViewTaxesObject allows the users to view the cost of taxes by taking the user choice of between sole proprietorship, partnership, s-corporation, c-corporation, or limited liability corporation and also the location of the business.

UC_1_Login

This use case details the actions taken when a user attempts to log into their personal account on the payday system.

The entity objects for this use case are the userObject and the loginObject. The user object initiates the login sequence by inputting a username and a password. The login object is used by the payday system to authenticate the user. The redirect object takes the user to either their appropriate dashboard, or it will display an error message on the login page.

The control objects for this use case are the checkUsername and checkPassword objects. These 2 objects compare what the user inputted into each field, with what information is available in the database. These control objects also initialize the redirectObject.

The boundary object for this use case is the loginForm. This provides the user with a form to put their username and password into

UC_2_Employee_History

This use case details the process of showing employee history

UC_2_Employee_Information

This use case details the process of showing an employee's information. Once a manager logs into their account they will be presented with the option to view an employee's history.

The entity objects for this use case are the managerObject and the employeeObject. The managerObject initiates the process of viewing an employee's information. The managerObject also designates which employee's information needs to be retrieved. The employeeObject holds the employee's information to be retrieved.

The control object for this use case is the retrieveObject. Once the managerObject initiates the process of showing an employee's information the retrieveObject will search the database, and it will return the information from the employeeObject.

The boundary object for this use case is the showEmployeeObject. This is the button the user will see on the manager pane. Once this button is pressed the show employee information process will begin.

UC_3_Expense_Reports

This use case explains the steps taken when a manager receives an expense report from the payday system.

The entity objects for this use case are the managerObject and the expenseObject. The managerObject initiates the process of receiving the expense report. The expenseObject holds all of the expenses accumulated in the payday system.

The control object for this use case is the retrieveObject. The retrieveObject gets the expenses stored in the database, and returns those expenses in the form of a expense report (i.e. The similar expenses will be grouped together, and shown in an expense report template).

The boundary object for this use case is the showExpensesObject. This is the button the user will see on the manager pane. Once this button is pressed the generate expense report process will begin.

UC_4_Edit_Employee_Functions

This use case describes the process of adding to the system and deleting employees from the system.

The entity objects for this use case are the managerObject, the employeeObject, the addEmployee Object, and the deleteEmployeeObject. The managerObject initiates the specified process and selects the employee to be edited. The employeeObject is the entity to be added or deleted from the system. The addEmployeeObject adds an employeeObject to the system, and the deleteEmployeeObject removes an employeeObject from the system.

The control object for this use case is the edit object. If the manager wants to add an employee the edit object adds the employee object to the database, it also performs a similar function if the manager wants to delete an employee

UC_5_Manager_Pane

Payday shall provide the managing user an employee management dashboard pane with Employee listed by name

This use case provides the core functionality of displaying the Employees on the Management Dashboard Pane. The Employee shall be listed by First and Last name in the Pane. The Manager will have full access to view all the employees through the Management Pane. The Manager will then be able to manage Employee Information through this Pane by clicking on the specific name.

The Entity Object for this specific use case is the viewEmployee object. viewEmployee object sets a constraint that only the Manager may view this through their Dashboard Pane.

The Control Object for this use case listEmployee object. The listEmployee object will control the display of Employee Names on the front end according to the access granted to user through the Login Page.

The Boundary Object for this use case the accessEmployee object. It will allow the user to send a request to the database when the user clicks on any Employee Name displayed in the Pane, and will lead to editing options.

UC_5_Manager_Pane

Payday shall provide the managing user an employee management dashboard pane with Employee listed by name

This use case provides the core functionality of displaying the Employees on the Management Dashboard Pane. The Employee shall be listed by First and Last name in the Pane. The Manager will have full access to view all the employees through the Management Pane. The Manager will then be able to manage Employee Information through this Pane by clicking on the specific name.

There is no object for this use case as this is a constraint that limits the view of this dashboard pane for strictly Managing Users.

UC_6_New_Employee_Information

This use case describes the core function of adding a new employee to the software database. It requires that the employee information: *Name, Address, Date of Birth, Social Security Number, Current Employment Status, and Job Title* shall be entered into the correct fields that are presented to the user. If any information in the Add Employee fields are left blank, then the user shall be presented with an "Insufficient Information" error notification and the Add Employee function shall be halted until filled out. Once the Employee is added, the Managing User shall be notified of the creation, record date/time of creation, and list the employee name under the Managing User Dashboard Pane.

The Entity Object for this use case is insufficientData object. This object shall analyze the fields required for a new employee to be added, and release an error to the user if the fields within Add Employee function are not filled.

There are no Boundary Objects or Control Objects due to this use case being a Non-Functional Constraint on the user to Add an Employee.

UC_7_Manage_Employee_Data

This use case has a function that provides the Managing User with the access to create Employee Checks/Pay Stubs, Track the hours worked by each Employee, Editing the Employee History(Hours Worked and Previous Pay Stubs), and Viewing the Employees Records.

The Entity Object for this use case is the viewEmployeeData object. This object defines the access the Managing User has to certain Employee Data. This access includes the ability to read the Employees data, print a check/pay stub based upon the hours worked data, editing the hours worked, and previous pay.

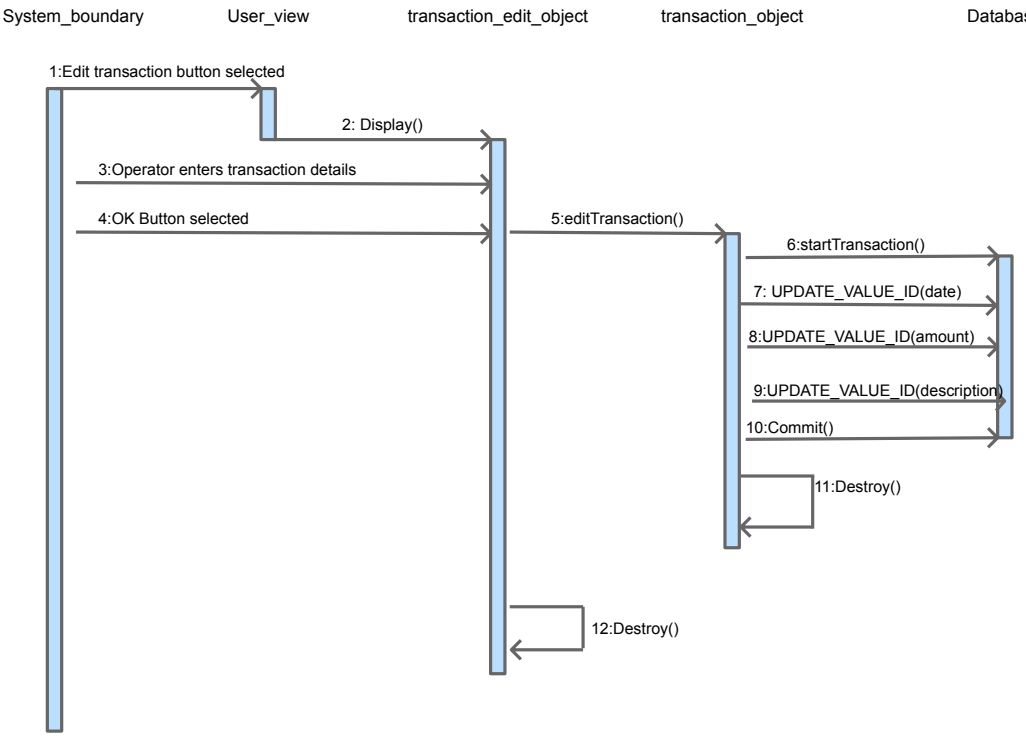
The Control Objects in this use case are printCheck object, editEmployeeHistory object, readEmployeeRecord object, and trackHours object. The printCheck object will allow the Managing User to print an Employee paycheck from the specific Employee record, according to the hours worked for the specific Employee as per the data in their database record by the click of a button. The editEmployeeHistory object gives the User access to view an Employees hours worked and their previous paychecks from the Managing Dashboard Pane, and edit it. The readEmployeeRecord object allows access for the Managing Dashboard Pane for a user to click on an Employee Name and view all of their information pulled from the database. The trackHours object allows for the Managing User to track the total number of hours worked by each Employee from the Managing Dashboard Pane, which is stored in a database.

The Boundary Objects in this used case are accessEmployee object, and editEmployeeData object. The accessEmployee object will send a request to the database when the user clicks on any Employee Name displayed in the Pane. The editEmployeeData object is the request sent to the database buffer in the software which will confirm the user to be a Managing User and respond to the request by opening a new panel for the Managing User where upon they can see the different functions involving the viewing/editing of Employee Data, and printing of their paychecks.

UC_edit Transaction_obj_diagram

Exported at: Sat Oct 18 2014 13:28:33 GMT-0400 (EDT)

