An evaluation of a performance management system using MEAN stack

Paul John Sharrock

Table of Contents

Problem description

Performance management is essential to businesses, as it helps align employees, resources, and systems to meet strategic objectives. Also, it provides an early warning of potential problems and allows managers to make adjustments to keep a business on track. Equally so, ensuring that individuals are performing in line with the overall company strategic objects requires some form of a performance management system.

The client for this project is the HR manager for a software development company, that currently records employee performance on a paper file system, which is cumbersome and cannot be measured. In addition, these records only exist for new employees within their first year of employment; therefore, employees that have been in the company for a more extended period have no records at all. The company is keen to adopt the Objectives and Key Results (OKR) process of performance management and to develop an in-house software solution that will measure these OKRs.

The company is currently using Angular and is also interested in evaluating compatible back-end technologies that could potentially reduce time to market for its products. Owing to the company’s commitments, researching new technologies would detract from employee work. The client’s company subsequently decided that this task could be something an undergraduate could do.

It was, therefore, agreed that this project will seek to design a new software system that will enable OKR performance management. In essence, the proposed system design will allow the company to identify under-performing, performing as expected and upcoming talent, as well as generating reports for a given query by management.

It was further agreed by the client that an evaluation of a MEAN (MongoDB, Express, Angular and Node.js) stack architecture would be conducted. The evaluation would entail the proposed design for the system being developed in a MEAN stack architecture, thus serving as a case study. As well as the pros and cons of adopting a MEAN stack technology elucidated within a concise report. The client agreed to support this project with some caveats from her employer.

These are primarily that the company remains anonymous for the purpose of this project and that there would be no access to any other employees from the company.

Project Goal

The goal for this project is three-fold: to produce a first iteration design for a potential solution for a performance management system; to evaluate the use of a MEAN stack as a technology; to bridge the design process to a developed solution using a prototype within a MEAN stack architecture.

To achieve the aforementioned goals the end product of this project consists of a Volere template, in which all design outputs are captured, ergo; domain modelling, requirements, and analysis generated during the engineering process. Which the client's company could take forward, refine and develop in house.

A prototype will also be made available to the client, which will seek to bridge the gap between the proposed design and how it could be implemented within a MEAN stack, as well as provide evidence of how this was achieved, which would serve the client as a case study.

In addition, the end product consists of a written report evaluating a MEAN stack, which has been appended to the Volere template (annexe A para 11) as it allows for the reader to flow from the design process through to the evaluation of a MEAN stack.

Analysis of the likely impact of the project

By developing an embryonic solution using MEAN stack as a technology, has allowed for this project to serve as both a practical evaluation of said technologies and a potential solution to adopting a performance management system.

Since the client for this project is a software development company, the processes used to design the potential solution are likely what they would find most useful in contrast to a non-software orientated client, whom would only benefit from an end product.

Communication with the client has been constant throughout the process of this project. The client has conveyed that the client's company will most likely select elements from the design process which they can take forward, refine and develop. For instance, should they decide not to adopt a MEAN stack architecture, the Volere template provided could serve as the basis for a developed solution in an architecture of their choice.

Relative to this is the fact that in documenting the development of a prototype for the proposed solution, other aspects for the client company to consider have become apparent. For instance, the use of traditional design models does not lend itself naturally to developing a solution within a MEAN stack technology, therefore, adopting a MEAN stack technology could result in how they design applications being changed.

With regard to the evaluation, the findings articulated within the evaluation will factor into the client’s company's decision making process on deciding what technologies to adopt in the future.

Overall, the end product of this project has been shaped by constant communication with the client and its three-fold approach is intended to bring all stakeholders responsible for adopting new technologies and developing new products together.

Analysis of the problem

Related literature

Owing to the evaluative nature of this project a research phase was incorporated into each phase of the selected development cycle. From the onset, it was realised that the credibility of this project is directly proportional to the creditability of the sources researched.

Therefore, in order to garner their credibility, the Open University’s PROMPT technique was used. The initial intent was to prioritise peer reviewed literature over other sources. However, as the project progressed it became apparent that such types of literature were less common than books written by selected authors.

Unlike papers, books are generally not peer reviewed. So in order to ascertain their credibility, the authors were researched. Areas that were focussed on were the achievements and qualifications of said authors and the overall creditability of the publishers (choosing predominately reputable names within the computing field, such as Packt and O'Reilly). In addition, multiple books were chosen for a select subject. Which allowed for their content to be corroborated with one another.

Sources that did not fall under the category of books were organisational sites, which proved easier to evaluate using the PROMPT methodology. Such as MDN Web Docs from Mozilla that cover a large amount of web technologies as well as white papers from organisational sites who specialise in Performance Management.

The analysis

The first step to approaching this problem was to interview the client in order to ascertain the problem statement and subsequently, the requirements of the system. Moore J.M, Shipman F.M (2000), highlight and compare a number of different methods for requirements elicitation. One key factor they point out is that usually a client has a holistic abstract idea as to what the system should do, but they find it challenging to articulate specific requirements, which proved to be true in this case.

The problem, as mentioned above, was further compounded by the fact that there was no access to other employees within the client's company; meaning that the use of questionnaires, and snow cards, as described within the Volere process (Volere.org, nd), could not be used. Conscious of the mantra that requirement elicitation is the most crucial part of developing a software solution, this situation was a risk to the validation of the proposed solution.

Formulating a questionnaire to guide the client was considered; however, Moore J.M, Shipman F.M (2000) also highlight that questionnaires lack the nuances of human interaction. Therefore, likely resulting in the time allocated to this phase over running owing to further clarifications.

Young (2003) articulated a method in which business scenarios are discussed with the client and requirements elicited from the discussions. Figs 1 & 2 illustrate the results of discussing business processes with the client which were captured and recorded as business process models.

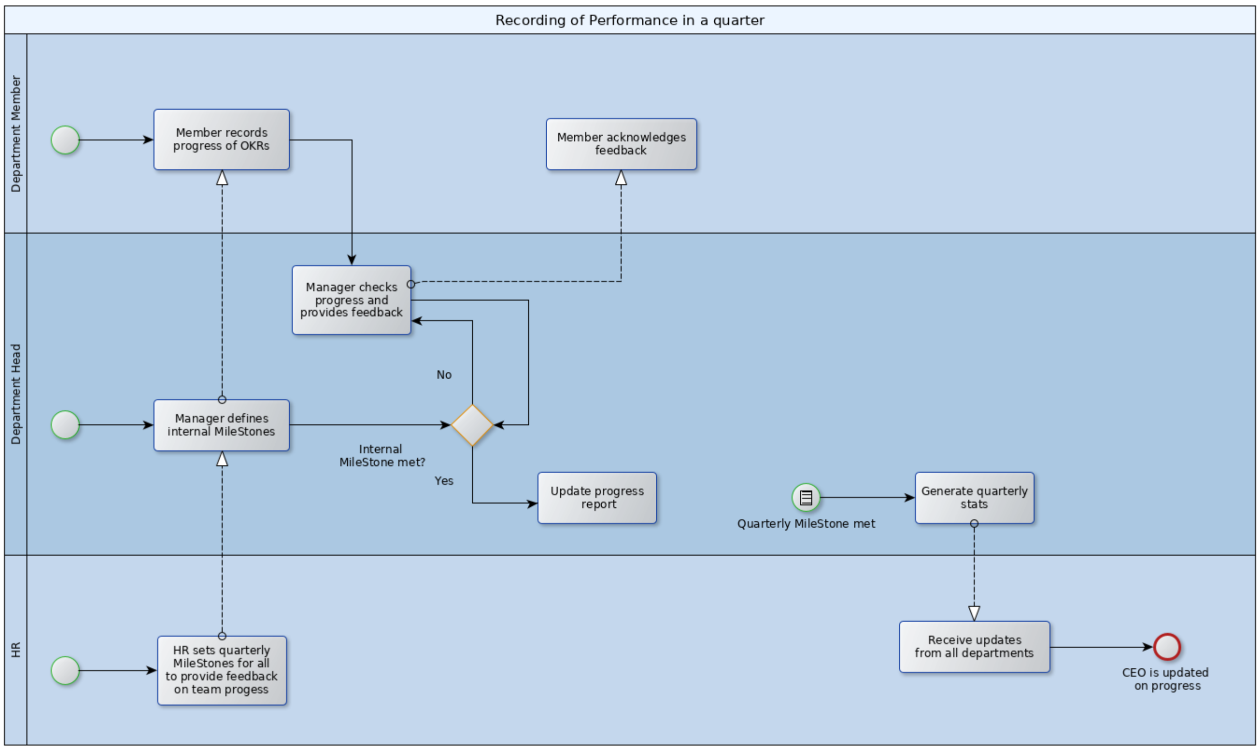

Recording of Performance in a quarter
Department Member
Department Head
HR
Member records progress of OKRs
Member acknowledges feedback
Manager checks progress and provides feedback
No
Manager defines internal MileStones
Internal MileStone met?
Yes
Update progress report
Quarterly MileStone met
Generate quarterly stats
HR sets quarterly MileStones for all to provide feedback on team progess
Receive updates from all departments
CEO is updated on progress

