Jigsa Haile Duguma

Metprotracker

Ruby on Rails Application for Project Timing and Management

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

10 April 2015

Metropolia

Author(s) Title Number of Pages Date	Jigsa Haile Duguma Metprotracker: RoR Application for Project Timing and Management 62 pages + 12 appendices 10 April 2015
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Aarne Klemetti, Senior Lecturer

The project was carried out to build a demo application for project timing and management using Ruby on Rails framework. Its main goal was to provide a system that could be used to facilitate live collaboration for group users and to provide a simple registration medium for single users to increase efficiency in project time management. It tried to solve the problem students have in keeping track of the time they spend on projects or assignments in general, and the problem they have in thesis project planning and scheduling in particular. It also intended to bridge the gap created because thesis advisors do not have an ideal medium to track their students' live activity while students are working on their thesis projects. In addition, it planned to include the needs of freelancers by including monetary fields and calculations. The project planned to show how a simple application could improve efficiency and productivity. It also intended to provide a medium to do projects in groups. In addition, business plans for the application were also discussed in the paper.

As a result, Metprotracker was developed in order to give users a mechanism under which they can register and analyze their performance data. The project uses Ruby on Rails framework for development; hence, it follows Model-View-Controller approach. Additionally, CSS together with JavaScript is used to style and present the data. The project provides a mechanism for thesis advisors to track the progresses of their students live and give the necessary encouragement or advice. It records data about the project prior, while and after it is carried out and provides a practical statistical analysis and presentation that can help users improve their efficiency. A collaboration option is also included in its features to enable users cooperate and contribute in group works.

Metprotracker shows how a complete version with additional features can be beneficial for users who do not collect their performance records that, as a result, might have been inefficient in their activities. It will also significantly contribute to a better communication and coordination between teachers and students, students and students, and freelancers and clients.

Keywords	

Contents

List of Abbreviations

DRY	Don’t Repeat Yourself - is a principle of software development which states that "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." [5]
ISV	Independent Software Vendor – traditional software where one-time licencing fee is paid for the software acquisition.
OOP	Object-Oriented Programming. Programming paradigm based on the concepts of Objects.
PMI	Project Management Institute - is a not-for-profit professional organization for project management. [1]
RDBMS	Relational Database management system. Software for maintaining, querying and updating data and metadata in a relational database.
RoR	Ruby on Rails - is a framework for development, production, and testing of web applications.
SaaS	Software as a Service – is a delivery model that allows users to operate a software , and vendors to provide it over the internet. [3]
SOW	Statement of Work – is a formal specific statement regarding the requirements needed in a project outlining and expressing work activities, deliverables and timeline of a contract or service that a vendor must perform for a client. [2]

1 Introduction

This paper is written to describe and document a demo application called Metprotracker developed to give users suitable features to time and manage their projects. The application can generally be used by any users who intend to perform a given task in an estimated duration of time collectively or individually for a purpose of their choice. The main target users that might benefit the most, however, are students, freelancers, and teachers. These target groups could use the application to track and time their assignments, homework, projects or other tasks. Students often delay and postpone their assignments and a simple application to guide them to disciplined time keeping and planning could benefit them greatly. When working on a thesis, a better medium that both the teacher and the student can access and update each other on its progress would be valuable.

In many cases, one can lose time-orientation while indulging deeply on a project or assignment and would lose track of the time spent on it. Having registers and statistics of that will make one look back and assess oneself to either make a better time allocation for certain tasks or know where to improve to best fit a given time allocation. This will lead to a path to continuous improvement by using performance records that would have been otherwise unconsidered and unregistered. This project aims to provide a demo application to serve this purpose and demonstrate, as a result, the feasibility and productivity of having an advanced version of it.

This paper uses conventional project management approaches to theoretically analyze the topic and uses the Ruby on Rails framework to develop the application. To grasp the whole architecture and implementation of the application a basic Ruby on Rails knowledge and an Object Oriented Programing understanding will help. The paper will try to cover the basic concepts and technical procedures behind the project and provide certain details when needed but will not document all the detailed processes of developing the application. It is also developed with a mind-set that incorporates commercial possibilities and thus business prospects will be included.

2 Project and Project Management

To understand the concept of this paper, defining and analysing the principles of projects will be essential. This process will include describing project management and applications for project management as well. The meaning of the word ‘project’ in its formal use might seem direct and less ambiguous considering it is a frequently used word that the readers are familiar with. However, the term is sometimes used to refer to activities that should not be considered as projects. On the other hand, some activities that are not seen as projects might fulfil its criteria; thus they can be considered as projects. As a result, the meaning of a project and what a project should include and exclude to be called a project will be discussed. It should be noted that the principles, complexities, and theoretical background of projects are more advanced than the depth it will be described in this paper. Nevertheless, attention will be given to include the relevant concepts that will to be sufficient for the scope of Metprotracker.

2.1 Defining a Project

Individuals or organizations perform tasks. Tasks generally can be of a project type or they can be of an operation type. These two have unique features that separate them but at the same time share several features that overlap. The overlaps and therefore the similarities these two possess sometimes leads to confusion as to which is which. However, when we clearly separate them based on their unique characteristics, what seemed to be a grey line dividing them becomes more visible.

Among the characteristics projects and operations share are that they

- Take time to be performed,
- Are performed by individuals or groups of people,
- Use limited resources,
- Involve planning, implementation and can be controlled. [1]

However, on the other hand there are certain characteristics that differentiate projects from operations. One general type of difference that can be vague depending on the practical circumstances and the theoretical approach is the fact that projects are mainly

implemented to achieve an individual's or a company's strategic goals. On the contrary, operations have a more tactical objective. The other properties that can more clearly separate these two, and thus the favourable approach, are that projects are temporary, while operations are continuous and sometimes very routine. Hence, we can define a project based on these two typical characteristics in the following way:

> A project is a temporary endeavour undertaken to create a unique product or service. [1, 4]

The fact that a project is temporary implies that there is a distinct and bounding beginning and conclusion, and its management designs activities so as to aid this property. Thus this is an integral property of a project and one of the main measuring scales to assess the success of a project. Simply put, how a project performs against its schedule and whether it punctually starts and ends is one of the main measurement methods for its success.

The second characteristic of a project is its progressive elaboration. Projects are designed and planned ahead but their implementation and delivery takes them through a process that might be very complicated involving many participants and several interrelated processes. Progressive elaboration is the coming into focus of abstractions and the revelation of details through time. It is always a good norm to be designing projects with a clear sight and to be anticipating the challenges it will face but there are almost always new challenges that will come into being. During these times participants adjust to get over the problems further getting experience that will help them clarify the abstractions. As a result, participants get clearer picture and the true scope of the project materializes through the practical implementations. It will keep on being elaborated until the final product is delivered. [1]

It was discussed above what makes a project different from an operation to have a clearer understanding of what a project is. What is worthwhile to add and discuss briefly is that a project is often confused with another category – a program. What makes a program different from a project is the clearer boundaries, objectives and results a project has as opposed to a program. A program is usually a collection of projects that are done in different times, taking unequal durations and covering a range of activities. This makes a program more abstract than a project and the more distinct elements of a project get blurrier on a program. A project has

- A more tangible outcome and objective than a program,
- More clearly defined problems and solutions,
- Strictly limited scope and duration,
- Relatively short term.

In general, a program is bigger with a more abstract and unquantifiable results, routes, elements and stakeholders than a project.

2.2 Project Management

The definition of project management takes different forms depending on the approaches used to view a project. A broad definition for project management can be as follows:

> Project management is the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements. [1,6]

This definition of project management came as a result of an intense discussion in the Project Management Institute (PMI) in the 1980s. In this discussion there was a debate on what to include in the definition and description of project management. One perspective was to look at it from written project requirements and specifications side and measuring its success by whether the project was able to do this or not. This is a more formal and perhaps very technical way to look at it while interactions between entities in the real world might diverge from being so.

The other side of the view was from the client's perspective. While a project's requirements and specifications are produced by taking the discussions, requests, and approvals of the client into consideration, a client's continuous involvement in the process of project execution will, more probably than not, produce a satisfied client.

Let us take a simple example. Suppose we want to have a chair made by a carpenter. One way to go about it could be we give the carpenter a specification of what kind of chair we want - say four legged, strong enough for 100kg, a certain colour, when we want to expect to get it etc. - and wait for it to be prepared and get it. In this process the carpenter will have the freedom of how and when to work on the chair as long as the requirements are met and will consider the project successful as long as the prior

requirements are met. In this case it is entirely possible that the carpenter could make a chair that perfectly satisfies the specifications and considers the project a success. However, that might not satisfy the client's expectations and the client might differ in opinion in the degree of the success.

The other way could be involving the client in the process of making it by giving progressive assessment and improvement means for the client in the hope of getting a maximum customer satisfaction. So we can adjust the requirements – say change the colour of the chair in the above example – through the process. This is a more client-centred approach thus leading us to a second possible definition to incorporate this fact.

> "A project management is the application of knowledge, skills, tools, and techniques to meet or possibly exceed the expectations of the client." [2]

2.3 Computer/Web Applications for Project Management

As mentioned in section 2.1 and section 2.2 above, project and project management concepts can stretch in size and detail depending on the task at hand. Projects can vary from a multi-billion euro ones employing thousands of participants to a simple school project where budgeting is minimal to zero done by an individual. The underlying fact that should not be forgotten, however, is that all projects share common grounds in that they intend to do a purposeful task with in a time limit consuming a limited resource. On the other hand, their vastness and importance will differentiate them, at least in what kind of management they should use if not anywhere else. Thus an all-purpose project management tool or application that can serve the needs of all kinds of projects irrespective of their detail, size and importance will be impractical.

Projects existed before computers or the internet and there were tools and mechanisms to manage them, albeit not as efficiently as today. In today's world where technology is more or less at the tip of our fingers, people have become increasingly reliant on computers and web technologies to a point where imagining and knowing how tasks used to be performed before them is challenging. Indeed, while looking forward to more improvement in our technologies, we sometimes tend to forget how far we have come from and become appreciative of the tools we have now. In light of this, it is believed that including into this paper a brief flashback into how projects were managed will

shed light on the degree of progress in the topic and with any luck might even inspire people to get an idea of how to further improve the mechanisms we have now. In the very least, it might inspire some to read and learn more about our lives before computer technologies in general, and to appreciate the tools we have for our projects now and use them well to effectively utilize our time and resources in particular to this topic.

History of project management before computers. A statement of work (SOW) is a formal specific statement regarding the requirements needed in a project outlining and expressing work activities, deliverables and timeline of a contract or service that a vendor must perform for a client. An SOW was carried out by procurement before computer technology existed and contract administration was considered as project management while it actually is project monitoring and it was considered as a part-time occupation that detracts employees from regular work. Good typists were highly needed for SOW preparations and mistakes were corrected with a white-out. Without computerised databases, as projects progressed, lessons learned and best practices were often memorized in minds of individuals and sharing them to all that might need them was far from reach. Individuals with good experience in SOW will be promoted often and all their experience will be practically lost as new people replace them and the bulky documentations on their practices, if any, will not be effective teaching tools and of little practical use. [1]

When contractors found holes on the SOWs that gave a room for scope changes that were highly profitable, they used to keep this a secret from the clients until the contract is awarded giving them a bigger profit from scope adjustments that they will eventually do later. Scheduling was even more challenging with the information to update the schedules coming from the weekly team meetings and bar chart updates to the client on the progress of the project were made on a monthly basis. In the monthly meetings the client would assume the data is correct and the progress truthful and proving this was expensive and difficult with limited ability to show supporting data without computers. Therefore, vendors were very reluctant to update schedules because it was a time and money consuming, manual process and some companies even had policies that they should only be made if, and only if, the path of the project changed.

A large portion of the budget was used for preparation of reports and companies often had a big Graphic Arts Departments that had dozens of employees whose sole purpose was preparing and designing reports and graphs. The element of communication

was probably the biggest setback in project management in the past as communication between employees and departments was poor as there was no option for one individual or department to easily track the progress of the other or even itself. For this reason, working from home for project managers or team members was not possible as it is nowadays. [2]

Current Computer/Web Applications for Project Management. After the Internet revolution of recent times, development of applications for project management improved and diversified, as is the case with other disciplines such as marketing, commerce, medicine etc. As all disciplines realized the potential advantages they will get from going virtual, a new big market was created for the development of such virtual services that improved and added new features that were not possible before. Computer engineers and developers were therefore in high demand for the creation of more powerful and improved virtual services and applications to fill up this void. However, this vast demand enticed individuals and investors to flock into the market creating a fierce competition.

There are currently a number of computer/web applications for project management. It cannot be said that all have equal market penetration, user-base, income or profitability. However, it can be seen from how much growth and popularity these companies get that a new and expandable market share can be obtained with intelligent designing and marketing strategies even when started from scratch.

The one advantage the IT sector gives is an opportunity for individuals with ideas, original or not, to implement them without necessarily needing a huge investment. This is a significant advantage for start-ups to start and flourish. Successes of companies like Facebook that started from small powerful ideas, modest capital and investment even though their idea is not entirely original are good examples. Without taking anything from the power of having an original idea, the power of changing styles and improvement of existing ideas should be underlined as well. The fact that MySpace started as a social networking media before Facebook is a good example. But improvements on features and effective management made Facebook surpass it and all the other social medias in all measurements and this is a good example of how changing styles pay off.

To answer the question ‘why another project management application if there are already a few?’ and address the need of this paper, a few perspectives will be proposed

below. The fact that there are already virtual applications for project management should not be the sole reason for pessimism since doing it first might influence, but not block, its success. As proven time and again in the technology world, adjusting and improving existing styles rewards. Besides, there will almost always be room for improvement and an option of targeting a neglected market. The market for project management applications is huge, perhaps too huge for one or a couple of companies or applications. Projects differ greatly in size, type, scope, complication, importance, budget, location etc. Hence, the applications they will need vary accordingly leaving small room for chance of monopoly by one or a few applications. What serves one company or project, what serves one sector of activity or discipline, will not serve another equally well. This resulted in the abundance of successful virtual project management applications, with more continuously coming and taking a market share.

The main target for existing computer/web applications are well paying customers which are usually companies employing a number of employees and involved in bigger projects. This drives up the degree of complication the application needs to have to address the needs of its target customers. When simplicity is compromised, perhaps reasonably in this case, an increasing a low to non-paying customer base gets neglected giving rise to the need to address their needs. One such customer base could be students who are of low to non-paying customer types who would not need complicated applications for the management of their projects. The compromise of simplicity also happens for aesthetics and although visually attracting users is an important factor that should be considered, the perfect condition will be attained when a balance between the two is reached. For the above reasons and others, simplicity will be a carefully managed aspect in this application.