Untitled Page

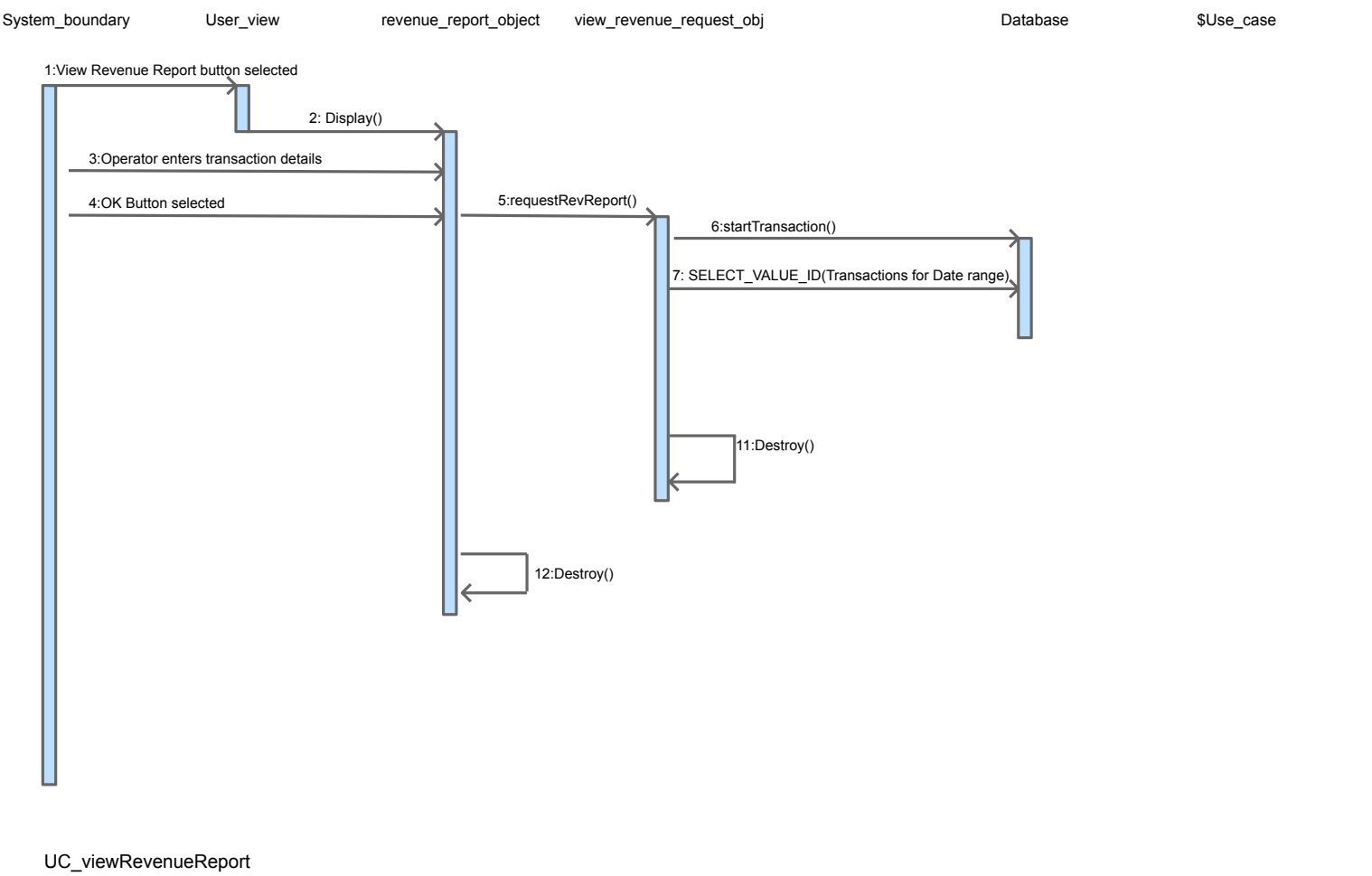


UC_editTransaction

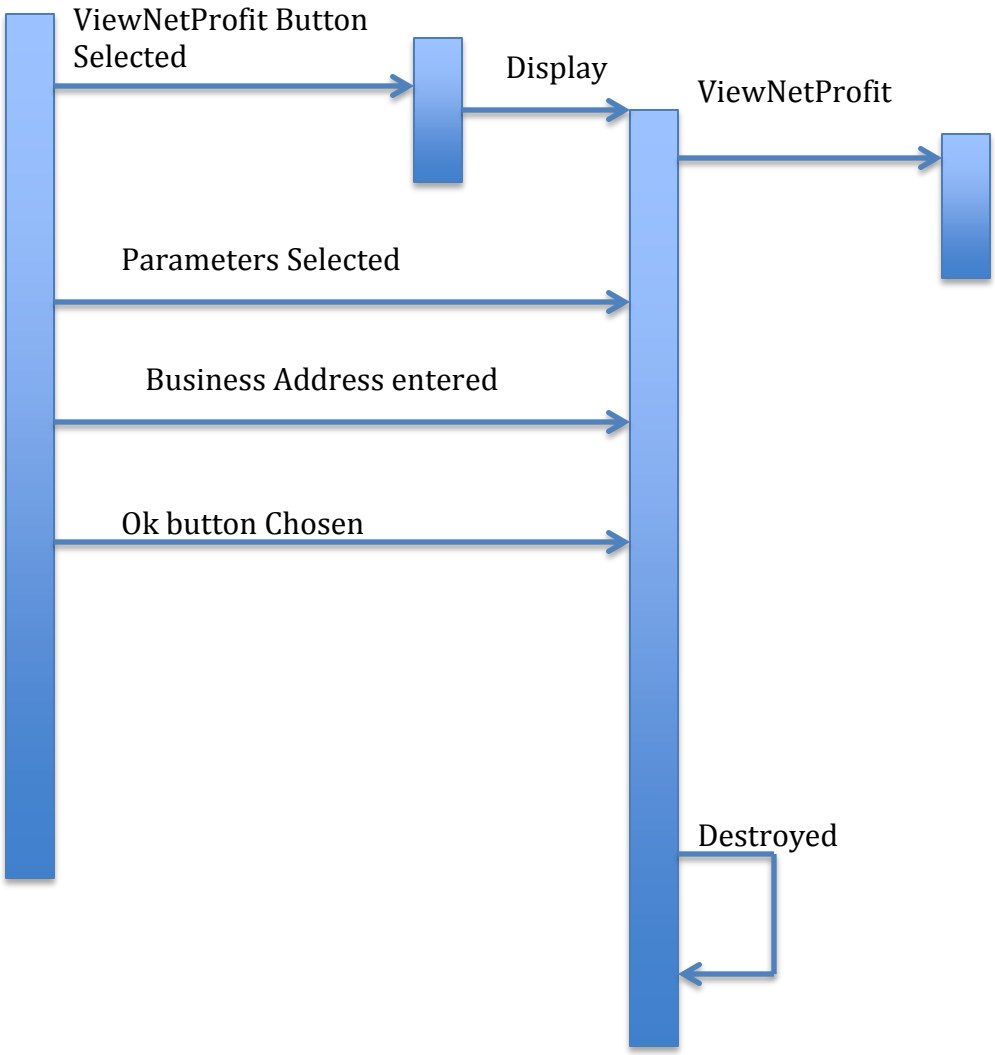
UC_view Revenue Report_obj_diagram

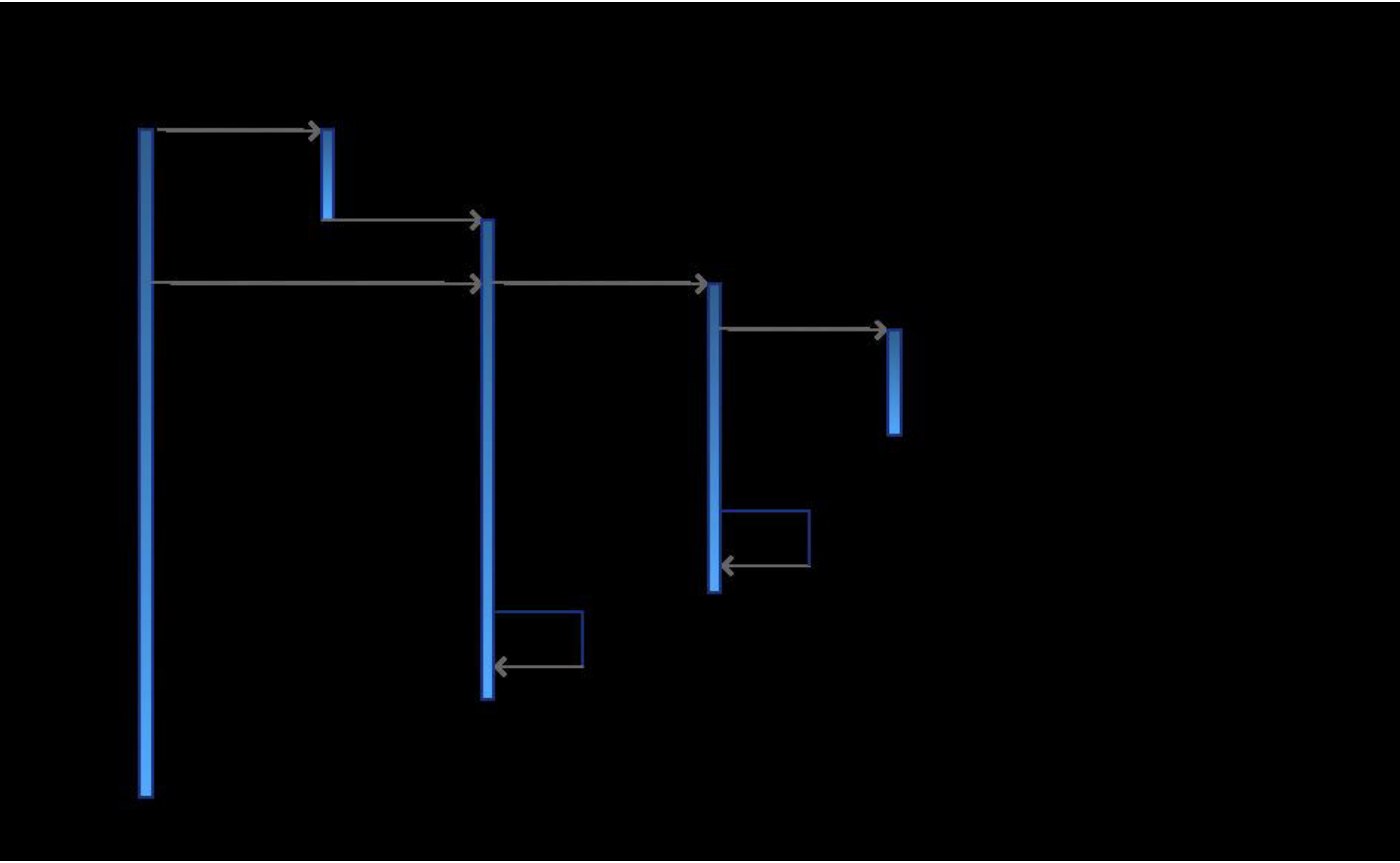
Exported at: Sat Oct 18 2014 13:40:22 GMT-0400 (EDT)