Fig 1

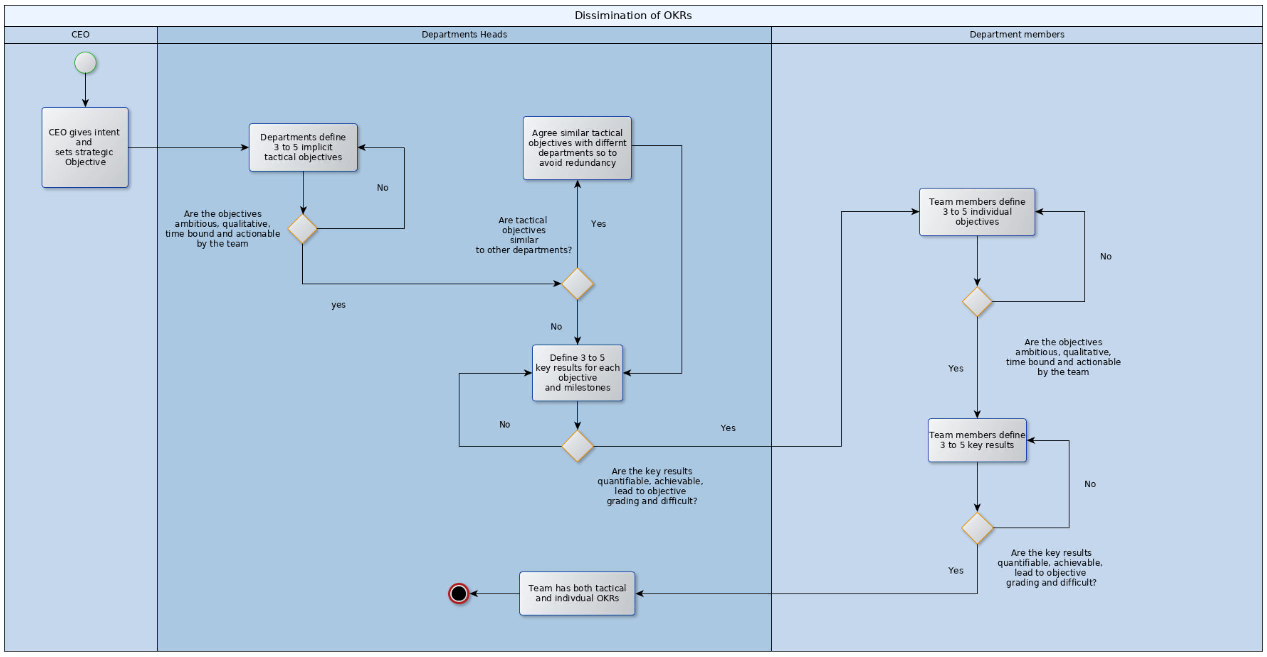

Dissimination of OKRs
CEO
Departments Heads
Department members
CEO gives intent and sets strategic Objective
Departments define 3 to 5 implicit tactical objectives
No
Are the objectives ambitious, qualitative, time bound and actionable by the team
yes
Agree similar tactical objectives with differnt departments so to avoid redundancy
Are tactical objectives similar to other departments?
Yes
No
Define 3 to 5 key results for each objective and milestones
No
Yes
Are the key results quantifiable, achievable, lead to objective grading and difficult?
Team has both tactical and indivdual OKRs
Team members define 3 to 5 individual objectives
No
Are the objectives ambitious, qualitative, time bound and actionable by the team
Yes
Team members define 3 to 5 key results
No
Are the key results quantifiable, achievable, lead to objective grading and difficult?
Yes

Fig 2

The method was taken further, by drawing on TM352 (Web Development) a paper mock-up of the interface was created. In this case, it was not to develop the interface but to allow the client to role play the business processes, which resulted in business use cases being drafted and later refined. This method proved to be highly effective, with the output being recorded as work partitioning within the Volere template (annexe A, para 6c).

Naturally, potential actors were decided upon for this to take place. Which allowed for an initial use case model to be produced concurrently (Fig 3). The analysis of which resulted in the initial requirements for the solution.

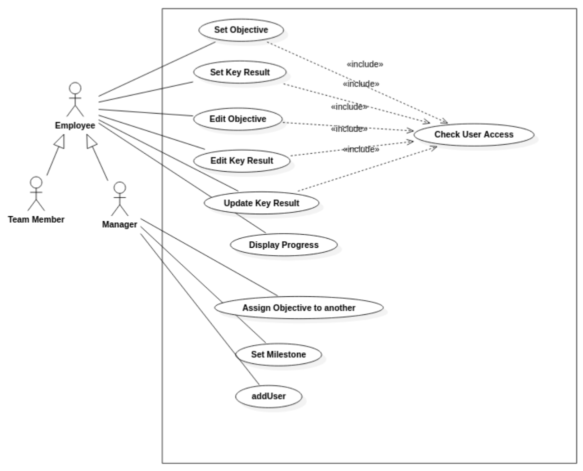


Fig 3

In order to mitigate against face-to-face interactions resulting in the developer leading the client towards a pre-determined biased design (Moore J.M, Shipman F.M, 2000), both the business processes and the use case diagram were then used to validate the requirements with the client. This approach was done in a similar manner, by "walking" through the process and linking the business event, with the use case and finally the requirements.

The initial list of requirements that were captured was recorded within the Volere template (annexe A). The template guidance (Volere.org, nd, a) and the process of populating it had proven to be both practical and efficient. It allowed for key outputs for each phase to be captured and served as a checklist of the work that needed to be done.

The practicality of the Volere template became apparent when populating the first five paragraphs of the template itself, which are dedicated towards business processes. The level of abstraction of the business process models was appropriate for requirements elicitation. However, in order for the context of the problem to be elucidated, further research was required.

The organizational site, Evolution4all (2018), contained a white paper in which the business processes were explained in detail, more specifically, the Objectives and Key Results (OKR) process. This understanding then allowed for focussed questions to be asked of the client. More specifically, understanding the OKR process led to the realisation that objectives and key results were likely candidates for classes and more importantly, that they would have a one-to-many composite relationship.

This understanding allowed for elaborate use cases to be documented within the Volere template (annexe A, para 8b) and a more detailed set of functional requirements were produced and recorded (annexe A, para 9), with them being linked to the elaborate use cases.

Moving towards a solution, as articulated, the client's company is keen to explore new technologies, particularly ones that could benefit them by reducing time to market. Therefore, architectural decisions had to be made earlier on in the development cycle.

Initial analysis for a potential architecture for this project took into consideration a system that allows for data to be retrieved from a database, allows users to interact with it, then returns the data to the database.

Research into architectures and Angular began to steer the project towards a MEAN stack. Holmes, S (2015) describes a MEAN stack as an architecture that brings together the best of the best and allows for a full system to be developed in the same language. Also, with consideration to the *'ilities'* of the system. Having the client and server side of the architecture written in JavaScript meant that the maintainability of such a system would be easily achieved.

Architectural decisions are paramount to how a system will operate and deliver its functionality. Ultimately, the design of the application logic would be dependant on the architectural decision made. One of the concerns for this project was that the OU module text on software engineering, although initially agnostic to a programming language, was geared toward Java and classic Object Orientated Programming.

However, as highlighted by MDN Web Docs (2019) both Node.js and Express are unopinionated frameworks, meaning it has far fewer restrictions on the best way to interface components together to achieve a goal, or even what components should be used; this, therefore, meant that case studies varied in where components resided within their architectural designs, ergo a logical view of a solution.

In addition, Node.js and Express use JavaScript. Although JavaScript allows for some aspects of Object Orientated Programming, following recent updates (ECMA-international, n.d), it still lacks some of the functionalities and differed in how it implemented those that it did facilitate, such as encapsulation.

Working towards a solution

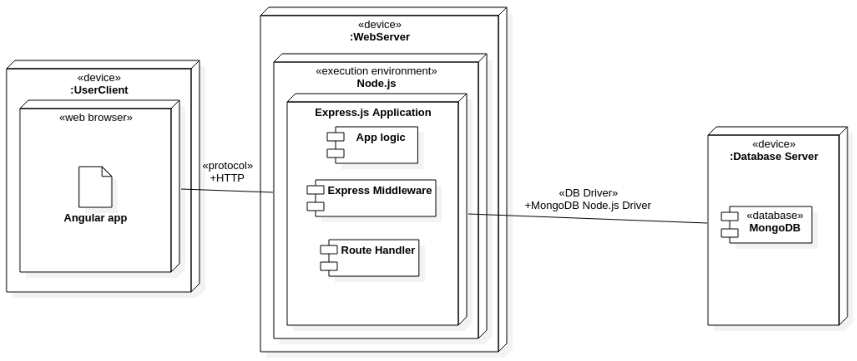


Fig 3

Using Tsonev, K. (2015) as a case study, the deployment view of the proposed solution was created (Fig 3), which also illustrates the architecture that shall be used in developing this solution.