Software as a Service (SaaS) / Web application - Before going into further detail, describing the idea of SaaS and some features and architectures will be vital for the overall understanding of the application. Furthermore, the differences and advantages of a web application (SaaS) and a desktop application (ISV - Independent Software Vendor) will be discussed which will justify the reason for choosing a web application for this specific project.

The main reason this application should be considered as Software as a Service (SaaS) application rather than an Application Service Provider application is in its multitenancy. SaaS is an idea that originated from the Application Service Provider applica-

tions of the 1990s which provided wrapped applications to business users over the Internet. These early attempts before the explosion of the Internet of trying to deliver software over the Internet had features that resemble a traditional on-premise application than the modern day SaaS applications. They had a very limited ability to share and process data with other applications because they were a single-tenant applications.

Traditional software or the locally installed software we are familiar with is a software that is licensed by a customer for its installation on a local hardware within a local system. One such software would be Microsoft Word where we install it on our local machine by buying its licenced version. Most software before the era of the internet are included in this category but even today, many business software still are of this type where they are installed and maintained by employees locally. SaaS on the other hand is a delivery model for applications that allows users to operate a software, and vendors to provide it over the Internet. Hence, SaaS architecture is designed to make the software provide a set of services as well as a vehicle for its delivery. SaaS revenues are usually subscription based, where users usually pay a flat recurring payment over a duration which can be annually, biannually, quarterly, or monthly. On the other hand, traditional software (ISV) pay a one-time licencing fee for the software acquisition that may not be extended further.

Today's SaaS applications take advantage of centralization through a single-instance, multi-tenant architecture in providing a multi-featured experience that is competitive with analogous on-premise applications. SaaS application is presented either directly by the vendor or else by an intermediary party known as an aggregator, which packages SaaS offerings from various vendors and presents them as part of a unified application platform. This arrangement where service and data are centrally processed gives a great deal of advantage and a vast opportunity to save money and redeploy scarce resources while also generating a threat to the reliability of the integrity and safety of data. Hence, the choice between SaaS and on-premise software is significantly dependent on the sensitivity to the integrity of the data and application the service should provide. [3]

Developing an application for a customer will always demand a great consideration for the need of the customer and understanding how customers make a choice on what to use is important. At the same time the developer should also consider facts that will

make the development attainable and feasible. Deciding whether to use SaaS or not involves taking into account several considerations most of which ultimately converge to the decision to compromise between our cost and control. Among the many considerations that can be taken for both customers and vendors, some of them are:

Technical considerations: SaaS applications usually offer flexibility for customer configuration, but there will be limitations. If an application demands particular technical knowledge for the control and support of it, or if it requires certain customization that a SaaS vendor cannot provide, a SaaS solution will not be feasible.

***Data size considerations*:** Another important factor to consider is the size and type of data that will flow from and to the application regularly. The difference between gigabit ethernet connection links in local LANs and Internet bandwidth is usually significant. Transmissions that might take a few minutes when transferred in LAN might take significant time when done over the internet on applications located across a country. Therefore, network latency considerations should also be made in the decision.

Financial considerations: The initial cost of ownership of a SaaS application will be lower compared to that of the same on-premise application. However, the long term cost of using SaaS will continue to increase from the subscription payments and the proper financial mechanism of comparison between the two should be made since their payment methods are not similar to compare directly. The cost of obtaining SaaS is affected by the degree of customization and custom configuration needed to integrate it with available infrastructure, the number of users, and amount of data among other things.

Legal considerations: Some industries might be subject to regulatory law which demands record keeping and reporting requirements. Industries might also have higher standards for privacy and data security which, if not met, will have legal ramifications. Therefore, the regulatory environments and legal obligations should carefully be considered in all jurisdictions the application might be licenced to.

Political considerations: The internal politics of organizations greatly affect the way a company decides. If important people in a company have reservations on exporting data out of the premise or adopting a SaaS system, other factors that might even justify adopting one will be unimportant. Managers and CEOs usually approve final decisions

and they might occasionally dislike, in theory, the idea of keeping data elsewhere without having a proper in-depth understanding of the advantages and risks of doing so. [3]

The general introduction of SaaS written above was intended to familiarize the reader with the concept of SaaS and provide a base for upcoming concepts such as multi-tenant data structure which will further elaborate the concept of SaaS.

3 Background to Technologies Used

The main methods of development and the technologies used to develop this application will be introduced and described below. A background in information technology might be needed to fully grasp the details and mechanisms of the development of this application. However, simplicity in describing the methods will be significantly considered whenever possible so as to make it an understandable documentation to readers that might be new to the rails framework.

3.1 Rails

Rails is a web application development framework written in the Ruby language. In late 2003, David Heinemeier Hansson was trying to create a web-based project management solution using PHP but was frustrated facing the common weaknesses of the language which is having to repeat the same code in multiple places. This process was time-consuming, redundant, and error-prone. Rather than using PHP, he used Ruby to develop the application in two months using object oriented programming to clear redundancy. Through the process, Hansson realized the code he used could be extracted into a framework that can be reused for other applications. In 2004, he released his framework which came to be called Rails. [4]

Rails is designed by making assumptions about what developers will need to get started making development of web applications easier. It is a highly efficient framework providing a way to write less code and get more results making web application development interesting and enjoyable. Rails is an opinionated software. This means that it encourages ways it considers are the best ways to do things and sometimes discourages other alternatives. For this reason, ‘The Rails way’ of doing things make a development process easier.

Rails philosophy is guided by two major principles:

DRY - Don't Repeat Yourself. Is a principle of software development which states that "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." Simply put, duplication of information or code should be avoided

and the same changes in different places should not be necessary making the code more extensible and maintainable and less buggy.

Convention Over Configuration. Rails does not possess many configuration options like other frameworks but by so doing have simplified the development environment by avoiding the need to go through many configuration files. It is sometimes called 'selfish' because it has its own assumptions of the best ways of doing things and defaults to this convention. [5]

3.2 Ruby

Before getting any further into the history and concepts of the Ruby programming language, defining Ruby will be helpful.

> Ruby is a dynamic programming language with a complex but expressive grammar and a core class library with a rich and powerful API. [6, 2]

Ruby was developed by Yukihiro Matsumoto, commonly known as Matz, in 1993 and was first released in 1995. It has features that relate it with other object-oriented programming (OOP) languages like Python and Perl but the degree of object orientation is more than what can be found in them. Due to this reason, it can be said that in some respects it has more common features with pure OOP languages like Smalltalk than Phyton and Perl. Even though Ruby took after these programming languages, it uses easier convention and grammar making it even easier for programmers that already know other programing languages like C and Java to learn. Ruby can be suitable for procedural and functional programming styles. Ruby is also interpreted programming language which means that it is read line by line instead of going through a compilation process to get an executable that is human unreadable.

Matsumoto developed Ruby principally to ease the job of developers using a principle of least surprise, meaning that programming language should behave more or less as the programmer expects it to. Methods are named after common English words that expressively suggest the action to be performed. For example, the actions delete, split, upcase will do actions their name implies on strings of text. In order to familiarize the reader with the Ruby language, important and interesting features of Ruby that might

differentiate it with the more common languages like Java or C will be discussed below with simple basic descriptions and examples when needed. [4, 5]

Object-Oriented - as was mentioned above, Ruby is a highly OOP language. The idea of object-oriented programming approach is that computer applications or programs can be built from a collection of units called objects that execute actions on one another to come up with the desired outcome. The earlier programming approach called procedural programming is different in that applications are written simply as a set or list of instructions to be performed procedurally for the computer to perform. There are three basic principles of programming approach that broadly encompass the difference OOP has with other programming paradigms. These are

- ***Encapsulation*** - refers to the ability of objects to hide details of data behind an abstracted interface leaving only a few ways for the outside world to manipulate it
- ***Inheritance*** - refers to the condition where classes lower down a hierarchy inherit features of those that are higher up in addition to having specific features of their own which they can pass on to descendants of them.
- ***Polymorphism*** – refers to the condition where behaviour of an object might vary depending on the input. [7]

Ruby is a pure OOP because every value is an object in Ruby. This includes numbers, Booleans and nil (null) that might not usually be objects in other OOP languages. We can invoke a method on them because these are considered objects.

Listing 1 – Method Invocation on Objects

```
6.class # => Fixnum: the number 1 is a Fixnum
7.9.class # => Float: floating-point numbers have class Float
true.class # => TrueClass: true is a the singleton instance of
                TrueClass
nil.class # => NilClass
```

As shown in listing 1, a method called class can be invoked on objects. Comments in Ruby start with **'#'** and the value returned by the code is indicated by '**=>**'. Parenthesis in function and method invocations is usually optional and can be omitted. As shown in the example codes, everything in Ruby is an object and can be operated upon with the right method.

Ruby Methods Methods in Ruby are defined with the keyword **`def`** and their end is indicated by the keyword **`end`**. This will encompass the body of the method rather than punctuations like in some programming languages and the value returned from the method will be the value reached after evaluation of the last expression in the body of the method.

Listing 2 – Method Definition in Ruby

```
class Math
      def summation(x, y)
            x+y
      end
end
```

The second line in Listing 2 defines a method called summation in the class Math which takes two parameters x and y. The third line returns the sum of x and y and the third and fourth lines signify the end of the method and class instance respectively. These kinds of methods are called instance/object methods. A method defined in a class by just giving it a method name will be an instance method operating directly on an object.

Other types of methods are class methods. Sometimes methods will be needed to work directly on the class itself rather than being available for object instances only.

Listing 3 – Class Methods in Ruby

```
class Math
      def Math.summation(x, y)
            x+y
           end
end
```

As shown in Listing 3, the method **`Math.summation`** is a class method working directly on the class Math. This can also be written as **`Self.summation`** and **`Self`** will represent the class the method will work on. The class method can be considered as a method asking the class to do something to the whole class than asking specific objects of the class. The class method will be defined by prefixing the name of the class to the method we want to define.

Ruby Variables Ruby has four types of variables. These are:

Local Variables These variables are used inside a particular Ruby method and they cannot be accessed outside of that method. Its scope is inside that method and it will die when that method ends.

Listing 4 – Local Variables in Ruby

```
def local_variable
      x = 5
      puts x
end
```

Listing 4 shows local variable x. A local variable and it cannot be accessed outside of the local_variable method.

***Global* Variables** These variables are almost the exact opposite of local variables and they can be accessed from anywhere within the application. Global variables can be needed sometimes but they are not encouraged in ruby because they will make debugging very difficult since they can be hard to locate. They are also discouraged because they will interfere with the basic idea of an object oriented approach where methods and variables are separated in blocks. Global variables are defined by putting a **$** sign infront of the variable name we want to define. Looking at Listing 4, adding a $ sign in front of **x** in line 2 will make **x** a global variable making **x** accessible from anywhere in the application.

Instance/Object Variables As in the other cases of variables, the scope of the variable defines which category a variable belongs to. An instance or object variables are associated to a current object and will be unique to each instance of a class we create. These are the most common types of variables when working in rails because we usually create instances of classes to work with. We define an instance variable by putting @ sign in front of the variable we want to create. If for example we have a class **`Car`** with the car color and car make defined on it, we can create an instance variable and manipulate those features. An instance variable will not be shared by the class's descendants. Instance variables are shown in listing 5.

Listing 5 – Instance Variables in Ruby

```
@car = Car.new  -> will give us a new object car
@car.color = 'Red' ->  will set the color attribute of the car
to Red
@car.make = 'Toyota'  --  will set the make of our object
Toyota
```

Class Variables This is a variable whose scope is within the current class and not in any specific object of a class. The main difference a class variable has with instance variable is that a class variable is shared among the class and all of its descendants too while an instance variable is not. Class variables are used to store information relevant of all the objects created in a certain class. Class variables are denoted by @@ symbol followed by the variable name.

Ruby Arrays and Hashes Both arrays and hashes are variables that we use to store more than one value. An array in Ruby references each value by a numeric id that starts with zero while a hash uses objects as a key to reference values. An array is easier to work with and hence is often used than a hash but there will be times where using a hash will be beneficial or necessary.

Listing 6 – Ruby Arrays

```
vehicle = ['car', 'truck', 'train']
puts vehicle[0]         # will return car
vehicle << 'bike'         # will add 'bike' to the array vehicle
at the end of the array
```

Listing 6 shows how an array is used in ruby. The first line shows how array is declared and the second line shows how to access the values stored in the array. Line three shows how a new value can be added into an existing array.

Listing 7 – Ruby Hashes

```
vehicle = {:car => 'road', :truck => 'road', :train => 'rails'}
puts vehicle[:car]        # will return road
vehicle[:bike] = 'sideroad'      # will add a hash value 'side-
road' with the key bike referencing it
```

Listing 7 shows a hash and on the first line a hash is created with the first terms like **:car** being the key pointing to a value for that key which is the string 'road' for :car. Hash keys must be unique for that particular array. The second line shows how hash values are accessed using their keys and the third line shows how a value is added to a hash by specifying a unique key :**bike**.

Ruby Decisions and Iterations Decisions are essential in programming and the two main decision structures are the **if** and the **while** conditions. Braces are not used in **if** and **while** statements and we signal the end of the block by the keyword **end**. **If** statement tests a condition and returns a result based on the condition and can be extended to **elseif** or **else** to test more conditions. **While** statement on the other hand will loop through commands until the conditions fail.

Iterations are done on integers objects by **times** and **upto** methods as shown below.

```
2.times { print "Hello! " }      # Prints "Hello! Hello!"
1.upto(5) {|x| print x }        # Prints "12345"
```

Iterations on arrays or hashes are done by the keyword **each** and the block will be delimited by **do** and **end**. The word **each** will iterate the associated block once for each element in the array or hash.

Listing 8 – Iterations on Arrays

```
vehicle = ['car', 'truck', 'train']
vehicle.each do |f|
      puts f
end
```

Listing 8 shows how iterations are performed on arrays in Ruby by using `each` method.

Listing 9 – Iterations on Hashes

```
vehicle = {:car => 'road', :truck => 'road', :train => 'rails'}
vehicle.each do |key,value|
      print "#{value}:#{key}, "
end
```

Listing 9 shows how hashes can be iterated using `each` method in Ruby. [8] [9]

3.3 Ruby on Rails

Ruby on Rails (RoR) is a framework for an easy web application development, production and testing. It is an open-source framework that is written in the Ruby programming language and it has grown in popularity significantly over the past few years. Ruby had been around for much longer than Rails (almost a decade) but had been relatively overlooked until the creation of Rails which brought it to attention. Ruby associated applications online (such as Twitter, Hulu, Bloomberg, and Scribd) use Rails in some form or another.