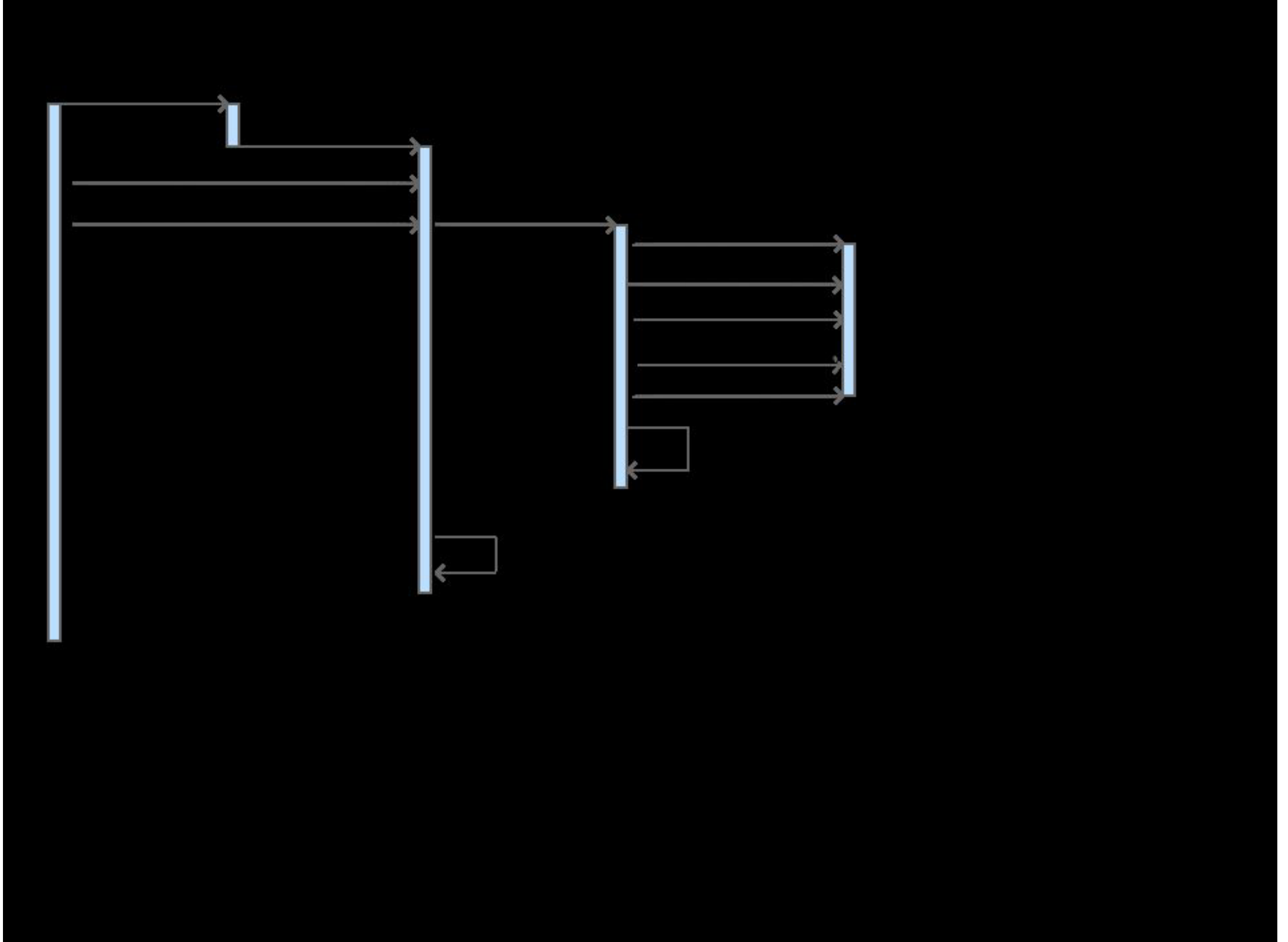
Untitled Page



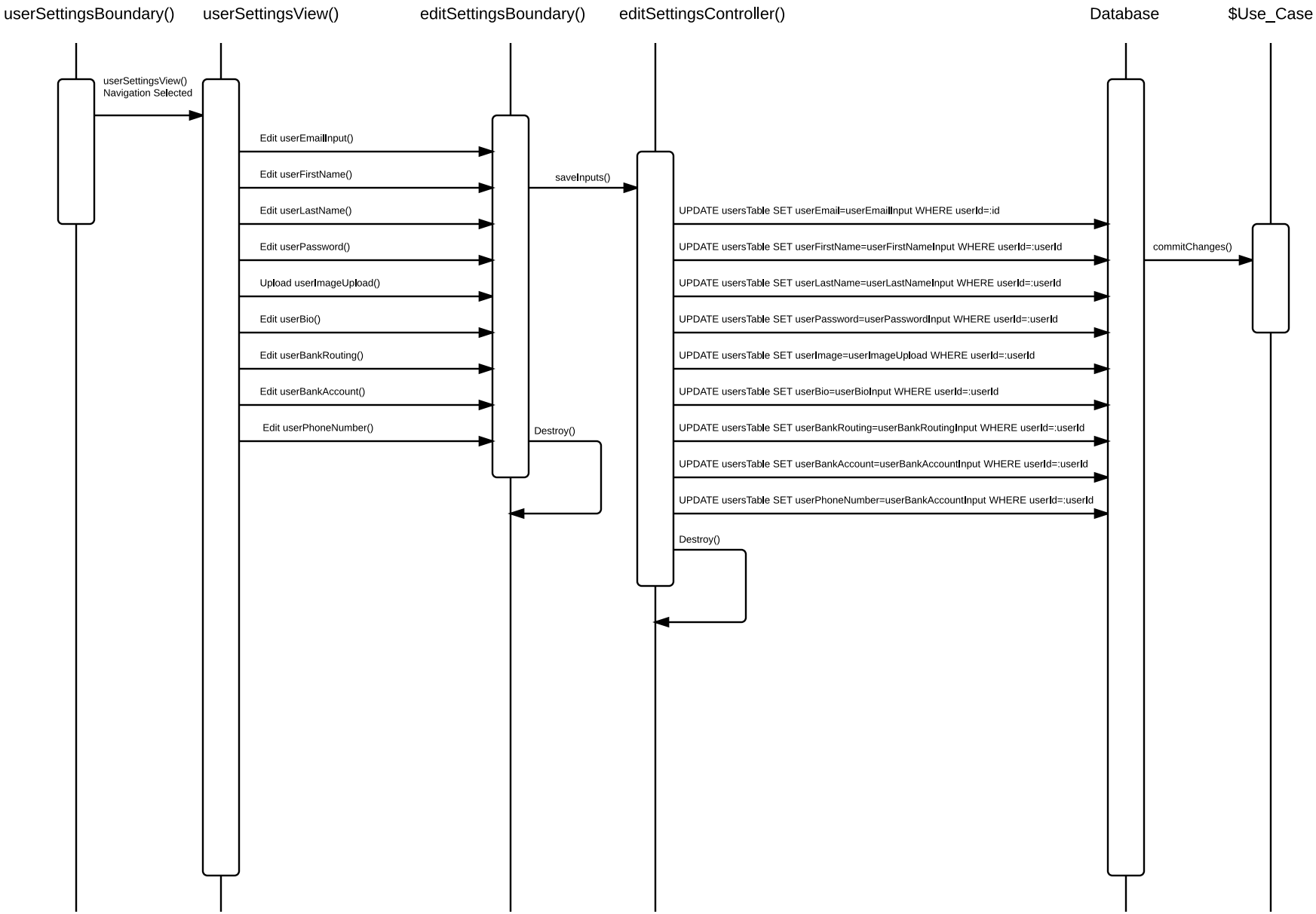
UC_viewRevenueReport


ViewNetProfit Button Selected
Display
ViewNetProfit
Parameters Selected
Business Address entered
Ok button Chosen
Destroyed

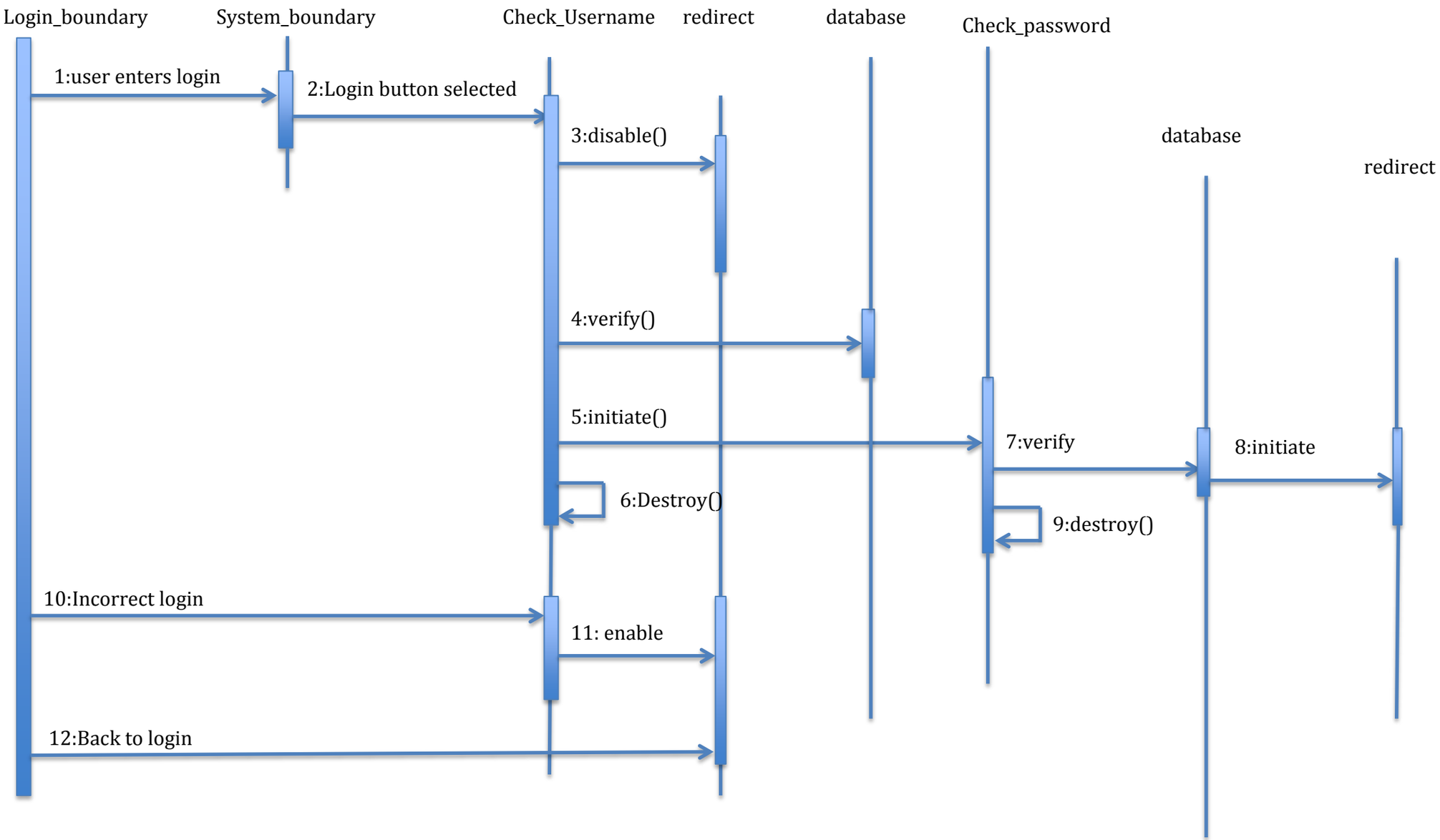




UC_13_CID_EditUserSettings



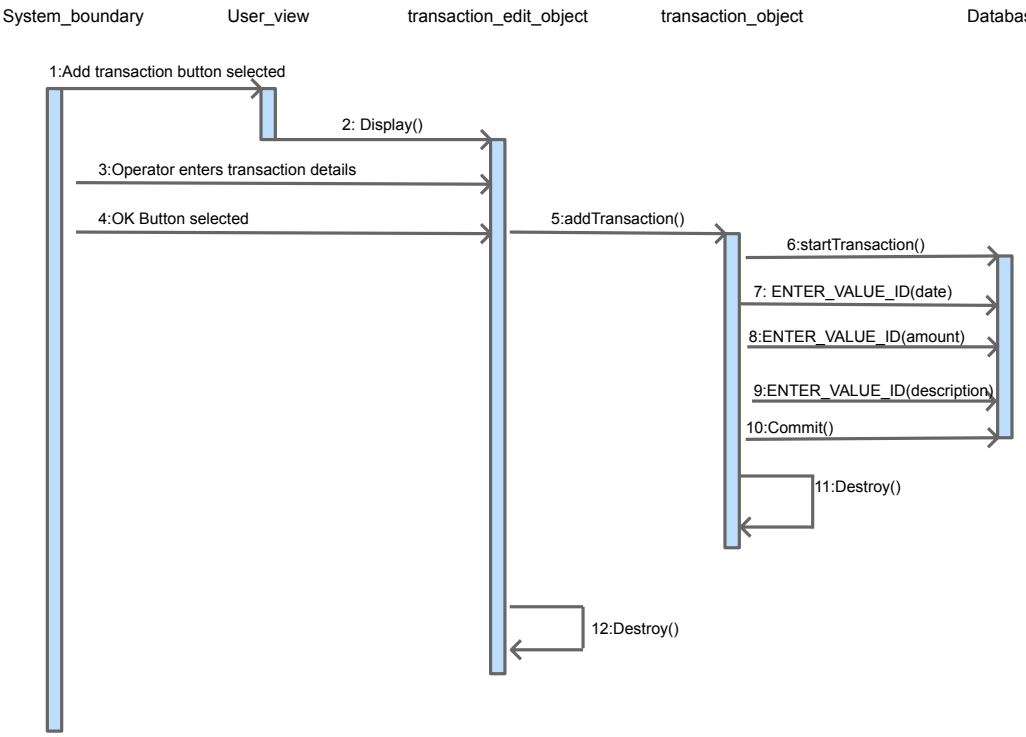
UC_1_Login


Login_boundary
System_boundary
Check_Username
redirect
database
Check_password
database
redirect
1:user enters login
2:Login button selected
3:disable()
4:verify()
5:initiate()
7:verify
8:initiate
6:Destroy()
9:destroy()
10:Incorrect login
11: enable
12:Back to login