Focussing on the application's logic, Tsonev, K. (2014) highlights three significant components within the Express application. The App logic where the main business functions will reside, a route handler to listen for traffic from both the back and front ends and contains a class that can handle and pass requests, and finally the middleware.

The company website of Apprenda (2019), a cloud platform provider and a corroborating case study, go into detail of the role and functionality of Middleware. Implementing Middleware is an architectural style in which standard functions are encapsulated within modules which are independent of one another; therefore, allowing for modules to be added or removed depending on the function required, without breaking the application as a whole.

Middleware modules are off the shelf solutions which use a call-return architectural style to interface with one another. In the proposed architecture (Fig 3), the middleware will be responsible amongst other things for sending and receiving data to and from the Angular app using REST, handling authentication and errors, session and cache management, as well as interfacing with the database. The use of Middleware would, therefore, reduce the time needed to develop applications and would benefit the client's company in reducing the time to market for its products.

Having decided on an architecture for the application the design of the applications logic could begin, Fig 4 illustrates the domain structural model, with proposed attributes.

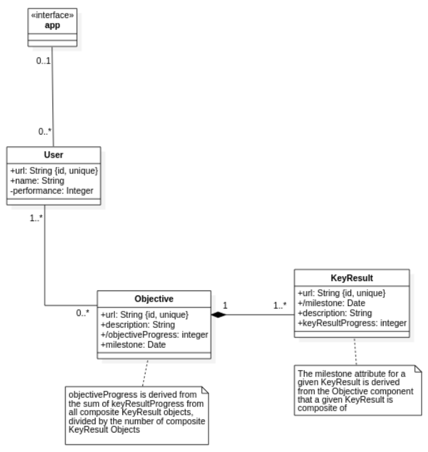

«interface»
app
0..1
0..*
User
+url: String {id, unique}
+name: String
-performance: Integer
1..*
0..*
Objective
+url: String {id, unique}
+description: String
+/objectiveProgress: integer
+milestone: Date
1
1..*
KeyResult
+url: String {id, unique}
+/milestone: Date
+description: String
+keyResultProgress: integer
objectiveProgress is derived from the sum of keyResultProgress from all composite KeyResult objects, divided by the number of composite KeyResult Objects
The milestone attribute for a given KeyResult is derived from the Objective component that a given KeyResult is composite of

Fig 4

Source Making (2019), a website dedicated to software architecture, highlight what they describe as "Software Architectural Antipatterns". Antipatterns are described as common bad design choices made by developers. One of the antipatterns they describe is called "Jumble", which describes an architecture with horizontal and vertical elements intermixed, resulting in an unstable architecture.

As alluded to, both Node.js and Express are unopinionated frameworks, and a MEAN stack as a whole is compatible with one another. This, coupled with its use of a middleware architectural style, suggests that developing a solution with MEAN could fall foul to the antipattern as mentioned above.

It was, therefore, decided that the application logic would make use of some form of singleton interface; therefore, ensuring that the logic developed is modular and interdependent of both the user interface as well as any future middleware used to implement the necessary services.

One such interface design pattern that could be utilised is the Facade pattern (Gamma, 1995). However, although this addresses the interdependency of the application logic, the use of middleware and a cascade approach to messages between classes (more specifically concerning the use of GRASP) could result in the antipattern describe above - more so if the middleware is interfacing the front or back-end of the stack.

Another option described by Gamma (1995) is the Mediator pattern. This pattern allows for the interface object to encapsulate how a set of objects interact with each other. Although this would make the interface more tightly coupled with the objects it mediates with, it does promote loose coupling between those objects. Subsequently, this should allow for middleware to be changed with minimal maintenance to the code, as changes will only need to be made in the mediator class.

The remaining proposed classes were garnered from analysis of the requirements, with their attributes captured within a data dictionary (annexe A, para 7b). The derived attributes: `Objective.objectiveProgress` and `KeyResult.milestone` were decided upon as a result of the aforementioned further analysis of the OKR process. The choice of a URL as an identifier will allow for a MongoDB schema to be created that will enable the persistence of class instances.

The resultant domain structural model allowed for the system operations of the proposed classes to be captured (annexe A, para 10b). Within this increment of the solution, the number of operations garnered was those that facilitated the core features of the solution.

However, as work proceeded on the sequence diagrams for the operations, further required operations became apparent, such as getters and setters that were required for the classes to carry out their functions. Fig 5 Illustrates the first increment of the design structural model, with said helper functions included.

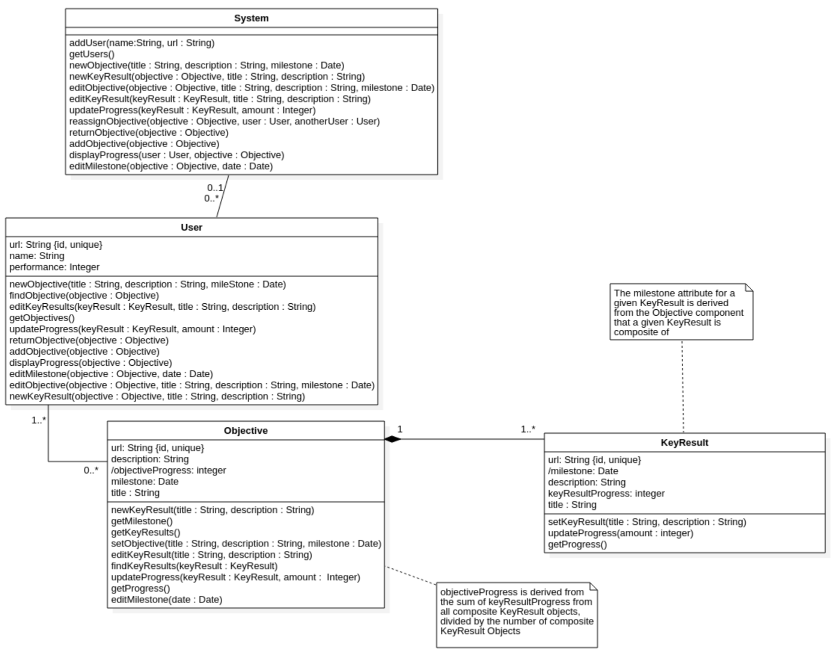

System
addUser(name:String, url : String)
getUsers()
newObjective(title : String, description : String, milestone : Date)
newKeyResult(objective : Objective, title : String, description : String)
editObjective(objective : Objective, title : String, description : String, milestone : Date)
editKeyResult(keyResult : KeyResult, title : String, description : String)
updateProgress(keyResult : KeyResult, amount : Integer)
reassignObjective(objective : Objective, user : User, anotherUser : User)
returnObjective(objective : Objective)
addObjective(objective : Objective)
displayProgress(user : User, objective : Objective)
editMilestone(objective : Objective, date : Date)
0..1
0..*
User
url: String {id, unique}
name: String
performance: Integer
newObjective(title : String, description : String, mileStone : Date)
findObjective(objective : Objective)
editKeyResults(keyResult : KeyResult, title : String, description : String)
getObjectives()
updateProgress(keyResult : KeyResult, amount : Integer)
returnObjective(objective : Objective)
addObjective(objective : Objective)
displayProgress(objective : Objective)
editMilestone(objective : Objective, date : Date)
editObjective(objective : Objective, title : String, description : String, milestone : Date)
newKeyResult(objective : Objective, title : String, description : String)
1..*
0..*
Objective
url: String {id, unique}
description: String
/objectiveProgress: integer
milestone: Date
title : String
newKeyResult(title : String, description : String)
getMilestone()
getKeyResults()
setObjective(title : String, description : String, milestone : Date)
editKeyResult(title : String, description : String)
findKeyResults(keyResult : KeyResult)
updateProgress(keyResult : KeyResult, amount : Integer)
getProgress()
editMilestone(date : Date)
1
1..*
KeyResult
url: String {id, unique}
/milestone: Date
description: String
keyResultProgress: integer
title : String
setKeyResult(title : String, description : String)
updateProgress(amount : integer)
getProgress()
The milestone attribute for a given KeyResult is derived from the Objective component that a given KeyResult is composite of
objectiveProgress is derived from the sum of keyResultProgress from all composite KeyResult objects, divided by the number of composite KeyResult Objects

Fig 5

Development of the MEAN stack solution

In order to bridge the gap between the proposed design and a developed solution within a MEAN stack architecture, a prototype of the solution was created. By doing so also allowed for the code to be tested in the wider context of it executing within a MEAN stack. Initially, the code was to be tested in isolation using test-driven development techniques, ergo with assertions.

However, this would not have served the purpose of evaluating the architecture as a whole. Test-driven development, in essence, focusses upon the logic of code which is analogous to developing narrow and deep. In contrast, it was identified that for this project to meet its objectives, it needed to be developed shallow and wide.

Therefore, three functions were decided upon that would utilise all parts of the architecture. Basically, taking input from a user that is handled by the business logic which returns and displays data from the database to the user, thus completing an event-flow.

Drawing on my knowledge of web, mobile and cloud technologies (TM352) garnered from the OU and extending it with valuable insights from MDN Web Docs (2019), I installed and deployed a Node.js environment locally on my machine.