Rails is chosen for this project over other frameworks because of its simplicity and wide functionality. Rails gives a head start for developers in a straightforward manner by making assumptions on configurations and designs. These assumptions can be overridden but they are usually sensible paths that make applications development fast and easy. Understanding the philosophy behind the creation and concept of Ruby on Rails will perhaps be easier if the creator of Rails David Heinemeier Hansson's saying is quoted.

> Rails gained a lot of its focus and appeal because I didn't try to please people who didn't share my problems. Differentiating between production and development was a very real problem for me, so I solved it the best way I knew how. [10, 1]

It can be seen from Hansson's words that rails was created to break the common hurdles of complexities developers go through to start developing applications and provide a default ways to bypass them. It was a gamble that traded flexibility, needed by people who do not share similar problems, with simplicity, needed by people who do. Its success through time had indeed proven that simplicity was an important feature desired by many developers that had been relatively unaddressed before.

Rails applications have three standard preconfigured modes of operations called development, production, and test. These are execution environments possessing an accompanying collection of settings to tell the application how to do certain things such as choosing which database to connect to. The default environment is often the development environment but environments can be specified through the environment variable by naming the mode of operation. Creating a custom environment is also easily possible if the need arises. Some fundamental characteristics of rails are governed by

environment settings, for example class loadings, and understanding these settings could be vital.

Rails applications follow a **M**odel-**V**iew-**C**ontroller **(MVC)** architectural pattern. This divides the process of the application into three separate but closely related components communicating with each other. It is a core concept of Rails and will be discussed below.

Model - View – Controller - The MVC architecture separates the structure of rails into the Model – business logic, the View – the user interface, and the Controller - the controller logic. The Model integrates the business logic, the View manages the presentation logic, while the Controller manages the application flow. This approach brings several advantages to the developer because it separates the components of the application into isolated code making it easy to manage and debug. For example, views can be managed and edited without worrying much about their effects on the controller or model making life easier for the developer than it would have been if everything was clustered together. This, among other things, makes Rails more preferable than PHP or ASP. [10]

When a browser sends a request, the web server will pass it to the controller that controls what should be done. Controller might sometimes render a view immediately for static contents which is converted to html and sent to browser. For dynamic contents, the controller will interact with the Model (Active Record) and fetch necessary data to then send it to View giving out html content to browser. These three components of MVC model are all created by rails with every new application and the developer will modify them according to his/her needs. These three components will be described below.

Model (ActiveRecord) - The Model layer holds the business logic (database) of the application together with the logic to manipulate and retrieve the data. Models are represented as classes in rails and they can be thought of as abstracted interfaces connecting controller code and data while also handling association, validation, transactions, and so on. ActiveRecord library implements them providing the interface for the programming code to manipulate tables and their data in the relational database. When we generate a model in Rails, a database migration file is created in db directory. The-

se migrations are Ruby classes and makes creating and modifying database tables easy and neat by running simple rake commands.

View (ActionView) - The view is the display logic representing the front-end of the application. The controller's decision to present a data to the user will trigger the production of view which is composed of templates usually formed from a mixture of HTML and Ruby code. The embedded ruby code that is found in views is fairly simple usually including conditionals and iteration because controller should be handling most of the logic. Views can also be rendered as XML, RSS or other formats. An Embedded Ruby (ERB) built system called the ActionView library implements views by defining templates for the presentation of data. View files are usually saved as ERB format but lately a much preferable templating solution for many developers is turning out to be Haml because of its neatness and readability. Haml uses indentation to determine the hierarchy of an HTML document.

Controller (ActionController) - Controller is the heart of the MVC model that directs traffic coming through the dispatcher, querying the model for specific data on one hand, and organizing it by doing various tasks on the other hand to then fit the need of the view to display the desired outcome to users. When someone connects to Rails application what is actually happening is the application is being asked to execute an action in the controller. In Rails controllers hold methods known as actions, such as '**update'**, '**show'**, '**delete'**, that usually are self-descriptive in their naming and they are called based on what the user requests. An ActionController implements the controller acting as a dealer between the ActionView (display) and the ActiveRecord (the database). [8]

Controllers are in charge of the flow of the program and they might seem to interact with both the model and view equally as necessary. But in actuality controllers are more closely tied with views than with models. It is possible for two people to work on the controller layer and the model layer without having to meet each other. It is also possible to write the model of an application before starting to write the controller. But views are much more closely linked to controllers and the need for communication between the controller and the view comes very early into the development of the application. In addition, they share a great deal of information and the variable names we choose on the controller will have a significant effect on the view. [11]

RubyGems - Ruby programs and libraries are packaged into a system that is easy for users to install and maintain. This packaging system is called RubyGems. Different versions of gems can easily be managed and installing them can be done with a single line of command in the command prompt. Gems have names, descriptions, and version numbers. Gems that a Rails application needs will be listed in the gemfile located at the root of the application. The gems used in this project will be mentioned with their purposes when needed in the implementation part.

Every time gemfile is modified or dependencies are introduced, a bundle install should be run to resolve all dependencies and install gems needed. Bundler is the most favoured way to manage application's gem dependencies. Rails 4 uses Bundler automatically and it is not necessary to install bundler gem. Bundler resolves dependency on the list of gems that are found in our application making it easier and faster to start the application development process.

3.4 Postgresql

PostgreSQL is an open source object-relational database management system (ORDBMS) that arose from a project which started at University of California at Berkeley Computer Science Department. The implementation of POSTGRES dates back almost as early as the beginning of relational databases to 1986 with the first demoware implementation a year later. The first and second versions were released in 1989 and 1990 respectively to few external users but it was not until 1994, when an SQL language interpreter was added, that it was released to the web as an open source descendant of the original POSTGRES under a new name – Postgres95. A year later, the name was changed to Postgresql in order to indicate its SQL capability for one, but also because the 95 suffix in it would not serve well through time. To this day people continue to refer to PostgreSQL as 'Postgres' which is widely accepted as alias considering simplicity of the word and its history.

A large part of the SQL standard is supported in Postgresql and offers several modern features. Some of its modern features include foreign keys, updatable views, transactional integrity, multiversion concurrency control, complex queries and more. It also gives the possibility for the users to extend it by adding new aggregate functions, index methods, operators, procedural languages and functions. In addition to this, Postgresql

provides enterprise class features such as the possibility to create aggregate functions and utilize them in window constructs, SQL windowing functions, common table and recursive common table expressions, and streaming replication. Seldom are these features found in open source database platforms. Newer versions of proprietary databases such as Oracle and SQL Server have these features but the simplicity to extend it without changing the underlying base and the speed it performs them makes it preferable. [12]

PostgreSQL has liberal license and it can be used, modified, and distributed free of charge for any purpose whether it is commercial, private or academic. Rails have SQLite as a default database but for this application, postgresql is chosen for its unique advantages on different areas related to the project. Some deployment environments also support only postgresql.

3.5 Multi-tenancy in Web Applications

Before diving into the concept of the topic, describing the meaning of some terms that will be used hereafter might be helpful. A **Tenant** is a client served by an instance of a web application. As the name suggests, it can be thought analogously as people living in a house served by its space provided by the land lord. Using this analogy, a house can be standalone rented to a **Single-Tenant** (which is analogous to an application installed on a separate instance) or can be an **Apartment** serving multiple tenants sharing resources.

A Multi-Tenant application architecture is a concept in software development in which a single codebase running an instance of an application installed in a single location simultaneously serving multiple clients (tenants). It is considered to be a substantial paradigm change for architects accustomed to developing isolated single-tenant applications. This will require an architecture that will share resources across clients (tenants) while at the same time maintaining the isolation of data belonging to different customers. Using the house example given above, this will be similar to the landlord of the apartment using shared resources like electric or water lines while keeping apartments separate with walls good enough to ensure the privacy of the tenants. This example can further extend to explain many features of multi-tenancy concept in a fairly sensible manner if used reasonably for further concepts from now on.

Trust, or more precisely the lack of it, is the main reason that reduces the scope and market growth of applications that use a Multi-Tenancy concept of architecture. At the core of every business or organization lies data, which could arguably be one of the most important assets of a company while its sensitivity might differ based on several factors. Conversely, Multi-Tenant applications use centralized system with network-based access to data that will make clients give up control of their data while providing a more affordable and economical option to an application service. So the difficulty of a Multi-Tenant architecture will be the challenge of providing a robust and secure service to satisfy clients while at the same time being cost-effective and manageable. [13]

The need for sharing of data, or on the other hand the need for isolation of it, depends on several factors and distinction between them is not discretely divided; rather it is a continuum. The continuum can have three distinct approaches based on where the data architecture need might fall in the continuum. These three divisions for creating a Multi-Tenant application are divided based on three approaches that we can use to manage a Multi-Tenant data model. The three Multi-Tenant data management approaches are mentioned below.

Separate Databases – In this approach each tenant will have a logically separated data set of its own while sharing computing resources and application code. Metadata will be used to associate the tenant with its corresponding database. Restoring data in case of failure will be easier because of the isolations and security and integrity of data will be maximum. However, this approach has high hardware, maintenance and support costs while giving a premium solution to the problem of data insecurity. Clients in fields of banking or medical records management would prefer this kind of approach because of the sensitivity of their data.

Shared Database, Separate Schemas – In this approach multiple tenants will use the same database but each tenant will have its own sets of tables grouped into a schema specific for that tenant. A moderate level of logical data isolation can be attained with this approach giving some level of security. Hardware for database will be a shared resource and thus can support higher number of tenants per database server than the isolated system making it more economical with medium data security needs. Restoring data for a single tenant will be much harder and will affect the operation of the other tenants.

Shared Database, Shared Schema – The third approach uses the same database with the same set of tables having records of multiple tenants in them serving multiple tenants' data. This approach has the least security compared to the previous approaches theoretically. It also has the least hardware costs and can serve the highest number of tenants per database server. However, this approach might pick additional development effort and cost in area of security since the chance of data being accessed by the wrong tenant might increase. The restoration of data for a tenant in case of failure will add additional layer of complication since tenants share rows. This approach will be best for clients that are willing to trade some level of data security for minimal costs for their services. [14]

4 Implementation

Chapter 3 mainly dealt with the introduction to the background of concepts, technologies and methods used for the development of the application. The implementation part will deal with the development process of Metprotracker application, including the architectural design and concept of the application, the methods used for the development, the approaches selected and other important procedures.

Through the development of Metprotracker, a number of paths were taken in the process of solving problems and producing the necessary or, if possible, better outcomes. However, it will be difficult to document all that process through the scope of this paper. Therefore this part will concentrate on the successful routes taken rather than the unsuccessful ones abandoned or changed to keep the documentation simple. In contrast, details and theoretical concepts will sometimes be added in this part in light of the potential extension the application might undergo in the future where a number of features will be added and the application scaled up. The demo application takes certain degree of simplifications as well that the final production level application would not.

Before the start of developing Metprotracker, a simple questionnaire was prepared to get an objective idea of how useful such an application would be. The questionnaire was distributed to 20 pure students (with no formal jobs) and 20 former students currently working in the IT business hoping to shed a light on how much disparity there is in their behaviour on time management while working on projects. The results showed that 17 out of the 20 students hardly used any mechanism to track how much time it takes for them to complete a project or an assignment out of which only 4 thought they could estimate with a reasonable accuracy in hours a task might take them. On the other hand, 18 out of the 20 employees in the IT industry said they had to and have used some sort of mechanism to track time working on projects out of which 13 said they can estimate with a reasonable accuracy how much time a task will take them.

The data collected described above and the statistical analysis from it would not be sufficient to make an accurate inference on the level of difference between IT students and IT employees. To be able to do that, rigorous data collection and analysis, various behavioural, environmental, conditional considerations should be made, which was beyond the capacity of this project. Nevertheless, the data collected exhibited the natural assumption and notion that students can be less time-constrained and organized.

Hence, a mechanism for them simple enough to interest them to use it while providing a benefit of creating time awareness on their tasks will be advantageous. Such an application will make them know their potential, point their weaknesses to improve, show their strengths to capitalize on and more. It should be noted that even though the application was developed taking students' needs into considerations, other people can use it as well effectively. Teachers or school staff could use it to keep track of their students' progresses on assignments or projects.

The application will have several features in its production stage most of which will be included in the demo-version developed so far. Its specification changed through the course of development from feedbacks and will further change as well to make it more useful and functional for users. The standout features the application has are

- Application and data isolation based on subdomain path,
- Collaboration option to work on projects as a team,
- Authentication mechanisms for access,
- Secure invitation mechanisms for collaboration,
- Time estimation option for project or task intended,
- Actual time registration and measurement mechanisms for projects or tasks,
- Status updating and updating notifications,
- Graphical reporting mechanisms on performed projects,
- Table based reporting options which can be extended in the future to printable invoices,
- Simple statistical analysis and remarks on performances,

The standout candidates for using the application are

Students – to keep track of their performances time-wise on tasks, assignments, and projects performed individually then collect and analyse the data to know where their drawbacks lie and improve their efficiency in time management.

Teachers – to assign students on tasks, assignments or projects and monitor their progresses and assess the result for a better understanding of their needs (for example

it is a good communication medium for students and thesis advisors. Advisors can push students to be efficient by setting them a goal and checking their progress.)

Freelancers – to keep track of their time management on assigned projects and analyse the data to make better monetary calculations and make productive considerations for improvement on conditions or choose favourable projects. In addition, it can be used to provide a report for clients as well as a good communication medium.

School/School staff – to track the performance and efficiency of people they assign a given task to (could be placement working students) and use the data for necessary adjustments.

4.1 Starting Development in Rails

Rails (Ruby on Rails) provides a more advanced and pre-created framework for starting to develop database-backed web applications than others. Downloading and installing the rails framework and ruby can be done in various ways depending on the operating system we use. Most developers use Ubuntu and Mac and the support for them is more readily available as well but Windows is also recently catching up as more new Windows based developers start to use Rails . This application was created in a Windows environment and describes the development process in that environment.

The easiest and quickest way to install and start developing in Rails is using packaged installer systems like RailsInstaller. However, there had been SSL certificates problems on RubyGems at the time of developing the application and RubyGems 2.4 on Windows had also been broken. Fortunately fixes on certificates were provided online to manually install the updates or changing the trust certificates and these steps were taken at the time of development.

4.2 Creating the Rails Application

After finishing the installation process, a directory named Sites will be created from where the development process will commence. The command prompt with Ruby and

Rails will be used to create the directories and files that will be needed and to also do installation or update of Gems. Navigating to Sites folder through the command prompt and running the command – **`rails new appname`** - will create a partially pre-configured development environment that has several folders and subfolders.