UC_add Transaction_obj_diagram

Exported at: Sat Oct 18 2014 13:26:19 GMT-0400 (EDT)

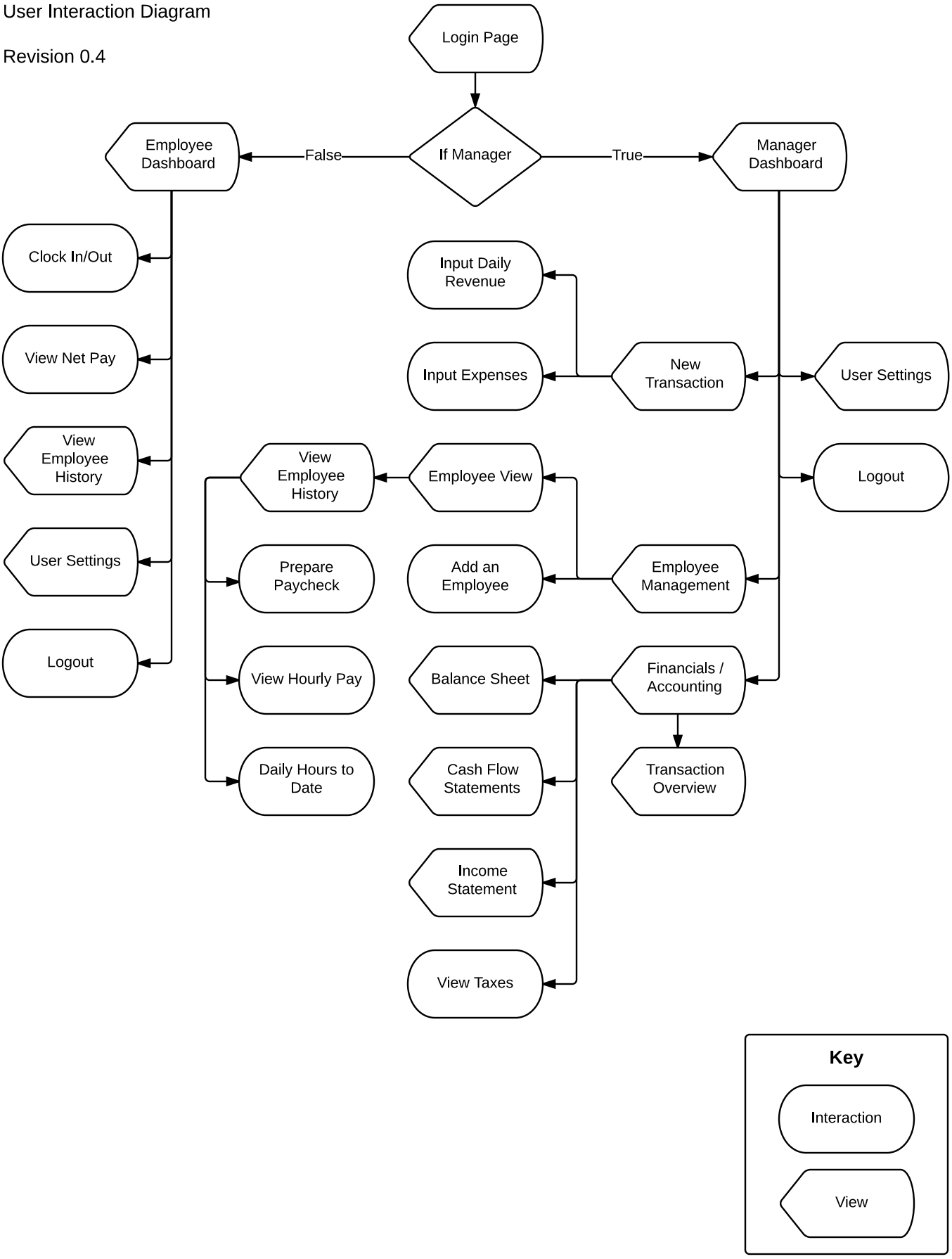
Untitled Page

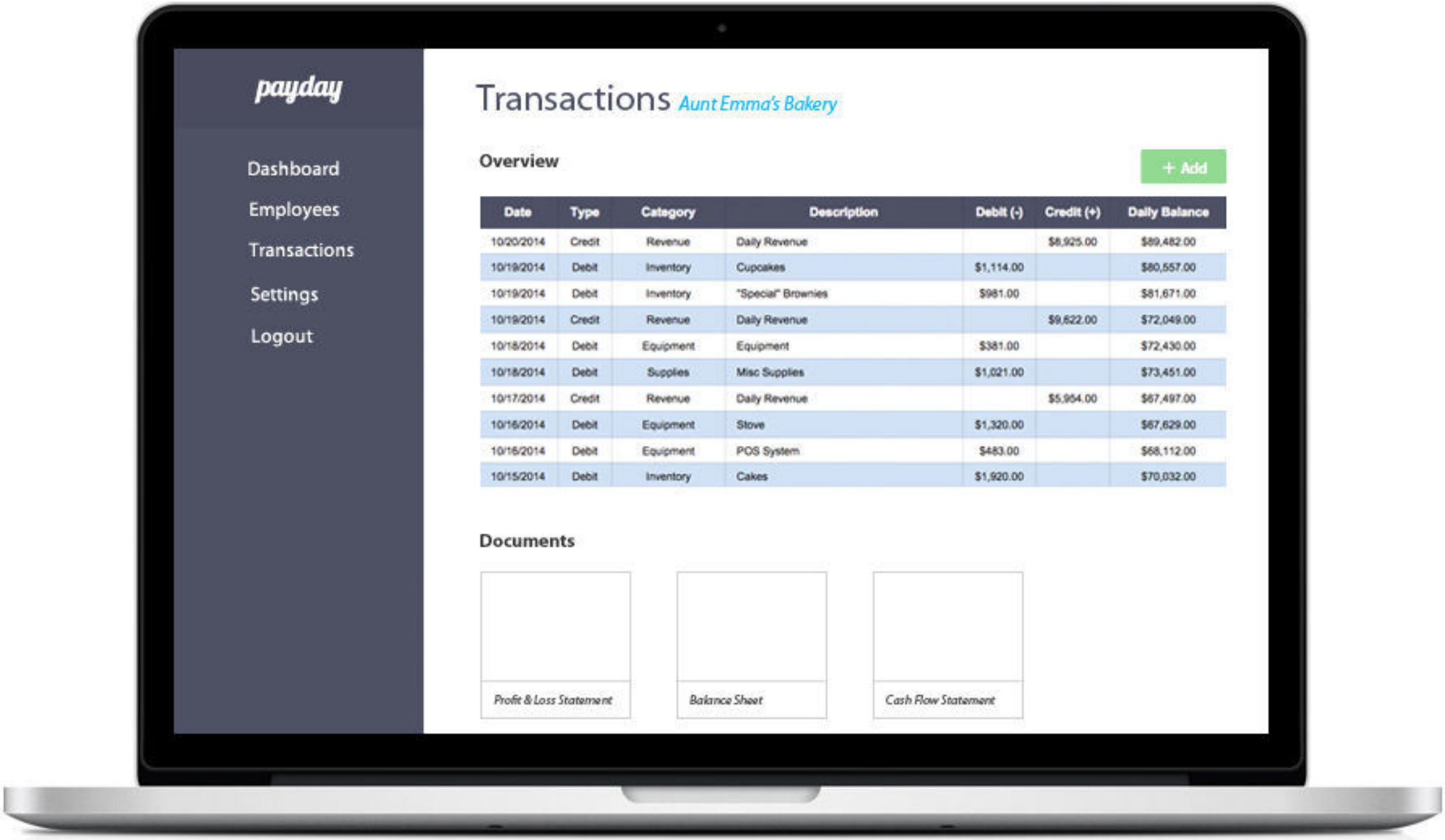


UC_addTransaction

Payday App
User Interaction Diagram

Revision 0.4




payday
Dashboard
Employees
Transactions
Settings
Logout
Transactions Aunt Emma's Bakery
Overview
+ Add
Date | Type | Category | Description | Debit (-) | Credit (+) | Daily Balance
10/20/2014 | Credit | Revenue | Daily Revenue | | $8,925.00 | $89,482.00
10/19/2014 | Debit | Inventory | Cupcakes | $1,114.00 | | $80,557.00
10/19/2014 | Debit | Inventory | "Special" Brownies | $981.00 | | $81,671.00
10/19/2014 | Credit | Revenue | Daily Revenue | | $9,622.00 | $72,049.00
10/18/2014 | Debit | Equipment | Equipment | $381.00 | | $72,430.00
10/18/2014 | Debit | Supplies | Misc Supplies | $1,021.00 | | $73,451.00
10/17/2014 | Credit | Revenue | Daily Revenue | | $5,954.00 | $67,497.00
10/16/2014 | Debit | Equipment | Stove | $1,320.00 | | $67,629.00
10/16/2014 | Debit | Equipment | POS System | $483.00 | | $68,112.00
10/15/2014 | Debit | Inventory | Cakes | $1,920.00 | | $70,032.00
Documents
Profit & Loss Statement
Balance Sheet
Cash Flow Statement

Rationale_User_Interface_Decisions

The following includes rationales for each major user interface (UI) decision in Payday.

Having the menu navigation on the left side

- The rationale for Payday to have the application menu navigation on the left side on the client views are that it allows for a much friendlier overall user experience, and it is a modern approach to user interface design.