Installing Node.js on a system automatically installs some of its key features, one being the Node Package Manager (npm). One of the key features of this execution environment that contributes to quicker development of applications is the fact that Node.js uses a modular approach to organising code, with the vast majority of commonly used functionalities being encapsulated within community written modules (referred to as Middleware) – all of which are incorporated into an application by the developer with use of the Node Package Manager.

One key piece of middleware is Express. Express is a web framework that allows for a web application to be structured to handle different http requests at a specific URL. Once Node.js is installed Express can be installed with a simple command within the Command Line Interface.

```
npm install express
```

Express also allows for an application skeleton to be created with one command. Which creates the key folder structure required for an application and generates templates within them.

```
npm install g express-generator
```

With the Express framework installed, it was configured to use Pug as the user interface. The decision to use Pug rather than Angular was deliberate owing to the fact that the client's company already uses Angular and the use of Pug allows for a prototype to be produced within the time frame of this project.

With my knowledge of database management (TM351), I created a MongoDB database using MongoDB's free tier hosting. Finally drawing once again from TM352 and MDN Web Docs (2019), a script to populate the database with sample data was created. With a skeleton architecture in place, I was ready to develop the aforementioned functions.

The main obstacle that was immediately apparent was how to translate the abstract design into a tangible solution within a MEAN stack. This was primarily due to the fact that in contrast to more classical object orientated languages, such as Java, a class as a concept could not be implemented as one encapsulated file.

For the purpose of this evaluation, this was a major obstacle, as being unable to implement the proposed design meant that the evaluation could not progress. A study of other implementations provided by MDN Web Docs (2019 a) and Tsonev, K. (2015) gave some insight into how the solution could be developed.

In essence, within a MEAN stack, it is a Node.js middleware component, Express, that facilitates the logic of a system. The concept of a class within the Express framework is that elements of a class are federated amongst separate files, grouped by function, into modules. With encapsulation and other concepts of object oriented programming being facilitated by the inter module communication within the framework.

It was, therefore, important to highlight how this was achieved, focussing predominately on the inter module communication of the system between the aforementioned functions and elucidating this within the Volere template.

Evaluation of the solution

Introduction

The evaluation of the solution defines a set of criteria that will be used. In exploring these criteria, what the components of a MEAN stack are, and how they operate, shall be articulated. Which will subsequently allow the reader to be in a position to better understand how the proposed solution could be implemented within a MEAN stack architecture.

The selected criteria were placed into two categories: the “ilities” of the system and how the system addresses some of the more common problems faced by development companies.

Looking at the first set of criteria in detail, the following “ilities” were considered.

- Maintainability.
- Scalability.
- Extensibility.
- Security.
- Portability.
- Performance.

Looking at the second set of criteria in detail some of the challenges facing the software industry were considered (CIO, 2019).

- Time to market.
- Required expertise for the system.
- Integration and interfacing issues.
- Acquiring and retaining talent.
- Internal sourcing and outsourcing required for the system.
- Adoption rates.

Some of the latter criteria are intrinsically tied with the "ilities". For instance, time to market is relative to the maintainability, portability, extensibility and scalability of software. The expertise required, and acquiring and retaining talent in relation to the system is relative to nearly all the "ilities" articulated. Integration and interfacing issues are relative to the extensibility of a system. Adoption rates are generally relative to performance as this is the most common complaint from end-users that can be addressed by the system itself.

What is a MEAN stack?

A MEAN stack is an architecture that uses Angular as the front end. Uses Node.js as the server side execution environment, within which Express is the web framework that handles the logic of the system. Finally, MongoDB serves as the back end of the stack. Although Node.js and Express can cater for other types of databases.

Node.js and Express are what form the core of a MEAN stack. As is evidential in the prototype, the front end can be replaced with other templating languages. Replacing the front end (as in this instance with Pug) still allows for the main benefits, as articulated within the set criteria, to be present within the stack as a whole.

Although Angular does elevate the benefits of a MEAN stack, as already alluded to, the choice to use Pug was a deliberate decision to allow for a prototype to be deployed within the time frame of this project. As the client’s company already use Angular it was assumed that they are in a better position to evaluate what those benefits are. In addition, following conversations with the client, the perspective of the client’s company is how can the other elements of a MEAN stack benefit their current use of Angular. In essence, if the prototype meets the criteria without using Angular then those benefits can only increase with Angular being applied to later iterations.

What is Node.js?

According to Node.js.org (n.d), Node.js is a server side execution environment that allows users to build scalable network applications quickly. It is written in JavaScript and is open source and cross-platform. Reinforcing this statement, Holmes (2015) highlights that a MEAN stack allows for an end-to-end framework using the best of the best of JavaScript. Ergo, both the client and server side will consist of the same language. The statements made by Node.js.org (n.d) suggest that the criteria of maintainability, portability and scalability can be addressed with the use of Node.js, and will be explored in more detail within this report and subsequently articulated to the client.

One of the benefits of having a system with the vast majority of its components written in the same language is highlighted by Cappelli (2000), who addresses the woes software development companies have when acquiring and retaining talent. In essence, he states that the more niche an employee is, the more at risk a company is of having said employee poached by other companies. He goes on to state companies that recruit employees *“whom can do the job but are not in high demand… may be able to shelter themselves from market forces”.* Therefore, using a system that utilises the same language for both the client and server side address not only the requirement for expertise (as it mitigates against niche specialities), but also the woes of talent being poached.

Equally so, it addresses a common concept known as a “skill-set silo” (Businessdictionary.com, 2019) which within the software development industry refers to teams that do not share goals, tools, priorities and processes with other departments and is detrimental to a company’s overall performance.

In contrast, having the same language within the client and server side of the system will bring together employees whose contribution will span the full development cycle. Thus, allowing for cross-functional teams where each member, drawn from different areas of an organisation, maintain responsibility for the system throughout its entire life cycle. Pinto. M, et al (1993) have attributed cross-functional teams to a company’s overall performance and effectiveness and, therefore, from the perspective of this project a decrease in time to market for products.

In addition, having a whole stack that is written in the same language allows for insourcing from other departments to meet required deadlines. Equally so, owing to the simplicity of JavaScript, the recruitment pool for potential outsourcing tasks becomes larger.

Stepping back and viewing this from the perspective of a potential future member of the software development industry. Any organisation that adopts such an architecture will create an environment that will easily allow employees to develop professionally (and by extension allow members to better enact elements of the BCS code of conduct (2019)). This is mainly due to the fact that the largest hurdle for one to gain experience in different areas (in this instance referring generally to the apparent division between front-end and back-end developers) is the language.

However, returning back to the benefits of a single language architecture, owing to the fact that Node.js is unopinionated with little guidance on how the components of the system could be implemented a decrease in productivity is likely to be seen during the initial adoption period of the technology. This is mainly due to the fact that the company developers will need to define and agree on what the company perceived as good processes for the development and maintenance of code.

This statement, therefore, brings into question the concept of niche specialities. In general, a niche speciality refers to expertise in a specific language within a specific area of a system. With Node.js being unopinionated, a company that adopts said technology and defines its own best practices has, in essence, created its own niche speciality amongst its employees. Two hypothetical companies that adopt a MEAN stack may develop its application in different ways.

Therefore, the notion that a freshly recruited employee will immediately begin to contribute to the company's overall productivity is flawed. As a degree of in house training will be required. However, owing to the simplicity of Node.js said training is likely to consume less time than alternative technologies.

With regard to the concept of cross-functional teams, Santa & Pun (2009), highlight in their study that simply having cross-functional teams does not automatically result in better operational performance. They observed that an increase in operational performance was as a result of quality interaction and communication between teams. Which although the concept of cross-functional facilitates this, it does not enforce it.

Looking at the performance of Node.js and by extension the potential adoption rates of the system, it is an asynchronous, single-threaded, event driven JavaScript runtime, using Google’s V8 engine.

The single-threaded, event-driven architecture approach of Node.js allows for multiple simultaneous connections efficiently. In contrast, a multi-threaded application will create an additional thread for each new process which will consume a hosting machine’s resources.

It achieves this by asynchronously using event loops and callbacks. Ultimately, allowing it to execute non-blocking methods. The most common of which are I/O operations which when run synchronously will block all other events until said operation is completed. In deciding on an architecture for a full stack this is an important consideration with regard to the performance of a system.

Google's V8 is responsible for not only compiling and executing the code but for handling the call stack, memory management, garbage collection and providing the data types, operators, objects and functions (v8.dev, n.d). Owing to the fact that performance is a key driver in the competition between web browsers V8 is constantly being optimised. Therefore, this competition will ultimately benefit developers who use Node.js in their applications.

Regarding scalability, Node.js derives its name from the fact that its developers wanted to emphasis that a Node application comprises of multiple nodes which communicate with one another. This approach is in essence what makes Node.js scalable. Although the scalability of Node.js cannot be achieved by means of the prototype, a number of case studies are available by large enterprise organisations that have use Node.js for its scalability properties, such as NASA (foundation.nodejs.org, n.d) and Netflix (Alex Liu – Scaling A/B testing on Netflix.com with Node.js, 2014).