4.3 Gems

Gems were briefly discussed in section 3.3 and this section will concentrate on the Gems used in the application. Rails framework has significant number of diverse Gems for developers to choose from according to their needs. They can be considered as packaged software that contains libraries that can easily be installed through the Rails command prompt through the command – **`gem install gem-name.`** The other way we can add Gems to our application is going through the *Gemfile* and add the Gem we want to include in our application. We can as well comment out a Gem in the Gemfile to stop using it in the application. After editing the Gemfile to include or exclude the Gem, we have to go back to the command prompt and run - **`bundle install`** – command for the bundler to install or uninstall the Gem and resolve its dependencies. The Gemfile is created by default on our application root directory and hold the basic gems needed to start developing. The Gemfile in Metprotracker is shown on appendix 1. The main Gems included in this application, other than the ones provided by default through Rails, that are worth mentioning are :

Apartment Gem is a convenient gem that deals with database multitenancy through ActiveRecord. It handles the subdomaining feature of the application keeping the data separate at the database layer for Rack applications.

Devise Gem provides a solution for a flexible and scalable authentication mechanism for Rails with Warden. Correspondingly, **devise_invitable Gem** adds authentication required invitation functionality by giving a mechanism to send invitations by email and accept them by a password setting. It does that by adding an **`invite!`** method to model that can accept attributes like calling a **`create`** action.

Pg Gem is a Gem that provides an interface to connect and integrate the PostgreSQL relational database management system. Rails has by default SQLite and Pg Gem

should be added and SQLite Gem disabled to use PostgreSQL as the RDBMS for the application.

Bootstrap-sass Gem is version of Bootstrap that is powered by Sass (Syntactically Awesome Style Sheets) and is organized to drop into an application. Bootstrap is a framework that allows developers an easy way to design a user interface for web applications by providing predefined CSS and javascript classes that can be integrated with an application. Sass is a CSS3 extension that adds variables, mixins, inheritance etc to CSS. Bootstrap-sass gem should have sass-rails to work but rails includes sass-rails by default when starting new application.

Letter_opener Gem is a Gem that offers a convenient mechanism to view email on the browser instead of actually sending one. This avoids the necessity to set up an email delivery mechanism during a development environment which minimizes the complication of doing so while still avoiding accidentally sending emails to people while testing in development stage.

Simple_form Gem is a Gem that provides powerful components to craft forms without affecting the layout of the application. Its flexibility enables users to manage different ways to present their form in their application.

4.4 Configuration

In rails a **router** is the first component that receives, handles, and dispatches requests from the web and match them to the necessary controller action. It also routes the web request for the homepage to the home page route defined in it. The root path to the application and the resourceful controller actions are enlisted and defined in a *routes.rb* file that is found in the *config* directory of the application. Resources enlisted for the controller actions can be **`index`**, **`new`**, **`create`**, **`show`**, **`edit`**, **`update`** and **`destroy`**. When browsers request pages from a Rails application they use specific HTTP methods like GET, PUT, PATCH, POST, and DELETE which are requests demanding a specific action to be done on the resource. The router will then map these HTTP verbs and URLs into actions that are in the controller which by convention will then be mapped to CRUD **`(Create, Read, Update,`** and **`Delete)`** operations in

the database and manipulate the data as requested. An entry **`'resources :projects'`** in the routes files will create seven routes in the application.

Table 1. Rails Routes

HTTP Verb	Path	Controller#Action	Used for
GET	/projects	projects#index	display a list of all projects
GET	/projects/new	projects#new	return an HTML form for creating a new project
POST	/projects	projects#create	create a new project
GET	/projects/:id	projects#show	display a specific project
GET	/project/:id/edit	projects#edit	return an HTML form for editing a project
PATCH/PUT	/projects/:id	projects#update	update a specific project
DELETE	/projects/:id	projects#destroy	delete a specific project

Table 1 shows the HTTP verb requests (column 1) coming from browsers with the path (column 2) to a controller action it will be directed to by the router (column 3) and the action it will perform on the data (column 4) created by the adding a **`'resource :project'`** to our router. Routes of an application can be viewed by running the **`'rake routes'`** command. Appendix 2 shows Metprotracker routes.

In the routes file a root for home page of the application can be routed with a line **`root 'Welcome#index'`** if the root page is Welcome/index which is the default root path coming with rails. In the case of Metprotracker, there should be a mechanism to route the request for the website into a subdomain present case or a subdomain absent case. That is because a functional subdomain accessible to users will be provided to them only after registration and there should be a mechanism for them to access the site without a subdomained path and therefore the registration home page will be a case when the subdomain will be absent. For this we need to have conditional statements in the router file to direct the request accordingly as such:

Listing 10 – Routing Conditions and Definitions

```
class SubdomainPresent
  def self.matches?(request)
    request.subdomain.present?
  end
end

class SubdomainBlank
  def self.matches?(request)
    request.subdomain.blank?
  end
end

Rails.application.routes.draw do
  constraints(SubdomainPresent) do
    root 'projects#index', as: :subdomain_root
    devise_for :users
    resources :users, only: :index
    resources :projects
    resources :reports
  end

  constraints(SubdomainBlank) do
    root 'welcome#index'
    resources :accounts, only: [:new, :create]
  end
end
```

Listing 10 shows how request for the site will be conditionally directed to the appropriate root path with the conditional access to the resources of the controllers of the application. This will mean that an initial request for access to the application website will be directed to a 'welcome/index' where a registration access is given to first time users. The resources that can be accessed to such users will also be only *accounts* with two actions **`new`** and **`create`** which will be the only needed resources for first-time or returning users that need to create an account.

On the other hand, users that have registered and provided a registered subdomain through their http request will be directed through the **`SubdomainPresent`** class to **`projects#index`** root path after authentication. They will also have much more resources and controller actions to access and use through their subdomain path after authentication.

4.5 Database Connection

A database connection should be configured to establish a connection with the locally downloaded PostgreSQL database management for the application to access and

store data. Three environments – development, test and production – are created by default in rails to simplify and tidy up development environment. For these environments three separate database connection options are available by default and separate connections could be made for each by modifying the config\database.yml file for development. The connection parameters that we set on our postgreSQL should be passed to the development environment. This part of the file should be edited according to the application's and database's information for the development environment.

Listing 11 – Establishing Database Connection

```
development:
  <<: *default
  adapter: postgresql
  database: metprotracker_development
  encoding: unicode
  username: postgres
  password: jixah
  host: localhost
  pool: 5
```

Listing 11 shows how the database.yml file should be set for the development environment to establish a connection with the database.

4.6 Model-View-Controller (MVC)

Controllers are the central part of the MVC model that control the flow of the application by using their action methods to render views and manipulate data from database. Of the controllers in Rails applications ActionControllers make the core of web request and are made up of actions (methods) that will be accessible to web-server. Only ApplicationController extends to ActionController by default and other controllers further extend to ApplicationController which is located at app\controllers directory.

An **ApplicationController** by default in Rails 4 includes **`protect_from_forgery with: :exception`** method which is found in ActionController. It prevents CSRF (Cross-Site Request Forgery) attack which is an attack that makes an authenticated user send malicious request that might alter the state of the user data, not steal the data. This protection is done by including a token based on the session with requests

and verifying the authenticity of the token in the controller. With the inclusion of an argument of 'exception' in the **protect_from_forgery** method, an exception will be raised with failure of authentication. This method will remain in the ApplictionController of the application for this purpose.

After this method, filters will be included in ApplicationController of the application and this will make the filters effective in all other controllers in the application because all others extend to ApplicationController. If need be, this filter can be circumvented in other controllers by **skip_before_filter** method specific to the controller or its actions. The first filter we will have in the ApplicationContoller and its methods is the **before_filter :load_schema.**

Listing 12 – ApplicationController Definitions

```
class ApplicationController < ActionController::Base

  protect_from_forgery with: :exception

  before_filter :schema_load
  before_filter :authenticate_user!
  before_filter :set_mailer_host
  before_filter :configure_permitted_parameters, if: :devise_controller?

  private

  def schema_load
    Apartment::Tenant.switch('public')
    return unless request.subdomain.present?

    if current_account
      Apartment::Tenant.switch(current_account.subdomain)
    else
      redirect_to root_url(subdomain: false)
    end
  end

  def current_account
     @current_account ||= Account.find_by(subdomain: request.subdomain)
  end
  helper_method :current_account

  def set_mailer_host
    subdomain = current_account ? "#{current_account.subdomain}." : ""
    ActionMailer::Base.default_url_options[:host] = "#{subdomain}lvh.me:3000"
  end
```

Listing 12 shows the ApplicationController with its filters and methods. This loads the schema by running the **schema_load** method which uses Apartment Gem methods to switch tenants with **Apartment::Tenant.switch('tenant_name')** command. With the call of **switch** all requests coming to ActiveRecord will be routed to the tenant specified. The schema_load method will have conditionals to check tenant and load the tenant requested on the subdomain. The current_account method then sets the

account from the subdomain. Mentioning the current_account in the helper_method is done because the current_account method will be reused in the views.

The **`authenticate_user!`** class method (Listing 12) ensures a logged in user will be available to all controller actions, and when **`before_filter`** method is used with it, Devise will check if the user is logged in and will back off and redirect if it not. Other methods are also included in the ApplicationController for mail invitation and confirmations options.

The **WelcomeController** is the controller that will be responsible to serve the first request to the application site. It should have a **`skip_before_filter`** to remove the specified filter in the ApplicationController to bypass the authentication to give a door of access to the application and serve users that have not registered yet. The **`skip_before_filter`** can also be specified by the **`:only`** and **`:except`** options to limit on which actions in the controller it will be effective. This controller will only have an index action since it will only be responsible to render the home page view of the website. Listing 13 shows the WelcomeController with its index action.

Listing 13 - WelcomeController

```
class WelcomeController < ApplicationController
     skip_before_filter :authenticate_user!, only: :index

    def index

    end
end
```

After the welcome page is rendered which gives users options to register to the application, the **AccountsController** takes over to enable users to create an account. Since these actions so far will be carried out before authentication process, the **`skip_before_filter`** should also be included in the AccountsController to give access to the application. Two actions will be included in this controller which are **`new`** and **`create`** which will be needed to create an account.

Listing 14 – AccountsController

```
class AccountsController < ApplicationController
    skip_before_filter :authenticate_user!, only: [:new, :create]

    def new
        @account = Account.new
        @account.build_owner
    end

    def create
        @account = Account.new(account_params)
        if @account.valid?
          Apartment::Tenant.create(@account.subdomain)
          Apartment::Tenant.switch(@account.subdomain)
          @account.save
          redirect_to new_user_session_url(subdomain: @account.subdomain)
        else
          render action: 'new'
        end
    end

    private
    def account_params
        params.require(:account).permit(:subdomain, owner_attributes: [:first_name, :second_name,
            :email, :password, :password_confirmation])
    end

end
```

Listing 14 shows the AccountsController with the two actions **`new`** and **`create`**. The new action will render a registration form for users to input their information. Upon successfully entering the information and submitting it, the controller grabs the values in the **`@account`** variable and passes them to the create action which will be responsible for creating and registering the profile and its details. The **`create`** action will include a conditional statements to check the validity of the account to be created and create it if the information is permitted or give another filling form if it is not. At the bottom of the controller, parameters will be defined when there is a data sent by the user that should be accessed in the controller action. In Rails 4 the use of strong parameters is necessary to provide an interface for the protection of attributes from end-user assignment. This will make parameters of the Action Controller forbidden for mass assignment in the model until they are whitelisted. An option of labeling them as required will make them stream through predefined rescue/raise flow to end up as a 400 Bad Request. The parameters the AccountController will use are written as private at the bottom of the controller as shown in Listing 14.

The **ProjectsController** in this application will hold more actions and parameters than the other controllers. It will not need a **`skip_before_filter`** condition because it will be after authentication that the ProjectsController will be accessed and used.

Listing 15 – First half of ProjectsController

```
class ProjectsController < ApplicationController

    def index
        @projects = Project.all
    end

    def new
        @project = Project.new
    end

    def show
        @project = Project.find(params[:id])
    end

    def create
        @project = Project.new(project_params)
        if @project.save
            redirect_to root_path, notice: "Project created!"
        else
            render :new
        end
    end

    def edit
        @project = Project.find(params[:id])
    end
```

ProjectController is divided into two parts for displaying purposes.

Listing 16 – Second half of ProjectsController

```
    def update
        @Project = Project.find(params[:id])

        if @Project.update(project_params)
            redirect_to project_path, notice: "Project updated!"
        else
            render :edit
        end

    end

    def destroy
        @project = Project.find(params[:id])
        @project.destroy

        redirect_to projects_path, notice: "Project deleted!"
    end

    private

    def project_params
        params.require(:project).permit(:name, :client, :description,
        :billable, :estimated_hours, :actual_hours, :payment_per_hour, :archived)
    end
end
```

Listing 15 and 16 show the ProjectController with its actions and parameters. The project **`index`** action is responsible for fetching all the data from the saved projects in the database. The project **`new`** action sets an instance variable **`@project`** to receive the data that will be received from the user and pass it to the **`create`** action when successfully submitted. The project **`create`** action will then grab the values provided and check them on a conditional statement whether the project is saved or not and redirect to the necessary path accordingly afterwards giving a notice upon the successful creation of a project. At this time the **`create`** action will write the data into the database.

Similarly the project **edit** action will receive values from the user and pass them to the project **update** in a similar way the project **new** action does to project **create**. The difference here is that the project **edit** action uses a **Project.find** method to fetch the values of the attributes of a project using the project id as a key. Lastly, the **destroy** action will be activated when a project is deleted and takes the project id as a key to fetch the necessary data belonging to the project and perform a **destroy** method on it deleting this information from the database. The parameters that will be accessed by the actions of the controller are listed at the bottom of the controller.

The **ReportsController** on the demo version of the application will only include an **index** action that will receive the values of all the projects by a **Project.all** method. This controller then will make this data available to the report view to be manipulated and presented as will be specified in the view section. Listing 17 below shows the ReportsController with its index action and the resource it uses.

Listing 17 – The ReportsController

```
class ReportsController < ApplicationController
  def index
    @projects = Project.all
  end

  private

  def resource
    @project ||= Project.all([:id])
  end

end
```

The users of the application will be largely managed by Devise. Therefore, the **UsersController** in this application will only use an **index** action which will use a **User.all** method to get the users from the application for users view page.

Listing 18 – The UsersController

```
class UsersController < ApplicationController

    def index
        @users = User.all
    end

end
```

Listing 18 shows the UsersController that is added to the application that will be used for users view.

The **model** part of the MVC architecture holds the business logic or the database part. A postgreSQL is used for Metprotracker's RDBMS for a couple of reasons. The first one is that postgreSQL has a better advantage for deployment as most free test deployment services use postgres for their databases. In addition, postgres is preferable and more compatible for applications using multitenant architectures.