Each user has a has a unique dashboard

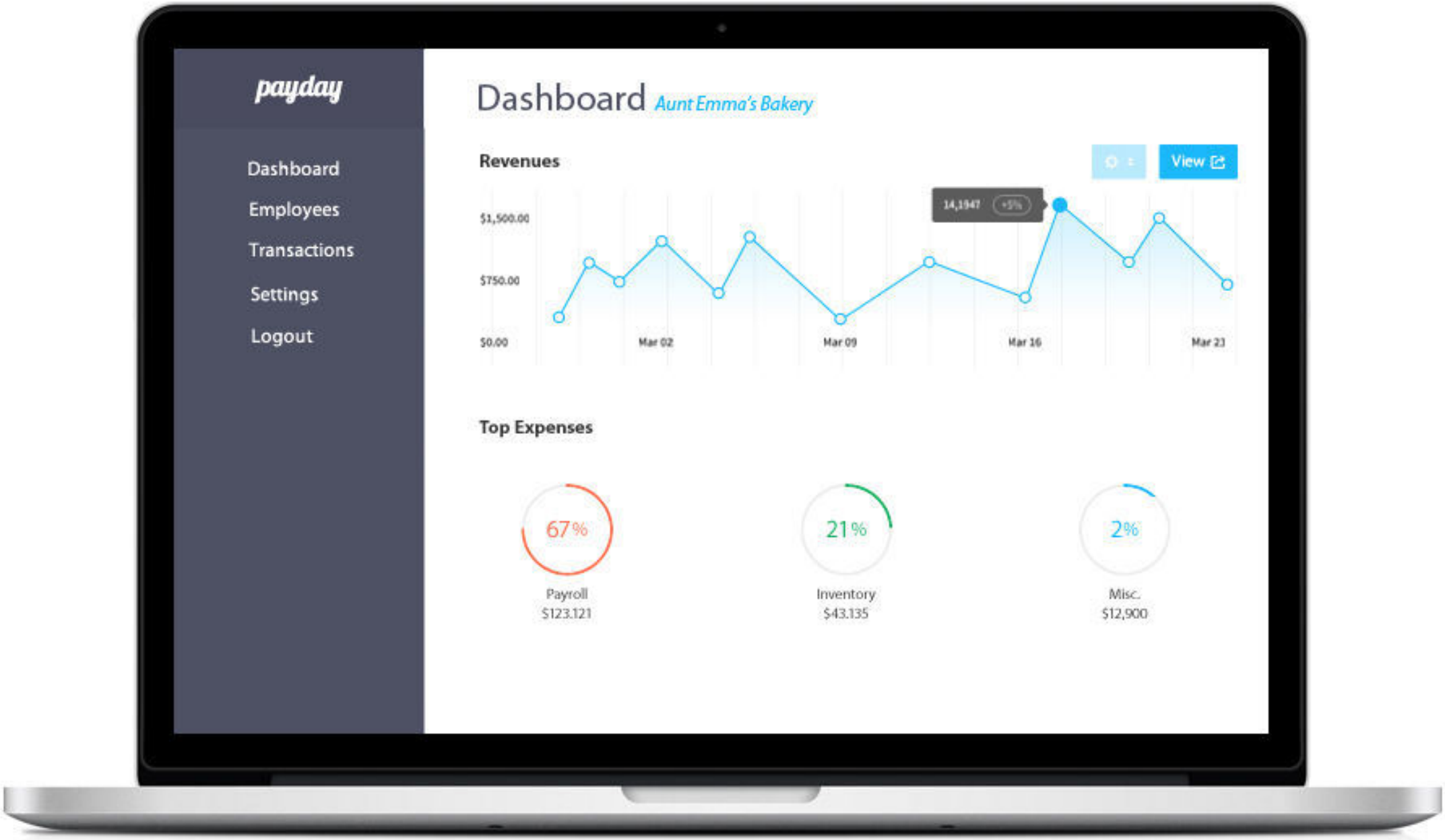
- The rationale for Payday to have the application serve a unique dashboard for each user is because it gives an overview of data that is needed by the user. The manager requires sales and expenses data, while an employee requires work history and paystub data.

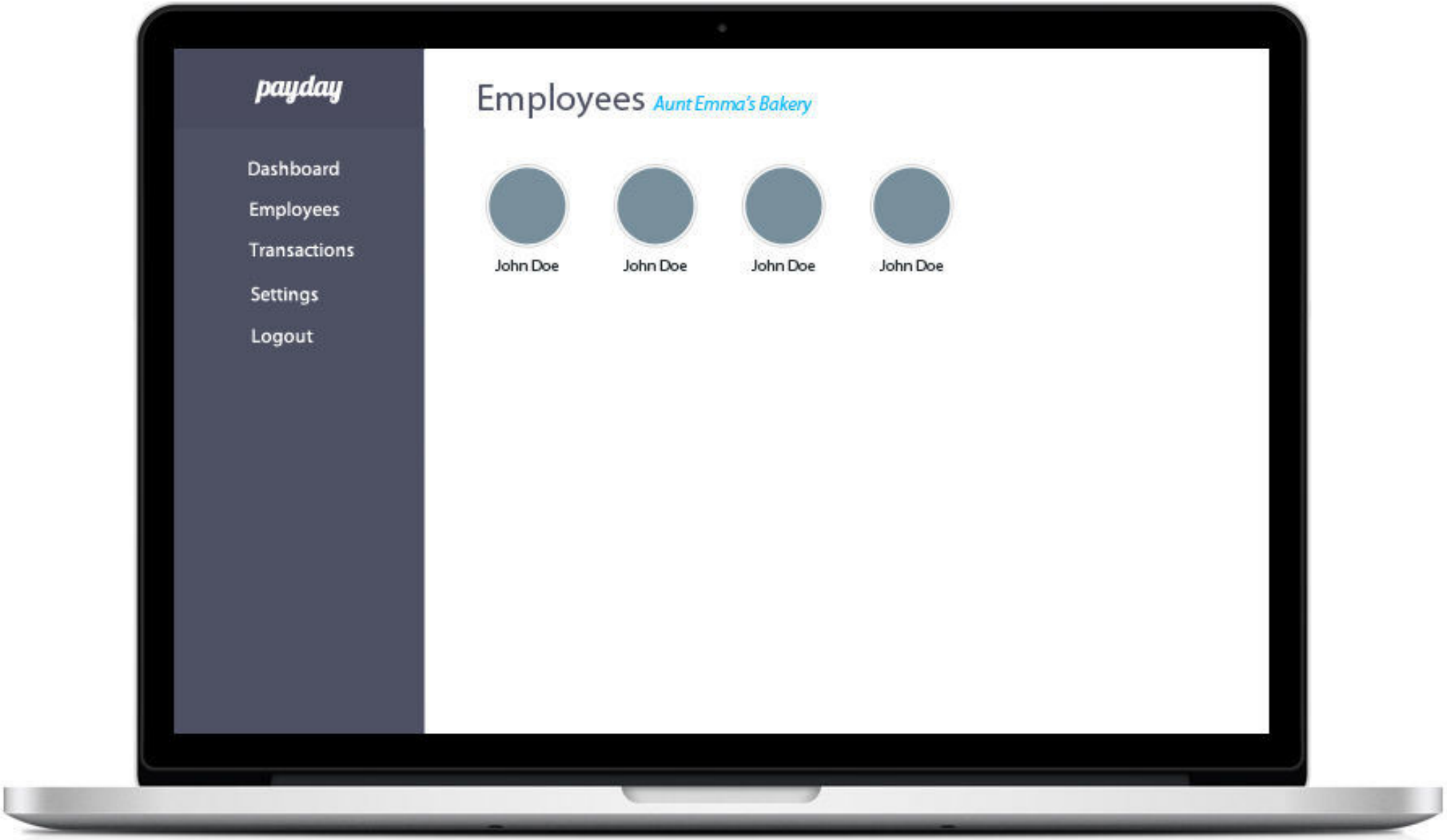
Dropdown for categories on manager transactions

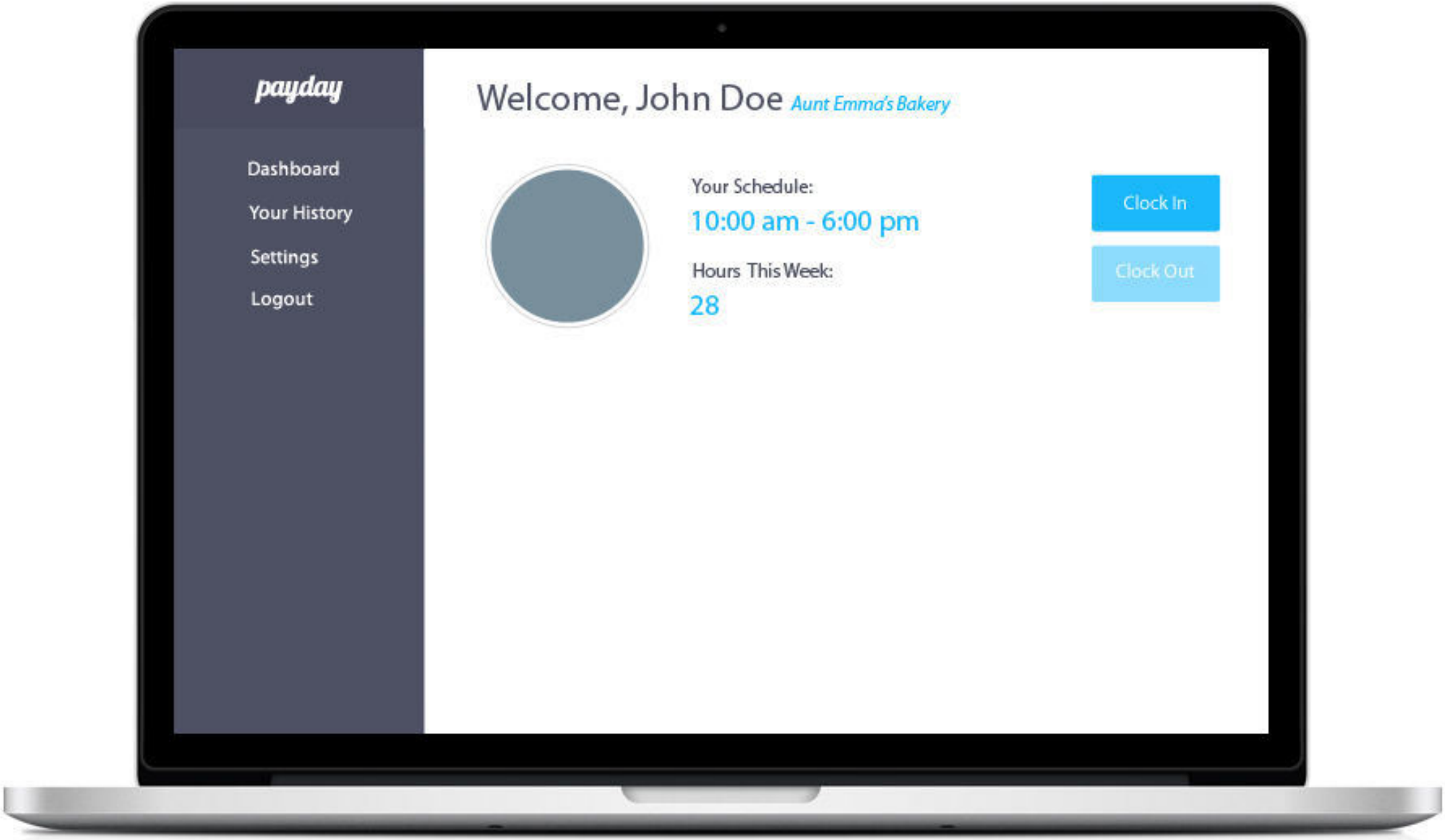
- The rationale for Payday to have a dropdown for categories when creating a new transaction on the manager transactions view is that it will be a friendlier user experience to track transactions by categories. Also categories will be able to be created by the managing user then the data of existing categories will be read by the dropdown menu.