Node.js’ modular approach equally addresses integration and interfacing issues, as well as security and extensibility. Integration and interfacing issues are addressed by the fact that as alluded to, a MEAN stack utilises the best of the best of JavaScript (Holmes, 2015). Ultimately, a MEAN stack is designed to work together. Node.js is not restricted to use only the elements of a MEAN stack, there are numerous modules that allow it to integrate and interface with other technologies, another factor that contributes to Node.js’ extensibility.

Regarding security, Node.js is just a vulnerable as the next programming language, and it is up to the developers to addresses known vulnerabilities. Node.js' module approach allows developers to address these with relative ease with modules that implement protection against known flaws. Within the prototype two such modules have been utilised: Helmet, which sets HTTP headers and provides 14 functions such as enforcing HTTPS, prevent clickjacking and hiding MIME types; and express-validator which provides a number of sanitation functions, for instance, HTML characters are automatically escaped, as well as validation functions such as checking for correct input type. The later can be customised with a built in function that takes specific parameters, for instance, checking if an email already exists or if a password is valid.

The aforementioned modules were specifically highlighted as an example of how this modular approach contributes to productivity and subsequently time to market. The full extent of this will be fully appreciated by someone who has had the pleasure of debugging another developers validation function littered with Regex and is a prime example of common functions that are required within the majority of systems.

However, returning back to Node.js' performance. Node.js' ability to work asynchronously does not come packaged in the initial installation. A utility module must be installed with the Node Package Manager and imported into the relevant code logic that will utilise it.

With this in mind, we shall address the flaws of Node.js' modular approach. Although the core Node.js is stable the fact that it is an open source ecosystem results in some packages not being supervised and as a result, being of poor quality or lacking proper documentation.

As alluded to early, the Node Package Manager allows for easy installation of modules, however, it gives no indication of the quality or source of such a package. This, therefore, requires experienced developers who can find and know what tools can be trusted. Which brings us back to the concept of niche specialities and a company’s best practices.

Equally so, it highlights that until a company establishes best practices, employee time is likely to be consumed with corroborative research into the system itself and what quality modules to implement.

The implementation

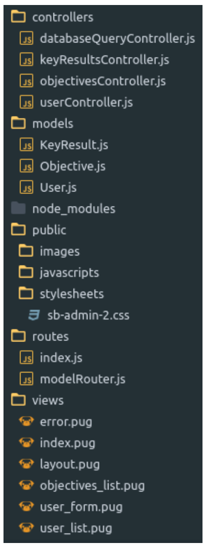
As articulated within the initial chapters of this project one of my main concerns was whether the design, which follows an object orientated approach, could be implemented within a MEAN stack. Recent updates to JavaScript have made it more object orientated, however, how it achieves this differs from other programming languages. For instance, JavaScript does not allow for access modifiers, however, within a Node.js environment it achieves encapsulation and data-hiding by its modular approach to where code resides within the system.

Exploring this in more detail, as Node.js is event driven this allows for the control flow of a program to be determined by an event. This is generally done with the use of event listeners which in turn trigger event handlers, which subsequently respond with some function that may or may not trigger further events. This is what allows for the data-hiding of code with event listeners only needing to know which event handlers to trigger. Before elaborating on this further what Express is, needs to be articulated.

What is Express?

As articulate within the previous paragraphs, Express is a web framework that allows for a web application to be structured to handle different http requests at a specific URL.

A class within the Express framework is, in essence, an abstract concept. It is defined by a file for its logic (controller), a separate file that in essence makes the logic within the controller encapsulated (route) and yet another file that essentially defines the persistence of an “instance” of said class within the database (model). Fig 6 illustrates the file structure for the prototype.



```
controllers
  databaseQueryController.js
  keyResultsController.js
  objectivesController.js
  userController.js
models
  KeyResult.js
  Objective.js
  User.js
node_modules
public
  images
  javascripts
  stylesheets
    sb-admin-2.css
routes
  index.js
  modelRouter.js
views
  error.pug
  index.pug
  layout.pug
  objectives_list.pug
  user_form.pug
  user_list.pug
```

Fig 6

From the folder structure that an Express application requires five files/folders are of particular importance and shall be explored in further detail (the content of these files is explored in detail within annexe A para 11b) :

app.js – (not captured within fig 6, however, it sits in the root of the folder structure) which initialises the application and in essence is equivalent to a main.java class.

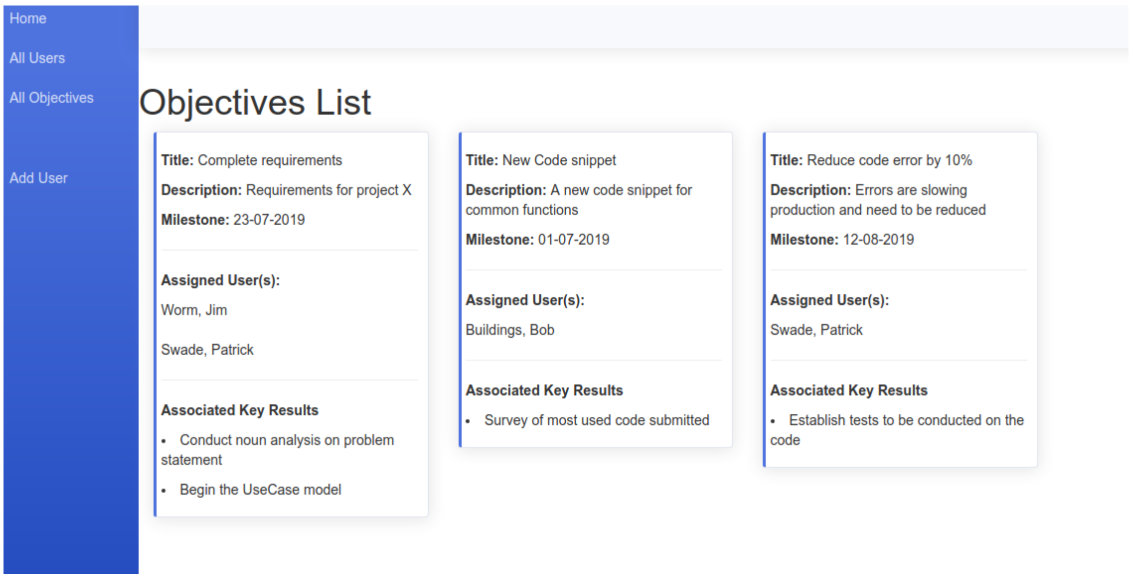
Controllers – these files are the application logic, with each file corresponding to a class within the Domain Structural Model

Routes – routing refers to determining how an application responds to a clients request which uses a URI. The URI is appended with a specific HTTP request method, such as GET and POST. The route will also define a handler function which is executed if a route is matched

Models – contains files each which will correspond to a database schema

Views – contains files containing the templates that display information to the user.

To elucidate this consider fig 7, which is a screenshot of the prototype returning data in response to a user selecting the "All Objectives" link in the navigational bar.



Home

All Users

All Objectives

Add User

Objectives List

Title: Complete requirements

Description: Requirements for project X

Milestone: 23-07-2019

Assigned User(s):

Worm, Jim

Swade, Patrick

Associated Key Results

- Conduct noun analysis on problem statement
- Begin the UseCase model

Title: New Code snippet

Description: A new code snippet for common functions

Milestone: 01-07-2019

Assigned User(s):

Buildings, Bob

Associated Key Results

- Survey of most used code submitted

Title: Reduce code error by 10%

Description: Errors are slowing production and need to be reduced

Milestone: 12-08-2019

Assigned User(s):

Swade, Patrick

Associated Key Results

- Establish tests to be conducted on the code

Fig 7

This particular function was implemented as it demonstrates an event flow that navigates the full stack. Ergo from the interface via the application logic to the database and back. Illustration 1 gives an abstract view of the event trace that shall be followed, although not technically accurate, this level of abstraction suffices for the following evaluation.

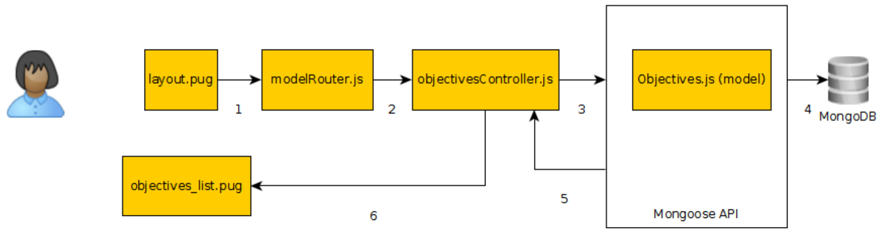


Illustration 1

Step 1

The user is displayed with a view that contains navigational links to list the objectives. This is provided with the `layout.pug` file. Note that this evaluation does not seek to elaborate on Pug, however, to avoid potential confusion Pug is templating engine, primarily used for server-side templating in NodeJS and uses indentation to define HTML elements.

```
body
      div(id="content-wrapper")
        block sidebar
          ul(class='navbar-nav bg-gradient-primary sidebar sidebar-dark accordion')
            li(class='nav-item')
              a(class='nav-link', href='/modelRouter') Home
            li(class='nav-item')
              a(class='nav-link', href='/modelRouter/users') All Users
```

```
22:                 li(class='nav-item')
23:                   a(class='nav-link', href='/modelRouter/objectives') All Objectives
24:                 hr
25:                 li(class='nav-item')
26:                   a(class='nav-link', href='/modelRouter/user/create') Add User
```

Of note is that each link has a reference to the `modelRouter` and provides a URI. Of relevance to this demonstration line 23 calls the `objectives_list` function.