The production level of Metprotracker will implement a more accurate and detailed data schema than the demo version of the application. The demo version of the application makes some simplifications in the database design in places where the features of the application to demonstrate will not be affected significantly. These simplifications involve approximating tables which will stand alone in the mature application to attributes in the demo application. For instance, a clients table with client details such as company name, address, company logo etc will be implemented in the mature application while client name will be added as an attribute to projects table in the demo version. This will not affect the integrity of the application in a significant way and would rather make the demo version more manageable to present the application's functionality.

The other simplification is the use of projects table as the main registration medium while in the mature application the project table will further extend to a tasks table enabling users to divide projects into one or more tasks. As in the client approximation case above, registering tasks as different projects in the demo version will replace the task division functionality. This means users can register their project tasks as separate projects and use the analysis features available for projects in the demo version. Hence, registering tasks as independent projects will not compromise the integrity of the demo application, and project comparison, analysis and presentation features in the demo application will be implemented to tasks in the final application.

When considering the time management aspect of the project, another simplification taken was using hour as an integer making it the smallest measurement scale for the demo application. Realistically, projects could take several hours to finish depending upon their scale but a scenario where measuring project durations to a more precise time scale will definitely be a possibility. Therefore, hours will further be divided into minutes or the option for so doing will be provided in the final application. The aforementioned simplifications are the major simplifications that should be mentioned. It should be noted that a careful consideration was made when these simplifications were done not to affect the demonstration capacity of the demo version to show the main

functionalities of the application. Apart from the removal of the simplifications, extending the features of the application is also in progress.

The **account model** of Metprotracker is responsible for the registration of accounts that will be created by users. The accounts table holds the account details of registered accounts to the application. It is a relatively simple table holding the account id, the subdomain name of the account, the owner id, the creation and updating date of the account. The tables of the database can be viewed from the db\shema file on the application. Models of Rails application are found in app\models and are ruby files that have a class with the same name as the controllers and views, and they extend to `ActiveRecord::Base`. These files contain association definitions and may also contain restrictions, validations and methods that control how data is written into the database. The account model in Metprotracker has a `belongs_to` association with owner (user) as can be seen in Listing 19 below.

Listing 19 – The Account Model

```
class Account < ActiveRecord::Base
    RESTRICTED_SUBDOMAINS = %w(www)

    belongs_to :owner, class_name: 'User'

    validates :owner, presence: true
    validates :subdomain, presence: true,
        uniqueness: { case_sensitive: false },
        format: { with: /\A[\w\-]+\z/i, message: 'contains invalid characters' },
        exclusion: { in: RESTRICTED_SUBDOMAINS, message: 'restricted' }

    accepts_nested_attributes_for :owner

    before_validation :downcase_subdomain

    private
    def downcase_subdomain
        self.subdomain = subdomain.try(:downcase)
    end
end
```

The **User Model** of Metprotracker is responsible for the registration of the users of the application. The users table is a devise and `devise_invitable` managed table and has a number of attributes that are needed to perform user tasks, such as invitations. In the user model, devise provides 10 modules that we can include into the model. From these the ones that might be used in the application are :

- **Database Authenticatable** - for encryption and storage of a password in the database to validate the authenticity of a user while signing in.
- **Invitable** – adds support for devise to send invitations through email.
- **Recoverable**: for resetting password and sending instructions for reset.

- **Rememberable**: for the management of the generation and clearing of a token to remember user from saved cookie.
- **Validatable**: provides optional use and customization to validations of email and passwords. [15]

These modules are included in the user model by preceding them with the word **`devise`** as such – **`devise :invitable, :database_authenticatable, :recoverable, :rememberable, :validatable`** and devise will configure the model based on these modules. The users table is shown in appendix 3.

The Project Model of Metprotracker is responsible for the management and registration of projects in the application. The projects table in the demo application is designed to hold the attributes and values that are needed for the analysis of projects. These attributes might be removed or new ones added on the final application as needed. One of the scenarios where some attributes might be removed is, as mentioned above, when additional tables are introduced into the application. This will make the data more manageable and the application more resourceful. Some of the tables that will be added are client and task table at which time the client attribute and some of the time registration attributes will be migrated to the new tables with proper relations with the project table. The attributes of the project table in the demo application are shown in figure 1.

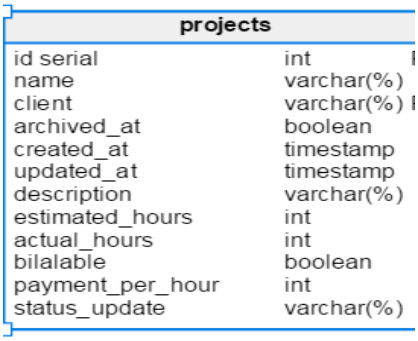


Figure 1. The Projects Table

These attributes of the projects table will be used for the analysis and reporting of the project data and explaining some of the attributes is helpful to better understand the functionalities. The archived_at attribute will hold a Boolean value which will be set to false by default. This attribute will be used to log the state of a project where a true value will be used to mark the completion of and a false value to show the project is in

progress. The estimated_hours takes an integer to register the number of hours the user will predict the project will take initially before the project starts and the actual_hours will register the actual time the project took upon the completion of the project or the actual hour through the progress of the project. The billable attribute takes a Boolean, true meaning a project will be a paid/billed project. Another attribute payment_per_hour will register how much an hour working on the project will be charged.

Views are the displaying mechanisms of the MVC model and determine how the user sees the results of the data and provide an interface to change them. Views are found in app\views directory and take the name of the controllers in plural and have further erb (embedded ruby) files based on the action names in the corresponding controller. These files can contain html, css, javascript and ruby codes. Ruby codes are enclosed in **<% %>** or **<%= %>** tag, the former signifying there is no views to render in that specific ruby code line and the later showing there is a view to render.

Views contain forms which are rendered to the user to input values into and should as much as possible be free from intense calculations of data. There is a directory called helpers that are found in app section where methods can be written and called from the views to avoid cluttering of the views with methods. The important contents of the view files in Metprotracker will be discussed briefly below showing their corresponding outcome when necessary.

The **layouts view** in Metprotracker holds the application.html.erb file that will be used to render any page views of the application. The accumulated result from any view in the application will be inserted in a **<%= yield %>** section of this file when the application is running. Since this file will be used whenever a view from the application is rendered, resources that will be needed in all the pages, such as navigation tabs, links and flash messages should be included in this file. In addition, it contains **stylesheet_link_tag** and **javascript_include_tag** that add CSS stylesheets and JavaScript respectively from the Rails Assets. Bootstrap and javascript are used for front-end development of the application. The navigation bars in the demo application are included in every page of the application by adding the code in listing 20 below in the application.html.erb file.

Listing 20 – Application View File

```
<header class="container">
  <nav class="navbar navbar-info">
    <div class="navbar-header">
      <strong><font face="algerian">
        <%= link_to 'Metprotracker', root_path, class: 'navbar-brand navbar-bg' %></font>
      </strong>
    </div>
    <% if user_signed_in? %>
      <div class="pull-right right-buttons">
        <%= link_to 'Sign out', destroy_user_session_path, method: 'delete', class: 'btn btn-info navbar-btn' %>
      </div>
      <ul class="nav nav-tabs pull-right">
        <li><%= link_to 'Create Project', new_project_path %></li>
        <li><%= link_to 'Projects', projects_path %></li>
        <li><%= link_to 'Users', users_path %></li>
        <li><%= link_to 'Project Reports',reports_path %></li>
      </ul>
    <% end %>

  </nav>
</header>
```

The **Welcome View** will be the page visitors will see when they access the homepage of the application before registration and will only contain an index.html.erb file. The router directs the web request for the website to this page, as discussed in the route section earlier, and the WelcomeController with its **`index`** action will display the index.html.erb file. This page will display simple description about the application and a tab that will lead visitors to a page that will enable them to create an account. This view as well as the other views that will be discussed thereafter will use bootstrap styling on parts of their content. The welcome page as will be seen by users is shown on appendix 4.

The **Accounts View** contains a new.html.erb file that includes forms to input registration information for the user to create an account. As the AccountsController mentioned earlier contains only two actions – **`new`** and **`create`** – no additional views are required for it. The new action in the controller will render the accounts/new view for the user to input the data into, then upon successful entry and submission of the data, it will use the **`create`** action to write this data into the database. Upon successful validation of the information provided to create a new account, the **`create`** action in the AccountsController will save the information and redirect the users to their new assigned subdomain address where they will get the devise users view to sign into. Appendix 5 shows the registration form.

Devise Views are located in devise\views directory that contains a number of directories that deal with different actions such as authentication, invitation, password reset

and so on. The first of these that will be reached on the application is the user authentication page. After the creation of an account, the application will immediately redirect the address to the registered domain name of the user and render a user sign in page. extracted from the devise\sessions\new.html.erb file. This file is styled with CSS and bootstrap and contains devise methods such as **`'rememberable?'`** that gives an option to remember password and give a selectable button for it. Upon successful authentication, the user will be signed in to the projects home page which will be discussed in the projects view section. Appendix 6 shows the sign in page.

Another functionality offered by the devise view will be resetting of password. This option will ask for an email input and send a resetting token link to that email. Letter_opener gem is used in development mode to open the email in the browser to follow the link (and mimic the real email sending process) to give a devise password resetting form to create a new password.

A users view is also included in the application in addition to the devise managed users view. This view uses the UsersController mentioned in the controller section with its **`index`** action to show the users of the application. This feature is added for project collaboration option to allow users to work on projects as a team in one account. This is done when the original user that has registered sends invitations for people to join in by filling out their email. The person that is invited can then follow the link that arrived through email to get a registration page for the original user's subdomain where the invited person can then create an account and join into the same account as the inviter. The users view will then show the successfully accepted users that are using that account. Appendix 8 shows the list of users in an account and their status.

The **projects view** directory has four view files which are index, new, edit, and show which are controlled by ProjectsController actions of corresponding names. The first page that the user will view after being authenticated and signed in will be the project/index page created from the app\views\project\index.html.erb file. This page will populate the registered projects in the account if there are any upon signing in. But if there are no registered projects, which will be the case when a user signs in for the first time, this page will be an empty page with no instruction how to get started with using the application. This is solved by using a conditional statement to check if there are any registered projects to show and, if none, display a welcoming message with a link to

create a new project. In this way the same index page will serve both kinds of users in the same page.

Listing 21 – Conditional in Project Index

```
<% unless Project.any? %>
    <div class="panel panel-default">
        <div class="panel-heading">
            <h2> <font face="david">Welcome to Metprotracker!</font></h2>
            <i><strong> An easy way to manage and time your projects. </i></strong>
            </br>
            </br>
            </br>
            <%= link_to "Create Your First Project", new_project_path, class: 'btn btn-primary' %>
        </div>
    </div>

<% else %>
```

Listing 21 shows the conditional check done before showing a welcoming note. The **`unless Project.any?`** line checks if there are any projects registered in the database. If there are none, the code below that condition will be executed giving a welcoming note (as shown in appendix 7) with a link to create new project. On the other hand if the user is a returning user that already has registered projects, the part of the page after the **`else`** will be executed and the registered projects will be populated (as shown in appendix 9).

Creating a new project can be done either from the welcoming page tab or from the tab provided at the top of the page. This tab will lead to a new project create view that will give the user a form to fill in about the project to be registered. This form is included in the views\projects directory to be rendered when other views of a project request.

Listing 22 – Project form

```
<%= simple_form_for(@project,
    wrapper_mappings: { boolean: :vertical_boolean }) do |f| %>

    <%= f.input :name, label: 'Project Name'%>
    <%= f.input :client, hint: 'If no client, fill "Self"' %>
    <%= f.input :description %>
    <%= f.input :estimated_hours %>
    <%= f.input :billable, hint: 'Select if you charge a bill'  %>
    <% if @project.billable = true%>
        <%= f.input :payment_per_hour, required: true, hint: 'Fill 0 if not billable' %>

    <% end %>

    <% if @project.persisted? %>
        <%= f.input :actual_hours, hint: 'Time spent to-date' %>
        <%= f.input :status_update, hint: 'Update what you did so far' %>
        <%= f.input :archived, as: :boolean, default: false, label: 'Complete' %>
    <% end %>

    <%= f.button :submit, class: 'btn-primary' %>
<% end %>
```

Listing 22 shows form for projects that will provide project attributes' input fields. An important thing to note here is the condition **`if @project.persisted?`** that is found at the middle of the form. This condition checks whether the from is being called in a certain project for the first time or not. This is important because there will be fields when creating a new project that would not make sense to be filled out such as actual hour or status update. So the project form will filter out these fields when project is created for the first time and display the fields when reopened (edited) which will make sense logically as well because users can not fill out actual hour that a project took if they did not start working on it.

The other field it gives out when the form persists is the archived field which is a Boolean and false by default. This field will be used to register the status of the project (true being a completed project) and this field will also not be available, rightly, when creating a new project. Thus, in essence the difference between the project/new and project/edit will be the availability of these fields. But the filling out of these fields, especially the archived field, will make a significant difference because it is this condition that will trigger some of the features of the application.

The project/show view uses the **`Project.find`** method with the project id passed as a key defined in the ProjectsController. It will assign that to an instance variable **`@project`** of the particular project id in the controller. This variable then is accessed in the view by passing the corresponding attribute as a method to fetch the value of that attribute. For example, project/show/1 will give the estimated hour registered for project id 1 when **`@project.estimated_hour`** is called.

The project/show view is more dense than the other views because it provides additional analysis of the data registered on the project. A number of conditional statements are used to make the projects/show more clean and informative for the users. These conditions are used to check the status of the project and offer options to change them while keeping the projects show readable avoiding empty fields. Then it will analyse the data according to the conditions and provide views according to its status.

The first project view a user will get after creating a new project is a basic view that will display the project name, the client name, the description, its estimated hour for completion and its payment (as shown in appendix 10). Together with these fields, tabs are created to 'update', 'delete' or 'complete' the project. The update tab will be used when

a user works on a project and wants to update an actual working hour that took him/her so far. The update tab will render the form in listing 22 above with the **@project.persisted?** condition true, thus, rendering the additional actual hour registration field. Here another condition is placed to check if any actual hour has been filled thus far. If an actual hour is filled, it will make the user fill a status update. The status update field will add the so far worked hours to it and display on the project show a handy status that describes what had been so far and in how many hours. This conditional field is shown listing 23 below.