Dropdown for type of transaction (Debit vs. Credit)

- The rationale for Payday to have a dropdown for the type of transaction is that the system reads the input through a createTransaction() method that has an if/else statement to determine if the user input of a new transaction type is a "Debit" or "Credit". The method checks against the string "Debit", so any other value will credit the manager's account, or return an error if the logic is built out in the future.


payday
Dashboard
Employees
Transactions
Settings
Logout
Dashboard
Aunt Emma's Bakery
Revenues
View
14,1947
$1,500.00
$750.00
$0.00
Mar 02
Mar 09
Mar 16
Mar 23
Top Expenses
67%
Payroll
$123.121
21%
Inventory
$43.135
2%
Misc.
$12,900


payday
Dashboard
Employees
Transactions
Settings
Logout
Employees
Aunt Emma's Bakery
John Doe
John Doe
John Doe
John Doe


payday
Dashboard
Your History
Settings
Logout
Welcome, John Doe
Aunt Emma's Bakery
Your Schedule:
10:00 am - 6:00 pm
Hours This Week:
28
Clock In
Clock Out

		Weighting Factor Es	
Measurement Parameters			
	Count	Simple	Average
Number of Inputs	11	3	4
Number of User Outputs	13	4	5
Number of User Inquiries	5	3	**4**
Number of Internal files	3	7	10
Number of External Interfaces of files	2	**5**	7
Grand Total (FP)			

FPC	264.48
Cost per FPC	$1,000
Total Cost Estimate	$264,480

timate

Complex		Total
6	=	66
7	=	91
6	=	20
15	=	45
10	=	10
	=	232

Rating Estimate of Categories

	Category	Ratings
1	Does the system require reliable backup and recovery?	5
2	Are data communications required?	4
3	Are there distributed processing functions?	2
4	Is performance critical?	4
5	Will the system run in a existing, heavily utilized operational environment?	3
6	Does the system require on-line data entry?	4
7	Does the on-line data entry require the input transaction to be built over multiple screens or operations?	2
8	Are the master files updated on-line?	4
9	Are the inputs, outputs, files or inquiries complex?	3
10	Is the interal processing complex?	3
11	Is the code designed to be reusable?	4
12	Are conversion and installation included in the design?	2
13	Is the system designed for multiple installations in different organizations?	4
14	Is the application designed to facilitate change and ease of use by the user?	5
	TOTAL sum of all category ratings	49

Function Point Analysis

To track our potential development costs, we developed a function point analysis document for the Payday online small business accounting system. To estimate the potential costs, we took into account all the requirements determined necessary by the client and compared them with the object design and system design requirements we developed. All three were used to determine the final estimate cost. First and foremost, we compiled our weighting factor estimate. Broken down into five different measurement parameters, we determined the estimate number of parameters for each category as well as the difficulty of implementation for each category.

The first measurement parameter, Number of Inputs, was estimated to be 11. We came to this conclusion utilizing the RTM document as a guideline, compiling smaller inputs into “input categories.” Since the different types of input categories vary in number of specific inputs, types of inputs, and size of inputs we determined this input category to require a “complex” weighting factor of 6.

The second measurement parameter, Number of User Outputs, was estimated to be 13. Again, utilizing the RTM document as a guideline, we compiled a list of the major outputs required to satisfy the requirements of the client. Typically, outputs of the Payday system will be some type of accounting request, such as transactions and financial reports. Since each of these various outputs will require a considerable amount of individual data, we assume the “complex” weighting factor of 7 to be appropriate.

The third measurement parameter, Number of User Inquiries, was estimated to be 5. We found in our analysis of the RTM document that the actual number of inquiries by the user was limited. Essentially, Payday will provide four major functions requiring user intervention to operate: Employee reporting, revenue reporting, transaction accounting, and employee payment. Additionally, basic user interface navigation and functionality will be required as well.

Since the system is designed for the small business owner, we determined that the system will be designed around average weighting factor estimate of 4.

The fourth measurement parameter, Number of Internal Files, was estimated to be 3. For the Payday system to meet the requirements provided by the customer, we determined the system to require three separate database tables: One for user information/tracking, one for transaction data, and one for revenue reporting. Since there is potential for a significant amount of data recorded and the individual data will comprise of many differerent components, we estimated the weighting factor to be the most complex at a value of 15.

The last measurement parameter, Number of External Interfaces of Files, was estimated to be 2. Since Payday will be using industry standard encoding of data for the web page generation and using standard database calls, we determined that we would at most have to develop 2 original external facing interfaces. The complexity should be low since we will implement standard interface designs, thus we utilized a simple weighting factor of 5.

The second major component of the Function Point Analysis was the Rating Estimate of Categories. Utilizing the requirement document, the RTM, the system design and object design we gave an estimate difficulty/importance rating for each category.

Since Payday is an accounting software platform, we will need to have both accurate and reliable record keeping. We rate this as a 5 in number of importance. Payday will be a web based accounting system, thus will require communications with outside systems. Category two was awarded an importance rating of 4. On the backend, Payday will perform simple calculations of financial transactions. Since these transactions will not require heavy processing and complex computation, we award category 3 an importance rating of 2. With that being said, performance is a must for Payday. The system will need to output data in a very efficient manner. Thus, we award category 4 a rating of 4. We don't expect the system to be used in an existing environment as it is designed for small businesses. Thus, we award category 5 a rating of 3. Payday will be online for all functions, thus we will award category 6 a rating of 4 and a

rating of 4 to category 8. Generally, most functionality will require inputs on individual screens for processing. Thus, we award a rating of 2 to category 7. All processing for Payday should not be complex, thus we award a 3 rating to both categories 9 and 10. Payday will utilize an object oriented design and thus will feature reusable code. We award category 11 a rating of 4. Installation will not be included in the design, thus category 12 is awarded a rating of 2. Since Payday is designed to be a simple small business accounting system, it will be available to many different organizations. Ease of use is a priority for a varied user base, thus we award a 4 rating to category 13 and a 5 rating to category 14.

For the final state of the Function Point Analysis, we used a cost factor of $1000 for each Function Point based on our previous experiences and current project loads. Given the total FPC estimate of 264.48, we determined the estimate cost for developing the Payday system to be $264,480.

payday

Team Rocket
Payday, Release 00.2 Alpha Web

System Test Plan
Revision 02

Specification Version: 1.0

Published: 2014

Author: Team Rocket

Table of Contents

Payday, Release 00.2 Alpha Web

Payday, Release 00.2 Alpha Web

1.0 Business Objective

1.1 The Problem

Handling large amounts of business accounting and bookkeeping data is a cumbersome process for many small businesses and e-commerce retailers. Issues for multiple businesses include handling multiple employee management, employee payroll, revenue and expense transactions, and tracking sales over multiple periods of time. Many small businesses use low-tech spreadsheet bookkeeping management techniques resulting in poor records, loss of money, and loss of time.

1.2 The Solution

An ideal solution would allow an user-friendly web based software to automate bookkeeping, manage employees and track revenue and expense transactions. Payday is a web-based accounting and employee management system developed for small businesses. Payday shall perform common accounting functions and keep track of employees.

2.0 Client Requirements

2.1 Out of Scope Assumptions

1. Payday shall not provide support for multiple businesses or multiple managing users.
2. Payday shall not provide support for tax calculations based on business profits.
3. Payday shall not provide support for generating payment stubs, employee checks, and/or direct deposit on demand.

2.2 Requirements Traceability Matrix

1. Payday shall provide a login page for users, which shall query the database to check for correct login information.
   a) Use Case Name: UC_1_Login
   b) Type: SW
   c) Functional vs. Nonfunctional: Functional

2. Payday shall keep track of employee information.
   a) Use Case Name: UC_2_Employee_Information
   b) Type: SW
   c) Functional vs. Nonfunctional: Functional

3. Upon user request Payday shall generate expense reports based upon transactions stored in the database.
   a) Use Case Name: UC_3_Expense_Reports

   b) Type: SW
   c) Functional vs. Nonfunctional: Functional

4. Payday shall provide functions for adding and deleting employee records. Payday will also provide functions to delete all records of a former employee. Payday will also notify the managing user of every creation, record date/time of employee creation, and list the employee under the main employee management dashboard pane.
   a) Use Case Name: UC_4_Edit_Employee_Functions
   b) Type: SW
   c) Functional vs. Nonfunctional: Functional

5. Payday shall provide the managing user an employee management dashboard pane with Employee listed by name.
   a) Use Case Name: UC_5_Manager_Pane
   b) Type: SW, SWC, DR
   c) Functional vs. Nonfunctional: Nonfunctional

6. Payday shall require at minimum the following employee information: Name, Address, Date of Birth, Social Security Number, Current Employment Status, and Job Title. Payday will prevent the creation of the employee record if the information asked has not been filled and provide the user with an "insufficient information" notification.
   a) Use Case Name: UC_6_New_Employee_Information
   b) Type: DWC, SW DR
   c) Functional vs. Nonfunctional: Nonfunctional

7. Payday shall provide functions for creating employee checks/pay stubs, tracking employee hours, editing employee history, and employee record viewing.
   a) Use Case Name: UC_7_Manage_Employee_Data
   b) Type: SW
   c) Functional vs. Nonfunctional: Functional

8. Payday shall provide the process of gathering the total amount of hours each employee has worked in a shift by getting the difference between his or her clock in and clock out time.
   a) Use Case Name: UC_8_Employee_Hours
   b) Type: SW
   c) Functional vs. Nonfunctional: Functional

9. Payday shall allow managers to be given access to alter/edit employee previous clock in and clock out times, number of hours worked by each employee, and quarterly employee pay stubs. Payday will also allow employees and managers to view a breakdown of each employee’s hourly work and payment history.
   a) Use Case Name: UC_9_Manager_Pay_Control
   b) Type: SW
   c) Functional vs. Nonfunctional: Functional

10. Payday shall allow the work history to be viewed as a past week, past 2 weeks, past month, or entire work history format.
    a) Use Case Name: UC_10_Pay_History
    b) Type: SWC
    c) Functional vs. Nonfunctional: Functional

11. Payday shall also calculate and allow managers to add revenue transaction and view daily, weekly, monthly, and yearly revenues.
    a) Use Case Name: UC_11_Revenue
    b) Type: SW
    c) Functional vs. Nonfunctional: Functional

12. Payday shall also take the revenue and expenses and allow the manager to view the information in a chart or graph format.
    a) Use Case Name: UC_12_RetrieveGraphData
    b) Type: SW, DR, NTW
    c) Functional vs. Nonfunctional: Functional

13. Payday shall allow the Managers and employees to change edit their personal user settings, including password, user ID, or profile image.
    a) Use Case Name: UC_13_EditUserSettings
    b) Type: SW
    c) Functional vs. Nonfunctional: Functional

14. Payday shall generate the net profit from the Profit & Loss statement of the business.
    a) Use Case Name: UC_14_ProfitLossStatement
    b) Type: SW, SWC
    c) Functional vs. Nonfunctional: Functional

3.0 Quality Assurance Guide for Test Case Review

3.1 Test Cases Review & Approval (to be done before testing):

Prepared By/Date:	Reviewed By/Date:	Approved By/Date:
Toufiq/16NOV2014	Patel/16NOV2014	Byrd/16NOV2014

3.2 Test Runs (to be filled during testing):

Run Number :	1	2	3
Conducted By:	Toufiq	Toufiq	Toufiq
Date :	16NOV2014	16NOV2014	16NOV2014

3.3 Defect Data Consolidation (to be filled after completion of testing):

Severity	High		Medium		Low		Total	
Type	Syntax	Assignment	Interface	Checking	Function	Data	System	Documentation
Cause	TRANSACTION ERROR			USER-EDUCATION / TRAINING REQ'D				

Phase of Origin							

3.4 Test Review and approval (to be filled after data consolidation):

Testing Reviewed By:	Date:	Testing Approved By:	Date:
Patel	16NOV2014	**Byrd**	16NOV2014

3.5 Test/QA Environment Information

Type	Server (Name, IP)	Schema or Link
Test Application Server	sheehantoufiq.com	http://sheehantoufiq.com/payday
Test Database Server	MYSQL	User: toufiq_payday Password: payday4350

3.6 Test/QA Credentials

Role	UserID	Password
User: Manager (Web Application)	manager@payday.com	test
User: Employee (Web Application)	*Manager must create employee as prerequisite. Manager shall determine UserID of employee.*	*Manager must create employee as prerequisite. Manager shall determine password of employee.*
User (MYSQL)	toufiq_payday	payday4350

3.7 Test Prerequisites:

This sub-section states all the key events that occurred before execution of testing.