Step 2

This particular URI is listed in the `modelRouter` in line 48 below. As the URI given matches a function, this function is called within the `objectivesController`.

```
47: // GET request for list of all objectives.
48: router.get('/objectives', objectivesController.objectives_list)
```

Step 3, 4, 5 & 6

This particular function is listed on line 5. Steps 4 & 5 are then facilitated by the Mongoose API which is accessed via the import on line 2 (see Objectives Model below for details). Looking at line 6 this calls a function to find the relevant data from the database. Lines 7 & 8 allows for recursive data to be returned. In this case, `assignedEmployees` and `keyResults` are relational elements from the `User` and `KeyResult` document databases. Within the Objectives document database, these are recorded by their `ObjectID`. The populate function allows for their actual details to be returned.

Finally, the `objectivesController` renders the data within the `objectives_list.pug` view, line 12 (note the express allows for file extensions to be omitted and will automatically decide on the files type by the context. In this case, the function `render` is called, therefore, Express can differentiate between `objectivesController.js` and `objectivesController.pug`

objectivesController.js

```
 2: var Objective = require('../models/Objective')
 3:
 4: // Display list of all Objectives.
 5: exports.objectives_list = function (req, res, next) {
 6:   Objective.find({}, 'title description milestone assignedEmployees keyResults')
 7:     .populate('assignedEmployees')
 8:     .populate('keyResults')
 9:     .exec(function (err, listObjct) {
10:       if (err) { return next(err) }
11:       // Successful, so render
12:       res.render('objectives_list', { title: 'Objectives List', objectives_list: listObjct })
13:     })
14: }
```

objectives model

```
 6: var ObjectiveSchema = new Schema(
 7:   {
 8:     title: { type: String, required: true },
 9:     description: { type: String, required: true },
10:     assignedEmployees: [{ type: Schema.ObjectId, ref: 'User' }],
11:     milestone: { type: Date, required: true },
    // objectiveProgress :{},
    keyResults: [{ type: Schema.ObjectId, ref: 'KeyResult' }]
  }
)
```

objectives_view.pug

```
  div
    - objectives_list.sort(function(a, b) {let textA = a.title.toUpperCase(); let textB = 
b.title.toUpperCase(); return (textA < textB) ? -1 : (textA > textB) ? 1 : 0;});
    each objct in objectives_list
      div(class="col-xl-3 col-md-6 mb-4")
        div(class="card border-left-primary shadow h-100 py-2")
          div(class="card-body")
            div(class="row align-items-center")
              div(class="col mr-2")
                p #[strong Title: ]#{objct.title}
                p #[strong Description: ]#{objct.description}
                p #[strong Milestone: ]#{objct.mileStone}
                hr
                p #[strong Assigned User(s): ]
                each val in objct.assignedEmployees
                  dl
                    dd #{val.name}
                hr
                p #[strong Associated Key Results]
                each kr in objct.keyResults
                  ul
                  li #{kr.title}
```

With a better understanding of how an application written in Express operates, the report will seek to address previous considerations and concerns with implementing the proposed design into a MEAN stack architecture.

As is evidential within the prototype, Express handles data-hiding and encapsulation in a different way to more classical object-orientated programming languages. More obviously is how it allows for Objects, Messaging, Methods and Abstraction. However, how does it handle other characteristics of object-oriented programming?

From the case studies examined during this project, Express applications were generally written in a hybrid of functional, object-orientated and procedural styles. Equally so, functions are objects that can be created outside of a class. Therefore, although implementing classes is possible within Express, it is likely to add overhead and complexity that is not necessarily required.

In implementing the solution into Express, the classes identified within the design phase were used as an abstract idea of what a combination of specific Model, Controller and Route files could achieve functionally.

It became apparent that design patterns such as the Singleton and Mediator were irrelevant. As in essence, Express by its nature employs a Mediator pattern by the way modules communicate with one another. Equally, some of the middleware modules used are in essence Singletons – for instance, Express itself.

Looking at other aspects of object-orientated programming adopting polymorphism is likely to be detrimental to how Express handles data-hiding via the unique URIs within a route file. Therefore, by extension inheritance is not necessarily needed.

Looking at composition, on the other hand, Express employs functional composition. Where the results of one function are pipelined into another to create a new function altogether. The `objectives_list` function gives an example of this by drawing on three separate models, executing the `populate` function to create a data object that is passed to the view and rendered.

Summary

Referring back TM354 (Software Engineering) the methods used to start with an abstract idea for a solution and end up with code that can be easily traced back through the system operations, functional requirements and use cases are highly effective when used with a classical object-orientated programming language.

When using a language such as JavaScript within an Express framework, although the methods prove effective, how the resultant code derives from the design is less apparent. For example, fig 5 displays the design structural model and the functions required for each class. However, as the concept of a class does not really exist within an Express application. Each one of those functions is an abstract concept as to what a specific combination of Controller, Model and Route files will achieve.

Should this project go on to be fully developed, I would probably ignore the system operations and communication models, and use the design structural model as an abstract model of what I want to achieve within the system. However, using a Volere template in this manner brings another antipattern in focus.

Source Making (2019), articulates a common antipattern within the software development industry known as the “The Grand Old Duke of York” which in essence captures the frictions between abstractionist and implementationists. In essence, abstractionists are comfortable discussing software design concepts. Whereas, implementationists, often require source code examples before they can grasp abstract concepts.

Having designed and developed this prototype in isolation, implementing an abstraction has been relatively easy – as I know what it is I intend to achieve. However, as part of a team, this approach would likely raise some issues. Source Making (2019), goes on to state that implementationists outnumber abstractionist within the software development industry, therefore, the use of complex abstract models could be detrimental to a company’s performance as more time will be required in clarifications and explanations.

In this instance, the more detailed the Volere template is the more likely that it would detract from the implementation of the solution. This would be further compounded by the fact that a MEAN stack architecture is unopinionated. For instance, two developers could arrive at the same solution in different ways.

From the client's perspective, this in itself is likely to raise other areas of focus. Ergo should they decide to adopt a MEAN stack architecture, are changes necessary in how they capture their designs.

Equally so, some of the findings within the evaluation suggest that a more agile approach to developing solutions using MEAN stack technology is more appropriate (assuming otherwise, without knowledge of how the client's company operates). Although this is beyond the scope of the project, it is an assessed impact of this project. Ergo, adopting the technology could potentially require procedural changes within the organisation.

Review

Current stage of project work

Within this current stage of work, the first iteration of a design has been created. In addition, a prototype of the solution has been developed within a MEAN stack and deployed online. As well as an evaluation of the use of a MEAN stack has been provided.

In order to address professional issues as defined by the BCS code of conduct (BCS, 2019). Mainly with regards to my capacity and knowledge to develop a working solution, but equally so with regards to my duty to the client. Communication with the client has been constant throughout the process of this project which served to refine the goals of this project as well as clarifying the end products. Therefore, ensuring that the end products met their required purpose.

To reiterate, the goal for this project is three-fold: to produce a first iteration design for a potential solution for a performance management system; to evaluate the use of a MEAN stack as a technology; to bridge the design process to a developed solution using a prototype within a MEAN stack architecture.

This three-fold approach would bring together key departments within the client's company, all of whom are stakeholders in the decision making process of the company. Ergo, DevOps, Risk Management, HR and Senior Management to name a few.

In essence, this project was analogous to building three pillars. With time being the main resource. Focussing on one pillar too much results in the others being underdeveloped. I believe that through communication with the client and time management this project has managed to produce three equally developed pillars that complement each other in the overarching goals of this project.

Legal, Social, Ethical and Professional issues

A Legal issue that was identified and addressed at an early stage was that of company confidentiality. Ultimately resulting in the company remaining anonymous throughout this project. It was agreed that I could use the situation as the basis of this project; however, I will not have access to specifics such as employees or in-depth processes that occur within the company.

In relation to social and ethical issues, should the proposed software solution go on to be developed and used by the client company, it will allow the company and its employees to quantify progress and set achievable objectives. Progress management is considered a positive function within business.

As highlighted by the Charter Institute of Personnel and Development (CIPD, 2018) *"Employees must understand what's expected of them, and to achieve those goals they need to be managed so that they're motivated, have the necessary skills, resources and support, and are accountable."* It is my belief that the impact on society from this system would be beneficial.

Equally so, as the client’s company is a software development company having a progress management solution will allow them to promote good practices within their organisation (BCS, 2019), particularly with their duty to the profession and encouraging fellow members in their professional development.