Listing 23 – Conditional to Check Actual Hours

```
<% if @project.actual_hours != nil %>
    <li class="list-group-item"><strong> Actual Project Time --  </strong><%= @project.actual_hours %> hrs</li>
    <li class="list-group-item"><strong> Status Update --  </strong>After <%= @project.actual_hours %> hrs - <%= @project.status_update %> </li>
    <li class="list-group-item"><strong> Last Update --  </strong><%= @project.updated_at.strftime("%b %d - %Y @ %H:%M") %> </li>
<% end %>
```

This loop will continue as the user keeps working on the project and update the actual hour field. An embedded javascript digital clock is available on the projects/show page for the user to refer from to register the duration of work easily. When the user finishes working in the project, a ‘Mark Complete’ tab will take the user to edit the project for the last time (can be reopened to mark it incomplete) and update its final actual hour field and mark the project complete. There are conditional statements that continuously check the status of the project (completed or not completed) whenever a show view is requested. The if statement that checks the completion of a project is –

```
if @project.archived == true
```

When this condition becomes true the methods protected by this condition which had been invisible to the user, thus keeping the projects view simpler, will be executed (as shown in appendix 11) that do the following things.

Project earnings is a field that calculates the amount of money earned by the user while working on a billable project. It uses the payment_per_hour data with the estimated_hours data assuming the client will be liable to pay the agreed rate per hour for the initially estimated hours the developer submitted. The method for this calculation is placed in the project helper directory together with other computations to simplify the view file. Listing 24 below shows this method found in the helper.

Listing 24 – Earnings Filler method

```
def earnings_filler
    if @project.payment_per_hour == nil
        'None'
    else
        @project.estimated_hours*@project.payment_per_hour
    end
end
```

Efficiency is a field which calculates and fills the efficiency in percentage of the user based on the estimated time the user provided initially and the actual time the project took. This will tell users important information about how good they are on the topic and how their time management could be adjusted. The helper method written to calculate this is shown below in listing 25.

Listing 25 – Efficiency Calculater method

```
def efficiency_calculator
    ((@project.estimated_hours.to_f/@project.actual_hours.to_f)*100).to_i
end
```

Since estimated hours and actual hours are registered in integer, an adjustment should be made on the variables because division of integers will give an integer that will lead to an unreasonable information loss. For example, dividing 3 by 2 will give 1 which in percentage will be 100 rather than 150. Therefore, both values should be changed to float so that their division will be performed in an accurate manner. This then can be changed to percentage by multiplying it with hundred and changed back to integer to have a cleaner value.

Efficiency adjusted earnings/hour is a field which is useful for freelancers who estimate a project working time and payment at first. It adjusts the actual payment per hour based on their efficiency in completing the project. In other words, it tells the users how much they really got paid per hour for actually working on the project distributing the money from the time they saved on finishing early or the time they lost in getting late. The helper method for doing so is shown on listing 26 below. It should be noted that the methods include conditions to check if the project is not billable.

Listing 26 – Payment Adjuster method

```
def efficiency_adjusted_payment_calculator
    if @project.payment_per_hour == nil
        'None'
    else
        ((@project.estimated_hours*@project.payment_per_hour)/@project.actual_hours).round(2)
    end
end
```

The **remarks** feature tries to comment on the efficiency of the project and give short verbal suggestions for future projects of similar type. While finishing projects early and getting an extra free time from doing so is a good thing for students or other users, there is a chance that they might not have made considerations on how to use that extra time for productive purposes and that time might be wasted. The remarks section takes this in to considerations and pushes for efficiency close to 100% for an ideal schedule planning and time allocation. Therefore, efficiencies higher than 100% will also include comments for improvements in time allocation even though higher efficiency is not a harmful thing by itself. The helper method for this uses switch cases to compare the efficiency variable with defined value ranges and return comments in string format as shown in listing 27. (cases could be added in the future for a more friendly and accurate remarks)

Listing 27 – Remark Generator method

```
def remarks_generator
    efficiency = ((@project.estimated_hours.to_f/@project.actual_hours.to_f)*100).to_i
    case efficiency
    when 0..30
    str = "Very Low Efficiency! Project highly Underestimated! Consider allocating significantly longer time for similar projects."
    when 31..60
    str = "Low Efficiency! Project Underestimated! Allocate more time for a similar project."
    when 61..90
    str = "Medium efficiency! Project slightly Underestimated! Consider slight adjustment of time allocation."
    when 91..110
    str = "Efficient! You had an excellent time allocation and a good understanding of your potentials."
    when 111..150
    str = "High Efficiency! Project slightly Overestimated! Consider slight adjustment of time allocation."
    when 151..250
    str = "Very High Efficiency! Project Overestimated! Consider allocating lesser time for similar projects."
    when 251..1000
    str = "Exceptional Efficiency! Project overly Overestimated! Consider allocating significantly shorter time for similar projects"
    else
    str = "No Remarks!"
    end
return str
end
```

JavaScript Charts are the charts in Metprotracker that use front end sketching of interactive charts by using javascript and uses highcharts library for it. The script is written in the projects\show.html.erb file using bar graph to show estimated hours and actual hours side by side. It will not be activated until a project is complete because it will be shielded by the **`if project.archived == true`** condition. The javascript func-

tion of the bar chart renders to a **div** class named in the function and this div class will be defined in the html.erb file of the application in the place where the chart should be displayed. The part of the code in the javascript for rendering to is:

```
chart: {
            renderTo: 'projects_chart',
```

where a **div** class defined by the name projects_chart in the show file together with completion check condition as shown in listing 28 below.

Listing 28 – div Class Definition for Charts

```
<% if @project.archived == true %>
    <div class="col-md-6 panel panel-default">
        <div id='projects_chart' style='min-width: 310px; height: 400px; margin: 0 auto'></div>
    </div>
<% end %>
```

The x-axis label takes string array with single quote while the y-axis label takes a single string value ‘Hours’ as shown in listing 29 below.

Listing 29 – xAxis and yAxis Labeling

```
yAxis: {
     title: {
          text: 'Hours'
     }
},
xAxis: {
     categories: ['Estimated Hours', 'Actual Hours']
},
```

The dynamic data feed for the plot takes an array of integers for the bars of the bar graph and this data is passed to the function from the application as shown in listing 30 below.

Listing 30 – Dynamic Data Feed for the Bar Graph

```
series: [{
    data: [<%= @project.estimated_hours %>,<%= @project.actual_hours %>]

    }]
```

The **reports view** uses the project data and does not have its own table or model in the database for the demo application. The reports view contains table, line graphs and pie charts to show the overall report and data analysis of the projects of a user as shown in appendix 12. This can be used to compare the productivity of projects with each other. (how this will be translated to tasks in the main application will be discussed below). The table shows selected data from each project in one table for an easy comparison between projects to enable the user to make decisions on future planning and choosing of types of projects. The columns on the table contain registered and calculated data from all projects such as the project name, client, the estimated time, the actual time, efficiency and earnings.

The Javascript **line chart** of the reports draws two line graphs on the same chart to visually present the difference in estimated and actual time of each project. This visual presentation offers a mechanism for comparing these two time fields for a project as well as the difference in efficiency between each project by placing registered projects side by side to easily spot time efficient projects visually. The line chart is rendered to a **`div`** class defined in the reports\index.html.erb in a similar way as the bar chart (as explained in the project show section) and the div class is defined below the table to make the script sketch the line graph below it.
The dynamic data the line chart takes for the x-axis can be in different variable forms but on the case of strings, the script for the Highcharts library takes strings or string arrays enclosed in a single quote. This created incompatibility because Ruby on Rails returns string arrays enclosed in double quotes. Trying to map returning double quoted string arrays from the application to single quoted ones for Highcharts leads to greater chance of instability and data corruption because quotes in the project name itself will affect the result. Therefore, the x-axis label is populated by project id (integers do not have the same issue) for the time being until Rails or Highcharts (both aware of it) provide a stable option for this incompatibility. It should be noted that labelling the x-axis with project ids rather than project names will only affect the readability but not the data. The **`pluck`** method is used to pick the values needed from the application database and pass to the script function as shown in listing 31. The line chart is interactive providing such options as pointing for values and removal of part or all the line series.

Listing 31 – Value Passing for Line Chart

```
title: {
     text: 'Project Charts'
},
xAxis: {
     title: {
          text: 'Project ID'
     },
     categories: <%= Project.pluck(:id) %>
},
yAxis: {
     title: {
          text: 'Hours'
     }
},
series: [{
     data: <%= Project.pluck(:estimated_hours) %>,
     name: 'Estimated Hours'
     },
     {
     data: <%= Project.pluck(:actual_hours) %>,
     name: 'Actual Hours'
}]
```

The **pie chart** is used to show the actual time breakdown of the user for each project. It will divide the chart proportionally by the time taken for each project and efficiency is not visualized in this presentation unlike the other charts. It rather shows the users which projects take more of their overall project working time than others. Similar incompatibility issue persists in passing string array parameters as mentioned above and the section labels of the pie graph will be labelled with a project id. The pie chart function of the javascript accepts data as an array of two-dimensional label-data array as:

```
[[label1, data1], [label2, data2],…...]
```

A method in rails to take two one-dimensional arrays and combine them into one array of two-dimensional array data is:

```
c = a.zip(b)
```

where the first values of array a and array b will combine to form an array with two values which will be the first value of the array c. In a similar way the data for the pie chart will be passed as shown in listing 32 below.

Listing 32 – Value Passing for Pie Chart

```
series: [{
     type: 'pie',
     name: 'Project Share by Project ID',
     data: <%= Project.pluck(:id).zip(Project.pluck(:actual_hours)) %>

}]
```

In the complete Metprotracker application, the reports part of the application will be replaced by the report of one project with the introduction of tasks for each project. On such a case the tasks of each project will be populated under the project each task belongs to and a table, line and pie charts will be showing the details of the time and payment breakdown for each task. An option for the removal of actual hour field will be available so that the estimated time data and payments for it will only be shown in a table to get a downloadable invoice for the client. Another option for the overall projects analysis will also available that can be queried over a duration of time for the personal use of the user to see the time breakdown for each project in a certain month or week.

5 Results and Discussion

5.1 Outcomes and Limitations

The application in the current state is developed to show the design concept and functionality of a bigger application with a similar purpose. For that reason, this application and its documentation should be considered as a demonstration mediums for an application that will be built in a more detailed and resourceful state. However, careful considerations were made to show the essence of the full application in this demonstration version. This version is sufficiently complete in its own scope in showing the main purpose of the application and, using it as a base, a more rigorous and meticulous development and testing mechanisms will be implemented to widen the scope and features of the final application.

The main target users of the application in its initial stage are students, freelancers and teachers. The application tried to address the needs of all these target customers by trying to avoid conflicting features that might make a benefit for one - a significant drawback for the other. In order to describe the outcome of the developed application and its limitations, taking users perspective to view the application might be a good approach. This approach will best explain results and drawbacks of the application by describing which part of the application was intended to satisfy the need of which kind of users.

Student’s Perspective

Students were the main target of this project. The first fundamental purpose this application intended to give this target group was to provide a simple medium to track and register project and assignment durations for personal use. From this data, a valuable information can be collected that will enable students to assess themselves and realize their strengths and weaknesses in different topics or subjects of their studies in particular, and their ability to predict their planning potentials for their projects and assignments in general.

To this extent, the demo application that was developed was successful in providing a simple medium for such a purpose. It provided an application students can register to

in easy steps, create project timing entries and get useful numerical and graphical performance analysis for their projects. Limitation in this regard is mainly the lack of additional features such as project subdivision into tasks which shall be included in the final application.

The other medium this application intended to offer was a collaboration environment for students' group work. Students are often assigned projects as a group where they have to divide tasks and contribute to the overall project by communicating with each other. In regard to this purpose, the application provides a medium where students can invite each other and access the same profile by registering. A group leader can be the one to create the account first and prepare project entries for each member, label it with their name and invite team members. Team members can then access this account and record their time logs on the project entry created for them and update their status in the description field for all the others to see. Each member will be able to see the progress and performance of each other. So the application was successful to create a medium several people can access and view.

One of the limitations in this perspective is the fact that project time logs can be edited by all members. There should be a mechanism to restrict the modifying rights of each member to their assigned project entry and allow only reading rights for projects entries assigned to other team members. The other limitation is the communication medium. The application can benefit from a better communication medium for team members where for example a live chat or something similar can be constructed. An update posting medium triggered by any change in information in each project entry can also offer a better communication.

Teacher's Perspective

Teachers can also use this application to assign group projects or individual projects for their students and assign a task for them. One ideal application it has in this regard is to give students and their thesis advisor a medium. A thesis advisor can register in the application and invite the student or vice versa after choosing a topic and agreeing on schedule. They can design the time schedule on the application together dividing the thesis into parts that should be done in time. The student can then log in to the application and register his progress which the teacher can check and encourage or advise him as necessary. The advisor can see the students thesis information like update

times and status updates which can tell him how efficiently the student is working and where he is in the process. In addition he can know how dedicated the student is and how much progress he has. This can also be replicated to teachers managing a team of students on a project. The limitation of this case are also similar to the one mentioned above in the students group view.

Freelancer's Perspective

Freelancers were also target clients for this project and some features were added for them. The inclusion of monetary fields and calculations were introduced mainly for this target group. The purpose of this application for this target group can be similar to solo students with the addition of payment considerations. An option to choose billable project time logging is included in the application to serve both free and payable project registrations. This application will provide a medium for freelancers to record their performances on their projects and their financial advantage difference. The decisions this group will make could be financial based as opposed to students and basic fields for that consideration were successfully included. The limitations for this target group could be lack of additional features that they can benefit from. One such feature can be the presence of a downloadable invoice (pdf for example) with optionally selectable fields to include that they can send to their clients.

It should be noted that the group view features will not affect users that intend to use the application for personal use such as single students or freelancers. The group view functionality will only be activated by the invitation of the account creator and not using this feature will provide enough guarantee for the personal use of the application. The objective of the application is to provide a medium for information gathering and analysis that users can use to guide themselves. How they will intend to use their information or features of the application will be largely up to the users.

There will not be a strict usage rule for any type of users. Some freelancers, for example, might want to keep their information on their actual working hours private from their clients and charge clients based on the initial estimated hour element. Other freelancers might want to use the application for a more transparent interaction with their clients and might want to invite them to the account and let them track the progress and the actual time the project is taking. Therefore, how the application will be used will be the decision of the users. Additionally, there will not be a right or wrong way on how the

users should use the result of their data. For example a low efficiency in java programming projects and a high efficiency in C# programming projects could make a student study more java than C# to balance his/her knowledge. On the other hand the same scenario for a freelancer could mean changing direction to favour C# projects than java for a more time efficient income.

5.2 Business Model

The application in the current scope will not have sufficient features to charge users; therefore the short term plan of the application will not consider a financial income generation. However, the long term strategy of the application makes revenue generation a main objective for the successful scaling up and improvement of it. For such a vision to materialize; there should be a clear plan to implement and set of goals to attain.