- Brower homepage must be pointed to http://sheehantoufiq.com/payday.
- MYSQL Database must have executed schema payday_schema.sql.

3.8 Execution Details:

Execution details of testing depend on the application and can hence vary widely. The application could be a batch job, or an interactive one calling for on-line inputs, or a client-server job calling for a totally different approach. Given the wide diversity of tests, it is not advisable to impose a

standard. However, it is suggested that regardless of nature of the testing, the following minimum information be recorded about the execution details of the testing:

- Behavior expected of the system
- Evaluation procedure, for interpreting the results
- Control decisions: When to start, end, suspend, resume, rerun the test; actions to be taken in case of exceptional situations
- Any other relevant aspect

If the test designer expects the test team to record or note down the behavior observed in a particular format, then he/she should provide samples of such forms as Annexures to the test cases, along with clear instructions on how the recording should be done.

3.9 Test Cases Statuses:

Pass: Expected and observed behavior is the same.
Fail: Expected and observed behavior is different. This is classified as a defect.

Defect Classifications:

- **H** - High severity defect
  Definition: System crash / Incorrect data which impacts other regions / etc.
- **M** - Medium severity defect
  Definition: Execution can continue / Incorrect data which does not impact other regions / etc.
- **L** - Low severity defect
  Definition: Cosmetic problem: Aesthetics / Message wording / Menu options / Wrong alarms / Help problems / etc.

4.0 Test Cases

4.1 Test Case - Login

Step No.	Data Input	Expected Behavior	Observed Behavior *					
			Run 1		Run 2		Run 3	
			(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L
1.	Navigate to Login Page	Payday's Login Page Displays	Pass	None	Pass	None	Pass	None
2.	Enter Manager email in email field	Manager email displays in email field	Pass	None	Pass	None	Pass	None
3.	Enter password in password field	Dots display in password field	Pass	None	Pass	None	Pass	None

4.	Click Sign in	User is forwarded to Manager Dashboard	Pass	None	Pass	None	Pass	None
Step No.	**Data Input**	**Expected Behavior**	**Observed Behavior ***					
			Run 1		**Run 2**		**Run 3**	
			(Pass / Fail)	**H/M/L**	**(Pass / Fail)**	**H/M/L**	**(Pass / Fail)**	**H/M/L**
Prerequisites: At least one employee must exist in the database, created by a managing user.								
1.	Navigate to Login Page	Payday's Login Page Displays	Pass	None	Pass	None	Pass	None
2.	Enter Employee email in email field	Employee email displays in email field	Pass	None	Pass	None	Pass	None
3.	Enter password in password field	Dots display in password field	Pass	None	Pass	None	Pass	None
4.	Click Sign In	User is forwarded to Employee Dashboard	Pass	None	Pass	None	Pass	None
Step No.	**Data Input**	**Expected Behavior**	**Observed Behavior ***					
			Run 1		**Run 2**		**Run 3**	
			(Pass / Fail)	**H/M/L**	**(Pass / Fail)**	**H/M/L**	**(Pass / Fail)**	**H/M/L**
1.	Navigate to Login Page	Payday's Login Page Displays	Pass	None	Pass	None	Pass	None
2.	Enter User email in email field (incorrect)	User email displays in email field	Pass	None	Pass	None	Pass	None
3.	Enter password in password field	Dots display in password field	Pass	None	Pass	None	Pass	None
4.	Click Sign In	Red message displays stating that the user does not exist	Pass	None	Pass	None	Pass	None
Step No.	**Data Input**	**Expected Behavior**	**Observed Behavior ***					
			Run 1		**Run 2**		**Run 3**	
			(Pass / Fail)	**H/M/L**	**(Pass / Fail)**	**H/M/L**	**(Pass / Fail)**	**H/M/L**
1.	Navigate to Login Page	Login page displays	Pass	None	Pass	None	Pass	None
2.	Enter User email in email	User email displays in email	Pass	None	Pass	None	Pass	None

	field	field						
3.	Enter password in password field (incorrect)	Dots display in password field	Pass	None	Pass	None	Pass	None
4.	Click Sign In	Red message shows up stating that the password is incorrect	Pass	None	Pass	None	Pass	None

4.2 Test Case - Manage Employees: Create New Employee

Step No.	Data Input	Expected Behavior	Observed Behavior *					
			Run 1		Run 2		Run 3	
			(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L
Prerequisites: A managing user must be logged in.								
1.	Navigate to Employees Page	Employees Page displays	Pass	None	Pass	None	Pass	None
2.	Click "+" button	A "Create A Employee" popup modal displays	Pass	None	Pass	None	Pass	None
3.	Enter Employee's first name in first name field	Employee's first name displays in first name field	Pass	None	Pass	None	Pass	None
4.	Enter Employee's last name in last name field	Employee's last name displays in last name field	Pass	None	Pass	None	Pass	None
5.	Enter Employee’s phone number in phone number field	Employee's phone number displays in phone number field	Pass	None	Pass	None	Pass	None
6.	Enter Employee’s bio in bio field	Employee's bio displays in bio field	Pass	None	Pass	None	Pass	None
7.	Enter Employee’s wage in wage field	Employee's wage displays in wage field	Pass	None	Pass	None	Pass	None
8.	Enter Employee's email in email	Employee's email displays in email field	Pass	None	Pass	None	Pass	None

	field							
9.	Enter Employee's password in password field	Dots display in password field	Pass	None	Pass	None	Pass	None
10.	Enter Employee's password again in confirm password field	Dots display in confirm password field	Pass	None	Pass	None	Pass	None
11.	Click Create Employee	The popup modal disappears showing the new Employee added on the Employee's Page	Pass	None	Pass	None	Pass	None
Step No.	**Data Input**	**Expected Behavior**	**Observed Behavior ***					
			Run 1		**Run 2**		**Run 3**	
			(Pass / Fail)	**H/M/L**	**(Pass / Fail)**	**H/M/L**	**(Pass / Fail)**	**H/M/L**
Prerequisites: A managing user must be logged in.								
1.	Navigate to Employees Page	Employees Page displays	Pass	None	Pass	None	Pass	None
2.	Click "+" button	A "Create A Employee" popup modal displays	Pass	None	Pass	None	Pass	None
3.	Enter Employee's first name in first name field	Employee's first name displays in first name field	Pass	None	Pass	None	Pass	None
4.	Enter Employee's last name in last name field	Employee's last name displays in last name field	Pass	None	Pass	None	Pass	None
5.	Enter Employee's phone number in phone number field	Employee's phone number displays in phone number field	Pass	None	Pass	None	Pass	None
6.	Enter Employee's bio in bio field	Employee's bio displays in bio field	Pass	None	Pass	None	Pass	None
7.	Enter Employee's wage in wage field	Employee's wage displays in wage field	Pass	None	Pass	None	Pass	None
8.	Enter	Employee's email	Pass	None	Pass	None	Pass	None

	Employee's email in email field	displays in email field						
9.	Enter Employee's password in password field	Dots display in password field	Pass	None	Pass	None	Pass	None
10.	Enter Employee's password again in confirm password field (incorrect)	Dots display in confirm password field. Red message displays stating that the passwords have to be the same	Pass	None	Pass	None	Pass	None

4.3 Test Case - Manage Employees: View All Employees

Step No.	Data Input	Expected Behavior	Observed Behavior *					
			Run 1		Run 2		Run 3	
			(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L
Prerequisites: At least one employee must exist in the database, created by a managing user.								
1.	Navigate to Employees Page	All existing employees are displayed	Pass	None	Pass	None	Pass	None

4.4 Test Case - Manage Employees: View Employee Information

Step No.	Data Input	Expected Behavior	Observed Behavior *					
			Run 1		Run 2		Run 3	
			(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L
Prerequisites: At least one employee must exist in the database, created by a managing user.								
1.	Navigate to Employees Page	Employees Page displays	Pass	None	Pass	None	Pass	None
2.	Click on existing	Employee's Information	Pass	None	Pass	None	Pass	None

	Employee	Page displays						

4.5 Test Case - Manage Employees: Update Employee

Step No.	Data Input	Expected Behavior	Observed Behavior *					
			Run 1		Run 2		Run 3	
			(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L
Prerequisites: At least one employee must exist in the database, created by a managing user.								
1.	Navigate to Employees Page	Employees Page displays	Pass	None	Pass	None	Pass	None
2.	Click on existing Employee	Employee's Information Page displays	Pass	None	Pass	None	Pass	None
3.	Click "Edit" button	Editable regions show	Pass	None	Pass	None	Pass	None
4.	Perform necessary edits into inputs	User edits show in inputted fields	Pass	None	Pass	None	Pass	None
5.	Click "Save" button	Editable regions disappear, the User edits have been saved successfully	Pass	None	Pass	None	Pass	None

Step No.	Data Input	Expected Behavior	Observed Behavior *					
			Run 1		Run 2		Run 3	
			(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L
Prerequisites: At least one employee must exist in the database, created by a managing user.								
1.	Navigate to Employees Page	Employees Page displays	Pass	None	Pass	None	Pass	None
2.	Click on existing Employee	Employee's Information Page displays	Pass	None	Pass	None	Pass	None
3.	Click "Edit" button	Editable regions show	Pass	None	Pass	None	Pass	None
4.	Perform necessary edits into inputs (leave a required field blank)	User edits show in inputted fields	Pass	None	Pass	None	Pass	None