In providing an evaluation of using a MEAN stack architecture, allows all parties involved to better themselves, thus participating in lifelong learning regarding the practice of the IT profession (ACM, n.d).

The system will, however, contain personal data. Therefore, the client’s company would become the controllers of said personal data should this system go on to be developed (European Commission, n.d). With reference to article five of GDPR (EUR-Lex, n.d) and the principles relating to the processing of personal data, there are some principles that the system design needs to facilitate in order for the end user to be GDPR compliant.

The principles considered are as follows: Purpose limitation and Data minimization, the purpose of personal data is specifically to identify a user and is limited to that function; Accuracy, the system will allow the user to keep personal data accurate and up to date; Integrity and confidentiality, which is enforced by using SSL and HTTPs for traffic encryption and MongoDB encryption at rest functions.

These considerations were kept in mind when designing the system and the mitigation factors were implemented into the proposed use cases for this system.

In addressing professional issues, the BCS code of conduct (BCS, 2019) was used as a reference. With regards to professional competence from the onset of this project, it was highlighted that I do not have the capacity and or knowledge to develop a working solution.

This suited the client as the company's business is software development, and they are far better suited to developing a solution. Hence, this project will serve as one potential solution that they can adopt, saving employee time by me doing the research and conceptual development.

In addition, as gains were made during the project the potential benefits to the company were being conveyed by the client (as the sole contact I had with the company). This helped shape the overarching goals of the project in order to maximise the usefulness of the end products.

One negative impact that could affect end users is a misunderstanding that the prototype is a fully developed system. Remaining with professional competence, this fact will be reiterated within the final report that the client will receive.

Maintaining my duty to the relevant authority, ergo the client was addressed at the time the decision to take on this project was decided. Mainly, it was clarified that there would be no time constraints imposed by the client's company for this project.

Project Management

The Plan

To capture the project plan a Gantt chart was used (annexe C). The schedule was broken done into four aims that corresponded to the phases of the Unified Process Model. Each aim was further broken down into four phases: Planning, Research, Conceptual Development and Evaluation. With each phases containing relevant tasks.

As alluded to the project management life cycle that was used for this project was the Unified Process Model (UP). In addition, owing to the evaluative nature of the project, I used a structured-case model within the inception phase for each discipline(domain modelling, requirements, analysis and design) which was captured within the aforementioned plan as a research phase. The structured-case model is, in essence, an iterative method for research. Fig 8 illustrates the structure-case model. This model saw me planning a search, collecting the data, analysing the data and finally synthesising said data. Ultimately, the latter stages revealed more research that needed to be done, therefore, beginning a new cycle.

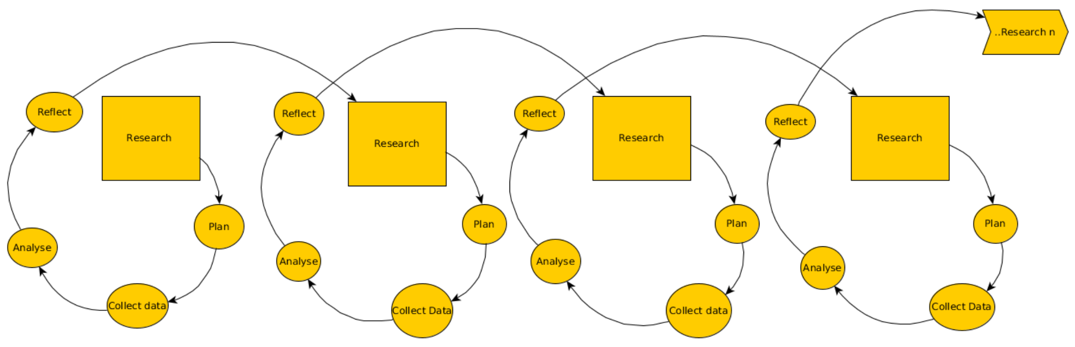


Fig 8

Risks

Fig 9 illustrates the risk assessment that was conducted on the project in the initial phases. The highest risks that were identified fell into a category of time, whereas the other risks generally fell into a category of "acts of nature". Further analysis of the risk assessment produced further examples that also fell within those categories, therefore, it was decided that the risk assessment had served its purpose and populating it further risked me becoming a slave to the tool. A common fallacy I am aware of owning to my current employment as an analyst and refers to the act of continuing to dedicate time and effort to a tool that has stopped being of use, or where the effort required out ways the tools usefulness.

Event	Likelihood	Impact	Risk Factor	Risk type	Mitigation
Time allocated reduce due to work commitments	0.8	0.7	1.5	High	Time has been allocated within the schedule as a buffer. Each time the buffer is used a new risk analysis Will be conducted.
Not understanding Angular enough to implement a design	0.6	0.8	1.4	Medium – high	Begin researching Angular now to give more time For understanding
Requirements changing	0.2	0.7	0.9	Low – medium	
Project being to ambitious	0.2	0.8	1	Medium	Evaluate the project after each phase and readjust
Time allocation miscalculated – overrunning	0.5	0.7	1.2	Medium	Re-evaluate time schedule at each phase and Adjust accordingly
Hardware failure	0.2	0.1	0.3	Low	
Reduced access to client	0.3	0.4	0.7	Low	
Illness	0.4	0.3	0.7	Low	
mental capacity reduce due to other module study	0.4	0.6	1	Medium	Create a time schedule for time allocation to Each module
Time allocated reduce due to family commitments	0.3	0.5	0.8	Low	
No access to internet due to work location	0.3	0.5	0.8	Low	

Fig 9

Moving forward in addressing the two holistic risks to the project a SWOT analysis on time available for this project was conducted, the highest threats to time were the project being too ambitious and the fact that there were several unknowns in the initial phases. It was assessed that in the later phases, the number of unknowns would proportionately grow with the ambition of the project.

The project ambition was driven by two factors: the content of the project being substantial enough; the project content complying with the BCS code of conduct. I had noted two potential end-states. In essence, the left and right boundaries of what I believe to be acceptable and what took into consideration the two previously mentioned factors. This allowed for the project goal to be refined, in order for it to remain within the boundaries of the project plan.

Mitigation

Mitigation efforts that were put in place to negate the aforementioned "acts of nature" was relatively simple. In essence, the external factors that caused them could be negotiated; using a mobile hotspot, rescheduling family commitments, etc.

In order to mitigate against risk factors relative to the time required measures being placed within the planning phase of the project. Ultimately, by placing a buffer within my schedule, which I could tap into as required.

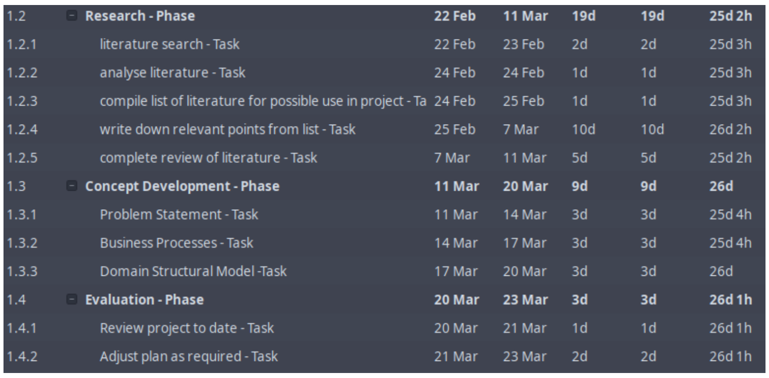
Applying a buffer was achieved by using the OU guideline of 10 hours work a week, which equated to two hours a day for a working week. Any work that was assessed to take two hours was calculated as a day on the planner. Also, I added extra days to tasks that I assessed could potentially be time-consuming.

This methodology left the weekend for a "Sprint" (adopted an Agile methodology) of two eight hour time-boxes as a buffer when required. This mitigation strategy was managed by constantly reassessing the project progress and the time available as the project progressed.

Review of the plan

In relation to the life cycle chosen, I found that blending the structured case model into the inception phase of each UP discipline seemed natural and useful. With regards to the time management, I found that populating a Gantt chart provided useful holistic time frames and a means to calculate milestones. In addition, it served as a reminder of what tasks needed to be completed within a given time frame.

Take into consideration fig 10, an extract from the Gantt chart. Initial observation suggests that these phases and tasks are linear. However, this proved inaccurate, as in reality many of this task were iterative and revisited throughout the illustrated phases.