The first stage in making this application a successful business will be the development of a production level application. This will involve including features simplified out from this version that are mentioned in the limitations above and adding more flexibility, accuracy, user friendliness, and aesthetics. The resulting application from these improvements will be a trial version. This free test version will then be launched and promoted around schools in order to get a customer base from which a valuable data and feedback will be collected. This data and feedback will then be used to add additional features or make modifications based on users' interests.

The success of the application as a business will significantly depend on the successes of the test versions. Hence, the process of collection of usage data and feedback and redesigning the application by using them could be done a number of times. A good indicator to show the readiness of the application for income generation will be its users count and their attachment to the application. A large count of registration of users does not necessarily show a large usage of the application and so the number of returning users should be considered. A higher number of returning users will indicate a higher attachment between the application and its users.

When this point is reached, development of smart phone apps of the application should be designed. This kind of application will, more or less, rely on a subscription based business model. However, there is often a great danger in the introduction of subscrip-

tion payments in a previously free application because it might lead to customer loss. Therefore, careful considerations should be made on these transitions and preferably the subscription additions should not involve taking features out of the free version but rather adding new features for subscriptions. One such addition could be smart phone integration.

Currently, project management applications that are well established are already in the market. However their main targets are employees in the work environment and students are usually neglected. That is the main reason this project concentrated on students and appealing to students. The application can capitalize on wide customer base with very low subscription charge to target students. Appealing to students and increasing productivity and efficiency of their performance will also appeal to their schools. That might in turn open a door for school subscriptions if the application can demonstrate its productivity effectively.

6 Conclusion

The aim of the project was to build a simple demo Ruby on Rails application called Metprotracker for an easy project timing, recording and analysis. While being simple, it planned to give functional features to users that would enable them to make use of the data they collect and make reasonable decisions in order to improve their productivity and efficiency. An appealing and explanatory presentation mechanism of data to the user was supposed to be given a significant attention. In addition, a functionality for sharing an account for two or more users was to be provided in order to give a collaboration option to students-students, teachers-students, and clients-freelancers relationship. The main customer bases it intended to target were students, teachers and freelancers and cautious approaches were to be taken in development to avoid satisfying the requirements of one group at the expense of the other.

As a result a demo application called Metprotracker was successfully developed based on the Ruby on Rails framework. This paved a road for the future production of a more complete version of the application by setting a foundation for its design and development while at the same time demonstrating the advantages and practicality of the application. The final product provided a mechanism for its target customer base to manage and time projects, view the results of this process, and to get valuable information from it that could improve productivity, efficiency and accuracy in time management.

The limitations this version faces are mainly from the planned simplifications done in order to reduce its details and magnify its principal purpose. These limitations mainly manifest themselves in the form of feature simplifications and reductions which can be amended in the complete version. It can thus be concluded that this application achieved the main goals it set out to attain in showing the productivity of having such an application and made developing an advanced version worthwhile.

References

1 A guide to the project management body of knowledge. Newtown Square, PA: Project Management Institute Inc.; 2000.

2 Jack R. Meredith and Samuel J. Mantel Jr., Project Management: A Managerial Approach. Hoboken, NJ: Wiley; 2006.

3 Amy Wohl, Succeeding at SaaS : Computing in the Cloud. Bohemia, NY: Wohl Associates, Inc.; 2008.

4 Justin Williams, Rails Solutions: Ruby on Rails Made Easy. New York, NY: Springer-Verlag, Inc.; 2007.

5 Ruby on Rails Community, Getting Started with Rails [online]. Creative Commons Attribution-Share Alike 3.0 Licence, MIT Expat Licence, Ruby Licence. URL:http://guides.rubyonrails.org/v4.1.8/getting_started.html. Accessed 15 September 2014.

6 David Flanagan, and Yukihiro Matsumoto, The Ruby Programming Language. Sebastopol, CA: O'Reilly Media, Inc.; 2008.

7 Sandi Mets, Practical Object-Oriented Design in Ruby: An Agile Primer. Crawfordsville, US-IN: Pearson Education, Inc.; 2013.

8 Peter Cooper, Beginning Ruby: From Novice to Professional, Second Edition. New York, NY: Springer-Verlag, Inc.; 2009.

9 Heather Moore Niver, Getting to Know Ruby. New York, NY: The Rosen Publishing Group; 2014.

10 Obie Fernandez, Kevin Faustino, and Vitaly Kushner, The Rails 4 Way. Vancouver, BC, Canada: LeanPub; 2014.

11 Michael Hartl, Ruby on Rails Tutorial: Learn Web Development with Rails. Boston, MA: Addison-Wesley; 2012.

12 Regina Obe and Leo Hsu, PostgreSQL: Up and Running. Sebastopol, CA: O'Reilly Media, Inc.; 2012.

13 Frederick Chong and Gianpaolo Carraro, Architecture Strategies for Catching the Long Tail
[online]. United States: Microsoft Corporation; April 2006.
URL:https://msdn.microsoft.com/en-us/library/aa479069.aspx. Accessed 20 October 2014.

14 Frederick Chong, Gianpaolo Carraro, and Roger Wolter, Multi-Tenant Data Architecture [online]. United States: Microsoft Corporation; June 2006.
URL:https://msdn.microsoft.com/en-us/library/aa479086.aspx#mlttntda_topic1. Accessed 24 October 2014.

15 Plataformatec. Devise [online]. São Paulo - SP, Brazil: Plataformatec Technologies; 2015.
URL: http://devise.plataformatec.com.br/#. Accessed 15 February 2015.

16 Hafiz, Nia Mutiara, and Giovanni Sakti, Learning Devise for Rails. Birmingham, UK: Packt Publishing Ltd.; 2013.

17 Dave Thomas, Chad Fowler, and Andy Hunt, Programming Ruby 1.9 & 2.0: The Pragmatic Programmers' Guide. United States: The Pragmatic Programmers, LLC.; 2013.

18 Huw Collingbourne, The Little Book of Ruby, Second Edition. Devon, UK: Dark Neon Ltd.; 2008.

19 David Berube, Practical Reporting with Ruby and Rails. New York, NY: Apress Media LLC.; 2008.

20 The PostgreSQL Global Development Group. PostgreSQL 9.3.6 Documentation [online]. The PostgreSQL Global Development Group under PostgreSQL License.
URL: http://www.postgresql.org/docs/9.3/static/index.html. Accessed 5 February 2015.

21 Ruby on Rails Community, Active Record Basics [online]. Creative Commons Attribution-Share Alike 3.0 Licence, MIT Expat Licence, Ruby Licence.
URL: http://guides.rubyonrails.org/v4.1.8/active_record_basics.html. Accessed 21 February 2015.

22 Ruby on Rails Community, Active Record Migrations [online]. Creative Commons Attribution-Share Alike 3.0 Licence, MIT Expat Licence, Ruby Licence.
URL: http://guides.rubyonrails.org/v4.1.8/migrations.html. Accessed 22 February 2015.

23 Ruby on Rails Community, Active Record Associations [online]. Creative Commons Attribution-Share Alike 3.0 Licence, MIT Expat Licence, Ruby Licence.
URL: http://guides.rubyonrails.org/v4.1.8/association_basics.html. Accessed 1 March 2015.

24 Mark Otto and Jacob Thornton, Bootstrap [online]. Twitter, Inc. under MIT Licence.
URL: http://getbootstrap.com/. Accessed 15 January 2015.

25 Ryan Brunner and Brad Robertson, Apartment [online]. RubyGems, Ruby on Rails Community Gem Hosting, MIT Licence.
URL: https://rubygems.org/gems/apartment. Accessed 18 September 2014.

26 José Valim and Carlos Antônio, Devise [online]. RubyGems, Ruby on Rails Community Gem Hosting, MIT Licence.
URL: https://rubygems.org/gems/devise. Accessed 15 September 2014.

27 Ryan Bates, Letter_opener [online]. RubyGems, Ruby on Rails Community Gem Hosting.
URL: https://rubygems.org/gems/letter_opener. Accessed 15 February 2014.

28 José Valim, Carlos Antônio, and Rafael França, Simple_form [online]. RubyGems, Ruby on Rails Community Gem Hosting, MIT Licence.
URL: https://rubygems.org/gems/simple_form. Accessed 25 September 2014.

29 Thomas McDonald, Bootstrap-sass [online]. RubyGems, Ruby on Rails Community Gem Hosting, MIT Licence.
URL: https://rubygems.org/gems/bootstrap-sass. Accessed 25 September 2014.

30 Michael Granger and Lars Kanis, Pg [online]. RubyGems, Ruby on Rails Community Gem Hosting, BSD Licence, RUBY Licence, GPL Licence.
URL: https://rubygems.org/gems/pg. Accessed 25 September 2014.

31 Sergio Cambra, Divise-invitable [online]. RubyGems, Ruby on Rails Community Gem Hosting, MIT Licence.
URL: https://rubygems.org/gems/devise_invitable. Accessed 25 October 2014.

Appendix 1: Metprotracker Gemfile

```
# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.1.6'
gem 'pg'

gem 'sass-rails', '~> 4.0.3'
gem 'uglifier', '>= 1.3.0'
gem 'coffee-rails', '~> 4.0.0'
gem 'jquery-rails'
gem 'tzinfo-data'

gem 'devise', '~> 3.2.0'
gem 'devise_invitable', '~> 1.3.4'

gem 'apartment'

gem 'bootstrap-sass', '~> 3.2.0'
gem 'autoprefixer-rails'

gem 'simple_form'

group :development, :test do
    gem 'guard'
    gem 'guard-livereload'
    gem 'guard-rspec'
    gem 'rspec-rails'
    gem 'capybara'
    gem 'factory_girl_rails'
    gem 'database_cleaner'
    gem 'shoulda-matchers'
    gem 'letter_opener'
    gem 'email_spec'
end
```

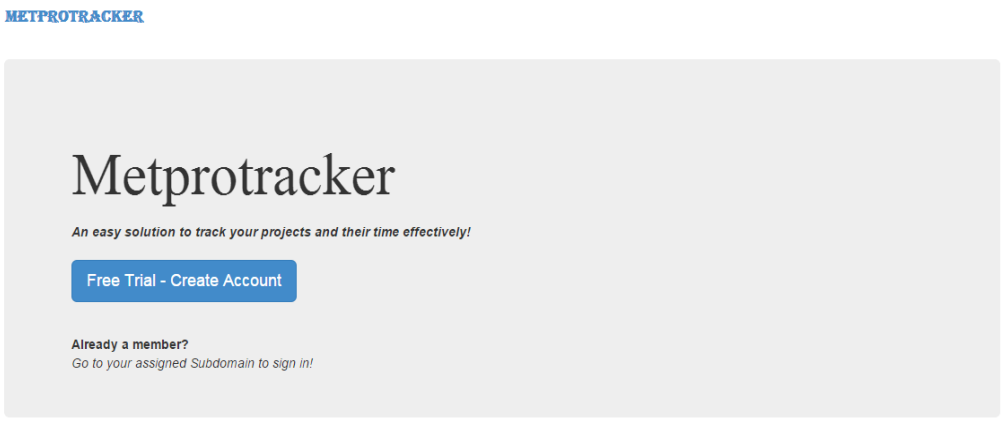
Appendix 2: Routes of Metprotracker

```
C:\Sites\metprotracker>rake routes
                  Prefix Verb   URI Pattern                                Controller#Action
          subdomain_root GET    /                                          projects#index
        new_user_session GET    /users/sign_in(.:format)                   devise/sessions#new
            user_session POST   /users/sign_in(.:format)                   devise/sessions#create
    destroy_user_session DELETE /users/sign_out(.:format)                  devise/sessions#destroy
           user_password POST   /users/password(.:format)                  devise/passwords#create
       new_user_password GET    /users/password/new(.:format)              devise/passwords#new
      edit_user_password GET    /users/password/edit(.:format)             devise/passwords#edit
                         PATCH  /users/password(.:format)                  devise/passwords#update
                         PUT    /users/password(.:format)                  devise/passwords#update
accept_user_invitation GET    /users/invitation/accept(.:format) devise/invitations#edit
remove_user_invitation GET    /users/invitation/remove(.:format) devise/invitations#destroy
         user_invitation POST   /users/invitation(.:format)                devise/invitations#create
     new_user_invitation GET    /users/invitation/new(.:format)            devise/invitations#new
                         PATCH  /users/invitation(.:format)                devise/invitations#update
                         PUT    /users/invitation(.:format)                devise/invitations#update
                   users GET    /users(.:format)                           users#index
                projects GET    /projects(.:format)                        projects#index
                         POST   /projects(.:format)                        projects#create
             new_project GET    /projects/new(.:format)                    projects#new
            edit_project GET    /projects/:id/edit(.:format)               projects#edit
                 project GET    /projects/:id(.:format)                    projects#show
                         PATCH  /projects/:id(.:format)                    projects#update
                         PUT    /projects/:id(.:format)                    projects#update
                         DELETE /projects/:id(.:format)                    projects#destroy
                 reports GET    /reports(.:format)                         reports#index
                         POST   /reports(.:format)                         reports#create
              new_report GET    /reports/new(.:format)                     reports#new
             edit_report GET    /reports/:id/edit(.:format)                reports#edit
                  report GET    /reports/:id(.:format)                     reports#show
                         PATCH  /reports/:id(.:format)                     reports#update
                         PUT    /reports/:id(.:format)                     reports#update
                         DELETE /reports/:id(.:format)                     reports#destroy
                    root GET    /                                          welcome#index
                accounts POST   /accounts(.:format)                        accounts#create
             new_account GET    /accounts/new(.:format)                    accounts#new
```

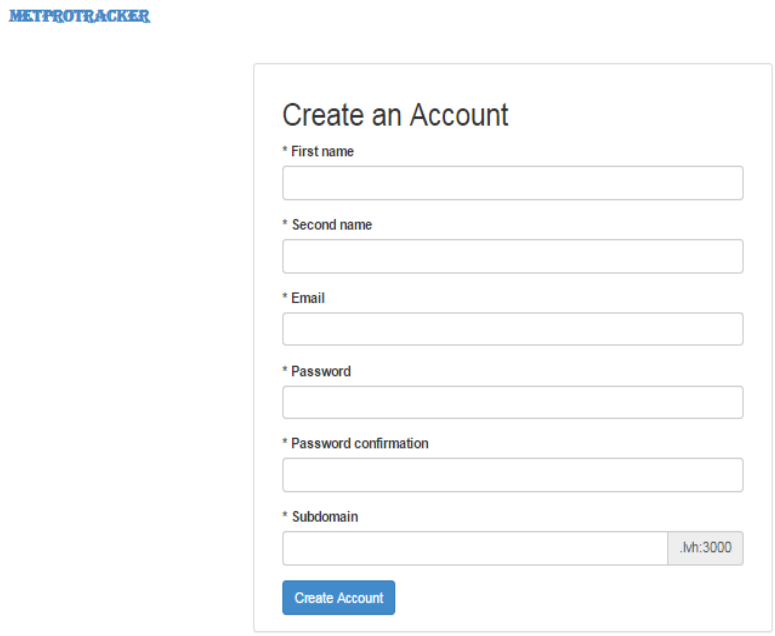
Appendix 3: Users Table and its Attributes