5.	Click “Save” button	A notification displays stating that the required inputs are blank	Pass	None	Pass	None	Pass	None

4.6 Test Case - Manage Employees: Delete Employee

Step No.	Data Input	Expected Behavior	Observed Behavior *					
			Run 1		Run 2		Run 3	
			(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L
Prerequisites: At least one employee must exist in the database, created by a managing user.								
1.	Navigate to Employees Page	Employees Page displays	Pass	None	Pass	None	Pass	None
2.	Click on existing Employee	Employee’s Information Page displays	Pass	None	Pass	None	Pass	None
3.	Click “Delete” button	A prompt appears asking “Are you sure you would like to delete this employee?” with a “Cancel” and “Delete” button	Pass	None	Pass	None	Pass	None
4	Click “Delete” Button	The user is redirected to the Employees page, the employee has been deleted, and no longer appears	Pass	None	Pass	None	Pass	None
Step No.	**Data Input**	**Expected Behavior**	**Observed Behavior ***					
			Run 1		**Run 2**		**Run 3**	
			(Pass / Fail)	**H/M/L**	**(Pass / Fail)**	**H/M/L**	**(Pass / Fail)**	**H/M/L**
Prerequisites: At least one employee must exist in the database, created by a managing user.								
1.	Navigate to Employees Page	Employees Page displays	Pass	None	Pass	None	Pass	None
2.	Click on existing Employee	Employee’s Information Page displays	Pass	None	Pass	None	Pass	None
3.	Click “Delete”	A prompt appears asking “Are you	Pass	None	Pass	None	Pass	None

	button	sure you would like to delete this employee?" with a "Cancel" and "Delete" button						
4	Click "Cancel" Button	The user is redirected to the Employee's Information Page. No changes have occurred.	Pass	None	Pass	None	Pass	None

4.7 Test Case - Edit User Settings

Step No.	Data Input	Expected Behavior	Observed Behavior *					
			Run 1		Run 2		Run 3	
			(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L
1.	Navigate to "User Settings" Page	User Setting Page displays	Pass	None	Pass	None	Pass	None
2.	Click "Edit" button	Editable regions show	Pass	None	Pass	None	Pass	None
3.	Perform necessary edits into inputs	User edits show in inputted fields	Pass	None	Pass	None	Pass	None
4.	Click "Save" button	Editable regions disappear, the User edits have been saved successfully	Pass	None	Pass	None	Pass	None
Step No.	**Data Input**	**Expected Behavior**	**Observed Behavior ***					
			Run 1		**Run 2**		**Run 3**	
			(Pass / Fail)	**H/M/L**	**(Pass / Fail)**	**H/M/L**	**(Pass / Fail)**	**H/M/L**
1.	Navigate to "User Settings" Page	User Setting Page displays	Pass	None	Pass	None	Pass	None
2.	Click "Edit" button	Editable regions show	Pass	None	Pass	None	Pass	None
3.	Perform necessary edits into	User edits show in inputted fields	Pass	None	Pass	None	Pass	None

	inputs (leave a required field blank)							
4.	Click “Save” button	A notification displays stating that the required inputs are blank	Pass	None	Pass	None	Pass	None

4.8 Test Case - Manage Transactions: Create New Transaction

Step No.	Data Input	Expected Behavior	Observed Behavior *					
			Run 1		Run 2		Run 3	
			(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L
1.								
2.								
3.								
Step No.	**Data Input**	**Expected Behavior**	**Observed Behavior ***					
			Run 1		**Run 2**		**Run 3**	
			(Pass / Fail)	**H/M/L**	**(Pass / Fail)**	**H/M/L**	**(Pass / Fail)**	**H/M/L**
1.								
2.								
3.								

4.9 Test Case - Manage Transactions: View All Transactions

Step No.	Data Input	Expected Behavior	Observed Behavior *					
			Run 1		Run 2		Run 3	
			(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L
1.								
2.								
3.								
Step No.	**Data Input**	**Expected Behavior**	**Observed Behavior ***					
			Run 1		**Run 2**		**Run 3**	
			(Pass / Fail)	**H/M/L**	**(Pass / Fail)**	**H/M/L**	**(Pass / Fail)**	**H/M/L**

1.								
2.								
3.								

4.10 Test Case - Manage Transactions: Update Transaction

Step No.	Data Input	Expected Behavior	Observed Behavior *					
			Run 1		Run 2		Run 3	
			(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L
1.								
2.								
3.								
Step No.	**Data Input**	**Expected Behavior**	**Observed Behavior ***					
			Run 1		**Run 2**		**Run 3**	
			(Pass / Fail)	**H/M/L**	**(Pass / Fail)**	**H/M/L**	**(Pass / Fail)**	**H/M/L**
1.								
2.								
3.								

4.11 Test Case - Manage Transactions: Delete Transaction

Step No.	Data Input	Expected Behavior	Observed Behavior *					
			Run 1		Run 2		Run 3	
			(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L	(Pass / Fail)	H/M/L
1.								
2.								
3.								
Step No.	**Data Input**	**Expected Behavior**	**Observed Behavior ***					
			Run 1		**Run 2**		**Run 3**	
			(Pass / Fail)	**H/M/L**	**(Pass / Fail)**	**H/M/L**	**(Pass / Fail)**	**H/M/L**
1.								
2.								
3.								

5.0 Sign-off

Signature	Name & Title	Date	Comments
	Sheehan Toufiq	16-11-2014	
	Justin Patel	16-11-2014	
	Benjamin Byrd	16-11-2014	
	Andrew Rutherford	16-11-2014	
	Suraj Sequeira	16-11-2014	

6.0 Appendices

6.1 References

- Document Store:
  /directory/document-store/

- Application URL:
  http://sheehantoufiq.com/payday

- Code Repository:
  https://github.com/bebyrd/payday

6.2 Supplementary Attachments or Samples

Attached Items:

- Product Blurb:
  Location:
  /directory/document-store/

- Requirements Traceability Matrix
  Location
  /directory/document-store/

- Use Case Presentation and Screenshots
  Location:
  /directory/document-store/

- Individual Use Cases
  Location:
  /directory/document-store/

- User Interaction Diagram
  Location:
  /directory/document-store/

- Initial Mockups
  Location:
  /directory/document-store/

7. Software System operation
(Lack of errors, system crashes, ease of use, readability of user manual, correctness and completeness of user manual, etc.)

8. Quality of presentation
(Organization, pertinence, clarity and understandability of oral presentation, preparation and use of visual aids, effectiveness of demonstration*, etc.)

9. Implementation faithfulness to design; programming style
(See the Peer Review handout for an example of how these points may be graded.)

Signed by Team Coordinator on Behalf of the Project Team _____
acronym

* Presentations are planned for a multimedia room with a stand alone PC; may have to use a portable with a projection system; may not have any computer for demonstrations. Be prepared!

Task	Task Description	Team Member	Start	End
Task 1	Requirements Analysis	All	10.Sep	16.Sep
Task 2	RTM	Ben,Andrew	15.Sep	16.Sep
Task 3	Task Allocation Chart	Andrew	15.Sep	17.Sep
Task 4	Explanation of Terms	Suraj, Justin	15.Sep	17.Sep
Task 5	PowerPoint	All	17.Sep	19.Sep
Task 6	Rationale	All	15.Sep	15.Sep
Task 7	Appendix	Sheehan	10.Sep	20.Sep

Task	Task Description	Team Member	Start	End
Task 1	Requirements Analysis	All	10.Sep	16.Sep
Task 2	RTM	Ben,Andrew	15.Sep	16.Sep
Task 3	Task Allocation Chart	Andrew	15.Sep	17.Sep
Task 4	Explanation of Terms	Sujai, Justin	15.Sep	17.Sep
Task 5	PowerPoint	All	17.Sep	19.Sep
Task 6	Rationale	All	15.Sep	15.Sep
Task 7	Appendix	Sheehan	10.Sep	20.Sep
Task 8	Prototype	Sheehan	22.Sep	22.Sep
Task 9	Database Used	Andrew	01.Okt	01.Okt
Task 10	Rationale for use cases	All	01.Okt	06.Okt
Task 11	Use Cases	All	02.Okt	06.Okt
Task 12	Sequence Diagrams	All	04.Okt	06.Okt
Task 13	Object Rationale	All	11.Okt	15.Okt
Task 14	Category Interaction Diagra	All	18.Okt	22.Okt
Task 15	Function Point Analysis	Ben	21.Okt	22.Okt
Task 16	Software Architechture	Sheehan	21.Okt	22.Okt
Task 17	Object Design	All	18.Okt	22.Okt

Terminology:

Employee Management: The Admin(Manager/Owner) has access within the database to alter/edit the users(employees) that work under he/she; Alter/editing includes monitoring the employees pay stubs, and clock in/out times.

Dashboard: A dashboard within the Payday program is an altered external view of what two different types of users would see. One type of user would be the Employee. The other type is the Admin(Manager/Owner). Upon login, if it is an Employee log in, he/she is lead to a new screen that displays options for Clock In/Out time, View Net Pay, View Employee History, and User Settings(Settings for Users to manage the layout within their personal dashboard). Upon Admin login, the dashboard displays Various other setting that allow complete access to everything in the database. The functions for the Admin entail: Input Daily Revenue, Balance Sheet, Input Weekly Expenses, Cash Flow Statements, Add an Employee, Income Statement, View Eomployee History, View Taxes, Prepare Paycheck, View Hourly Pay, Daily Hours to Date, and User Settings.

Input Daily Revenue: This function is strictly for the Admin access and is a way for the Admin to enter in various revenue/profit earned in normal business hours in one day.

*(Change to income statement)Balance Sheet: This function is to gather all the financial income/losses from the difference between the revenue earned and the expenses accumulated quarterly onto a sheet for archiving.

Input Weekly Expenses: This function is used by Admin to archive to total amount of monetary expenses accumlulated in a week. The expenses would include input costs for products, taxes, and employee pay.

Cash Flow Statements: This function allows the Admin to see the amount of cash that has been accumulated quarterly through the business(Excludes Credit/Debit Charges). This relies heavily on the Balance Sheet.

Add an Employee: This function is basically a easy way of adding a new employee and there information to the system.

Income Statement: This function is used to get the company's total income over a quarterly period using the revenues and expenses accumulated during that time.

View employee history – Function that allows the manager to view employees clock in and clock out activity. It also allows the manager view the payment history o the employees.

Prepare paycheck – Function that allows managers to view information that allows and calculates the correct amount of payment to give to the employee.

View hourly pay – Function that allows the manager to view the hourly pay of every employee under him.

Daily hours to date – Function that allows the manager to view and keep track of the daily hours of the employees.

Employee View- Function that allows the manager to view different aspect and attributes of the employees such as employee history, preparing paycheck, view hourly pay, and view employees' daily hours to date.

New transaction – Function that allows manager to conduct new deals with the client that has the client give the payment and the manager uses the payment to payoff his expenses.

User Settings – Functions that allow the managers to view and edit their settings to fit their preference.

Logout – Allows the manager to exit and disconnect from Payday

Financial/Accounting – Function that allows managers to keep track of all the accounting and financial services that their company deals with. This function will allow managers to view current financial/ Accounting contracts and view their transaction overview.

Transaction Overview – This function allows the managers to view current our old transaction deals that they have already paid.

View taxes - This function allows the manager to view the taxes that will affect the paychecks of each employee. This function will also show the managers what amount needs to be set aside for taxes.