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.