Users	
id	int
first_name	varchar(%)
second_name	varchar(%)
email	varchar(%)
encrypted_password	varchar(%)
reset_password_token	varchar(%)
reset_password_sent_at	timestamp
remember_created_at	timestamp
created_at	timestamp
updated_at	timestamp
invitation_token	varchar(%)
invitation_created_at	timestamp
invitation_sent_at	timestamp
invitation_accepted_at	timestamp
invitation_limit	timestamp
invited_by_id	int
invited_by_type	varchar(%)
invitations_count	int

Appendix 4: Welcoming Page of Metprotracker

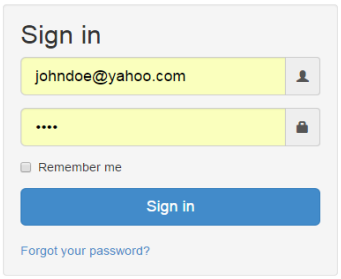


Appendix 5: Registration Page

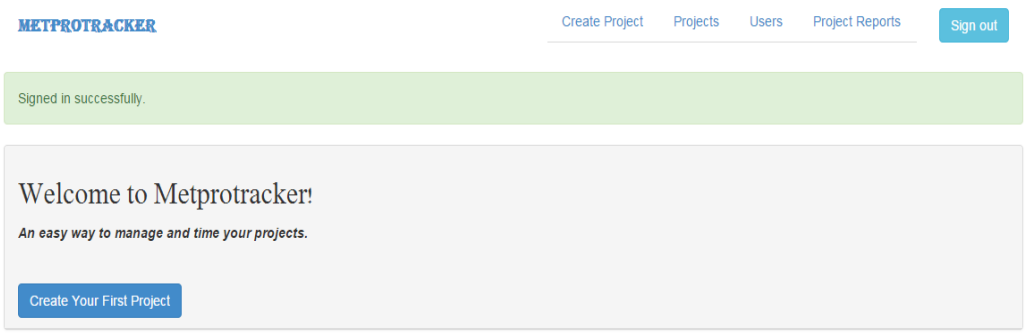


Appendix 6: Users ‘sign in’ Page

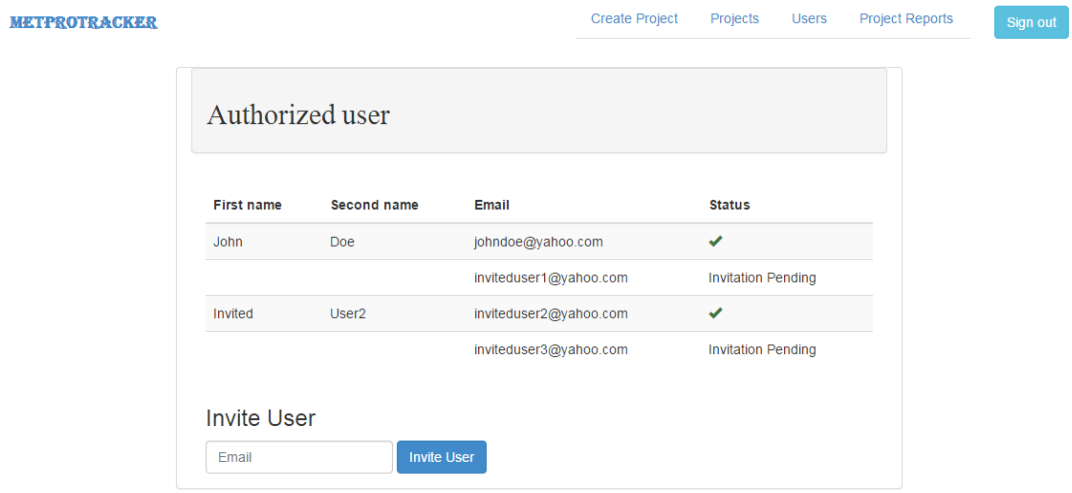
METPROTRACKER



Appendix 7: New Users Welcoming Page

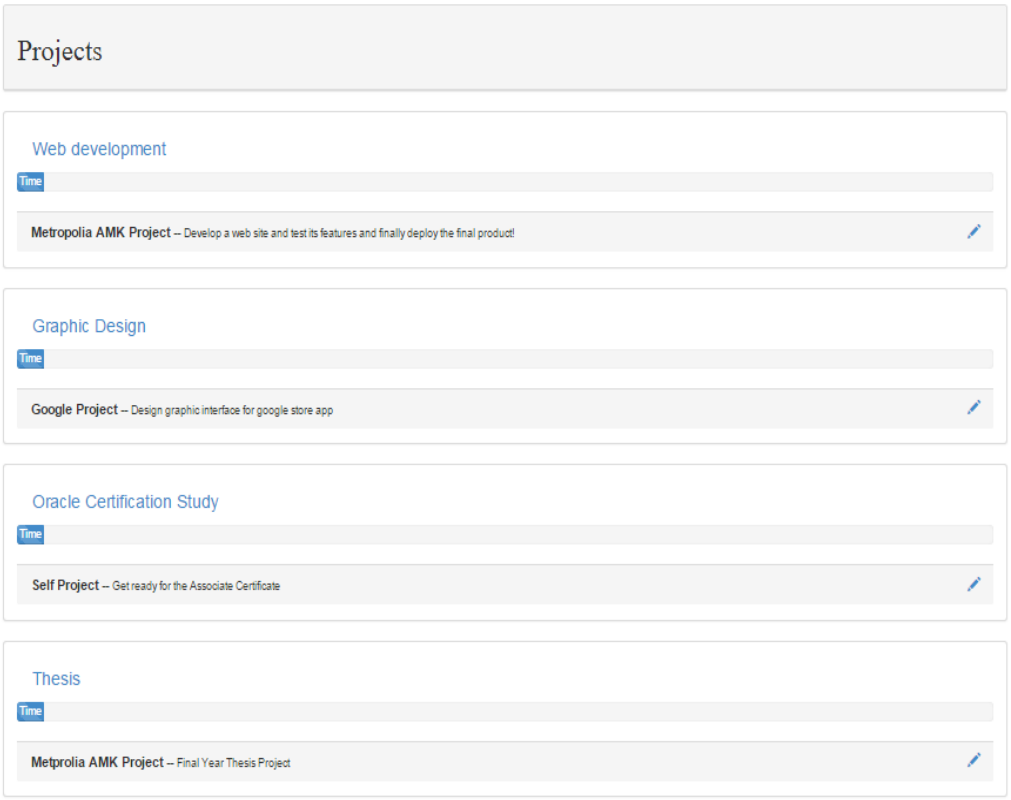


Appendix 8: Users List View



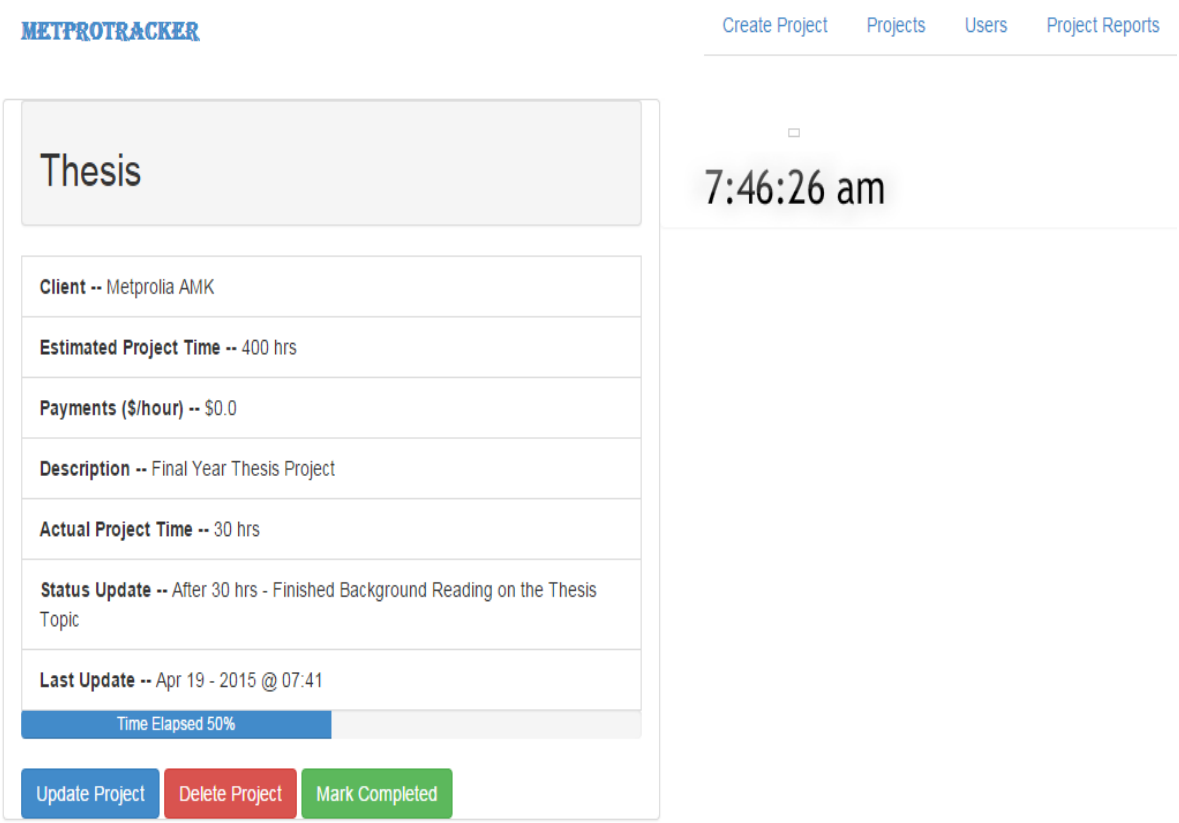
Shows the users that are invited and have accepted the invitation labelling it as '√' on the status column. Users invited but that have not yet accepted it are shown as 'invitation pending' on status column. Invitation option is at the bottom with a field to fill email of the person to be invited.

Appendix 9: Projects View Page



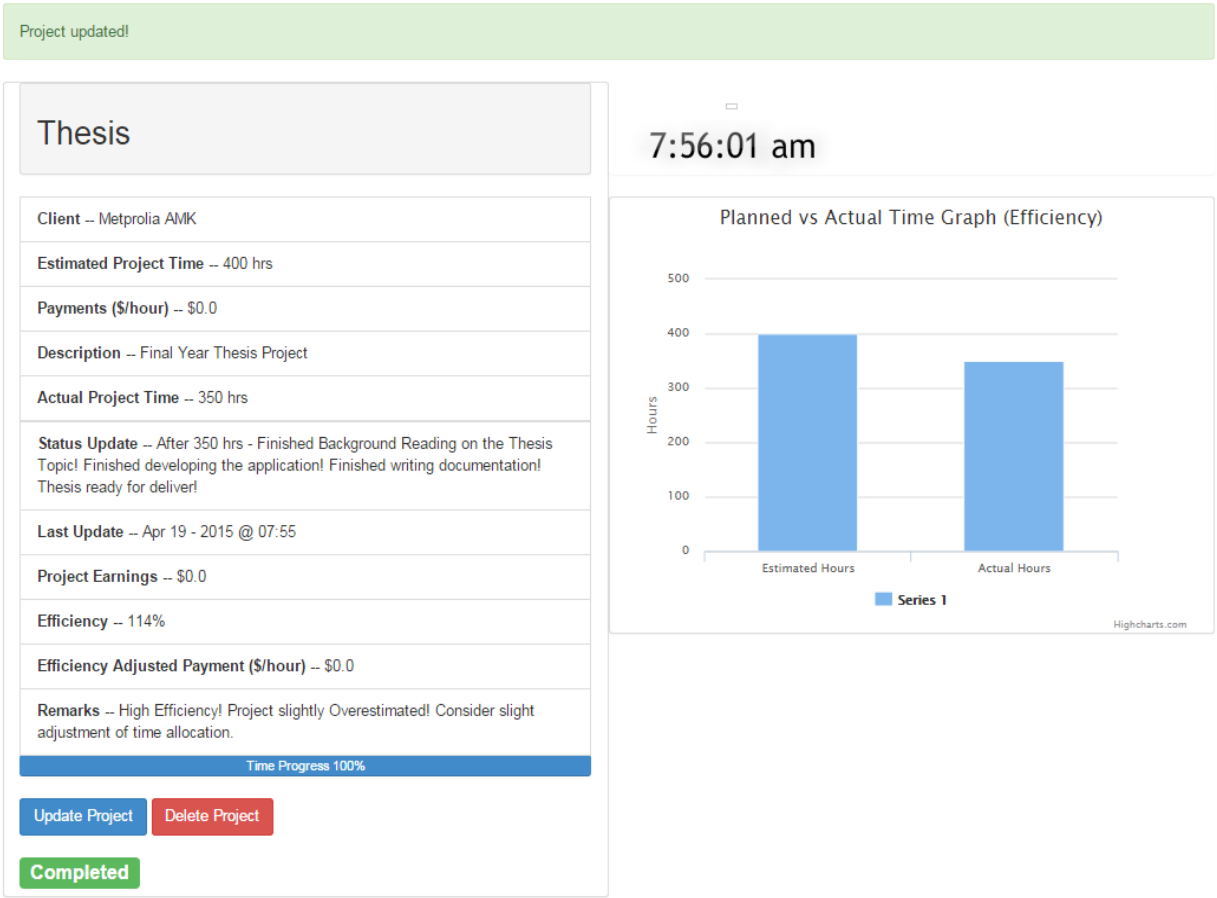
Appendix 9 shows an example registered projects of a user.

Appendix 10: Single Pending-Project View



Shows an example view of a student working on a thesis after 30 hours. Teacher can view this page. Fields like actual time spent so far, last update time (activity of student) can be important for the teacher to view and comment on time management of the student.

Appendix 11: Single Completed-Project View



An example project view of a student who finished thesis.

Appendix 12: Reports View of All Projects of a User

Project ID	Project Title	Client	Estimated Hours	Actual Hours	Efficiency	Payment ($/hour)	Earnings
1	Web development	Metropolis AMK	60	45	133%	10.0	$600.0
3	Graphic Design	Google	33	27	122%	15.0	$495.0
2	Oracle Certification Study	Self	50	17	294%	0.0	$0.0
5	Thesis	Metropolis AMK	400	350	114%	0.0	$0.0