1.2	**Research - Phase**	**22 Feb**	**11 Mar**	**19d**	**19d**	**25d 2h**
1.2.1	literature search - Task	22 Feb	23 Feb	2d	2d	25d 3h
1.2.2	analyse literature - Task	24 Feb	24 Feb	1d	1d	25d 3h
1.2.3	compile list of literature for possible use in project - Ta	24 Feb	25 Feb	1d	1d	25d 3h
1.2.4	write down relevant points from list - Task	25 Feb	7 Mar	10d	10d	26d 2h
1.2.5	complete review of literature - Task	7 Mar	11 Mar	5d	5d	25d 2h
1.3	**Concept Development - Phase**	**11 Mar**	**20 Mar**	**9d**	**9d**	**26d**
1.3.1	Problem Statement - Task	11 Mar	14 Mar	3d	3d	25d 4h
1.3.2	Business Processes - Task	14 Mar	17 Mar	3d	3d	25d 4h
1.3.3	Domain Structural Model -Task	17 Mar	20 Mar	3d	3d	26d
1.4	**Evaluation - Phase**	**20 Mar**	**23 Mar**	**3d**	**3d**	**26d 1h**
1.4.1	Review project to date - Task	20 Mar	21 Mar	1d	1d	26d 1h
1.4.2	Adjust plan as required - Task	21 Mar	23 Mar	2d	2d	26d 1h

Fig 10

In order to manage this, I used a combination of the project logs and a "Get Things Done" (GTD) methodology. For this I used a minimalistic application within the Linux command line – Task Warrior. This choice was made owing to self-awareness of how to maximise my own productivity levels.

Using the project logs I identified the minimal level of completeness for a task needed before I could move on to the next phase. For instance, a certain piece of knowledge needed to be garnered before moving on.

Taking the research phase as an example, key bits of research were entered into the GTD application bounded by the due date of 11 Mar. Models that needed to be completed before the evaluation could start were bounded by a due date of the 20 Mar and so on.

Subsequently, the completion of all the tasks as a whole (the research, conceptual development and the evaluation in this instance) were bounded by a due date of 23 Mar. This allowed for the project's overarching aims and milestones to be delivered on schedule, as per the Gantt plan (Annexe C), whilst still allowing for controlled elasticity in the completion of subordinate tasks.

This methodology mitigated against having to dedicate valuable time (the main risk identified) to minor alterations of the project plan as a whole, ergo the overarching aims (see Annexe C). In contrast, replicating the over-arching aims and phases of the Gantt plan within Task Warrior allowed for single line commands to dynamically plan, adjust and monitor subordinate tasks if the need arose.

Personal Development

In the past when conducting a project, I tended to have one milestone, which was the deadline. In this instance, the projects that I have conducted have been in my current line of work, which is as a military intelligence analyst within HM Forces.

It is important to understand that in some environments were I was required to operate, time and planning tools were a luxury that was not available. Therefore, my ability to date to manage projects without said tools is due to over 20 year of experience.

Conducting this project has allowed for a large change in mindset. Looking forward to a potential career within Computer Science, such a mindset is highly inefficient and does not allow for adequate updates to be given to stakeholders of the project.

The project has allowed me to adjust my project and self-management skills for a discipline that I lack experience in. Ultimately, it has allowed me to define a "unit of work" within this new field of expertise, and has given me a greater appreciation of how much resources (time and focus) a "unit of work" requires. This, therefore, will allow for more accurate time management in the future.

Related to the above-mentioned management skills is the discipline required to enact the plan and resisting what in essence is muscle memory and reverting back to what I have been doing for the past 20 year. It has given me a greater appreciation of keeping systematic records as I have worked through this project. Ultimately, it was these systematic records that allowed for meaningful communication with the client and subsequently shape the overarching goals of this project.

Approaching the synthesis of a solution I drew upon a number of skills that I had garnered during my study with the Open University. Primarily, these were as follows:

Software engineering (TM354) which allowed for the holistic design of a solution. It also allowed for an effective evaluation of a MEAN stack architecture using a classical Object Orientated architecture as a comparison.

Web, mobile and cloud technologies (TM352) which allowed for an in depth understanding of how the elements within a MEAN stack communicate, as predominantly it uses REST protocols to do so. In addition, this particular module introduced me to JavaScript which is the overall language of a MEAN stack.

Data management (TM351) and my acquired skill and knowledge of databases allowed me to develop a MongoDB database that will serve the back end of the MEAN stack.

Skills garnered externally to the OU that benefited me within the development of a solution is my understanding of the Linux command line interface (CLI). This is primarily due to the fact that Node.js – the execution environment, is managed and deployed using the CLI.

During the course of this project, one of the main skills that I have acquired is the ability to bind together all the skills that I have brought with me from my studies with the OU. More importantly, how I can adapt elements of this knowledge to achieve an end-state.

For instance, following the procedures of developing a design for a software solution has given me a greater understanding of the concepts involved. More importantly, it has taught me how to adapt existing concepts to meet a client's requirements. In the case of this project, this has been to adopt classic Object Orientated Programming concepts to a language whose approach to said concepts is different; for example, how it handles encapsulation and therefore by extension the concept of a class.

(Word Count 9715)

References

ACM (n.d) *ACM Ethics* [Online]. Available at https://ethics.acm.org/code-of-ethics/software-engineering-code/ (Accessed 29 Jun 2019).

Alex Liu – Scaling A/B testing on Netflix.com with Node.js (2014) YouTube video, added by Cian O'Maidin [Online]. Available at https://www.youtube.com/watch?v=gtjzjiTI96c& (Accessed 23 August 2019).

Apprenda (2019) *Types of Middleware* [Online]. Available at https://apprenda.com/library/architecture/types-of-middleware/ (Accessed 21 Jun 2019).

BCS (2019) *BCS Code of Conduct* [Online]. Available at https://www.bcs.org/membership/become-a-member/bcs-code-of-conduct/ (Accessed 29 Jun 2019).

Businessdictionary.com (2019) *Silo Mentality Definition* [Online]. Available at http://www.businessdictionary.com/definition/silo-mentality.html (Accessed 23 Aug 2019).

Cappelli, P (2000) *A Market-Driven Approach to Retaining Talent*. Harvard Business Review, vol. 78, no. 1, 2000, p. 103. Gale Academic Onefile [Online]. Available at https://link-gale-com.libezproxy.open.ac.uk/apps/doc/A59023945/AONE?u=tou&sid=AONE&xid=fa8774fe (Accessed 19 August 2019).

CIPD (2018) People Performance Fact sheet. [Online]. Available at https://www.cipd.co.uk/knowledge/fundamentals/people/performance/factsheet (Accessed 10 April 2019).

CIO (2019) 8 Challenges affecting software project management. [Online]. Available at https://www.cio.com/article/3065984/8-challenges-affecting-software-project-management.html (Accessed 18 August 2019).

ECMA-International (n.d) *ECMAScript 2017 Language* [Online]. Available at https://www.ecma-international.org/ecma-262/8.0/index.html#sec-ecmascript-language-functions-and-classes (Accessed 27 Jun 2019).

European Commission (n.d) *What is a data controller or a data processor* [Online]. Available at https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/obligations/controller-processor/what-data-controller-or-data-processor_en (Accessed 30 Jun 2019).

EUR-Lex (n.d) *General Data Protection Regulation* [Online]. Available at https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2016.119.01.0001.01.ENG&toc=OJ:L:2016:119:FULL (Accessed 30 Jun 2019).

Evolution4all (2018) *How OKR works*[Online]. Available at https://www.organisationalmastery.com/how-okr-works/ (Accessed 10 Mar 2019).

foundation.nodejs.org (n.d) *Node.js Helps NASA Keep Astronauts Safe and Data Accessible* [Online]. Available at https://foundation.nodejs.org/wp-content/uploads/sites/50/2017/09/Node_CaseStudy_Nasa_FNL.pdf (Accessed 23 Aug 2019).

Gamma, E. (ed.) (1995) *Design patterns: elements of reusable object-oriented software*, Addison-Wesley professional computing series, Reading, Mass, Addison-Wesley [Online]. Available at https://proquestcombo-safaribooksonline-com.libezproxy.open.ac.uk/0201633612 (Accessed 5 February 2019).

Holmes, S. (2015) *Getting MEAN With Mongo, Express, Angular, and Node*. Shelter Island, NY, Manning Publications [Online]. Available at https://learning-oreilly-com.libezproxy.open.ac.uk/library/view/getting-mean-with/9781617292033/kindle_split_000.html (Accessed 29 March 2019).

MDN Web Docs (2019) *Express/Node introduction* [Online]. Available at https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction (Accessed 30 March 2019).

MDN Web Docs (2019 a) *Express/Node introduction* [Online]. Available at https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs (Accessed 30 March 2019).

Moore, J. M. and Shipman, F. M. (2000) 'A comparison of questionnaire-based and GUI-based requirements gathering', *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering*, Grenoble, France, IEEE, pp. 35–43 [Online]. DOI: 10.1109/ASE.2000.873648 (Accessed 26 February 2019).

Nodejs.org (n.d) *About Node.js* [Online]. Available at https://nodejs.org/en/about/ (Accessed 17 August 2019).

Pinto, M., Pinto, J., & Prescott, J. (1993). 'Antecedents and Consequences of Project Team Cross-Functional Cooperation', *Management Science, 39*(10), 1281-1297. [Online]. Available at http://www.jstor.org.libezproxy.open.ac.uk/stable/2632967 (Accessed 23 August 2019).

Santa, R & Pun, W.K.D (2009) 'The impact of cross-functional teams on operational performance after the implementation of intelligent information systems' *2009 IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications.* Rende, Italy, 21-23 Sept. IEEE [Online]. Available at DOI: 10.1109/IDAACS.2009.5342921 (Accessed 24 August 2019)

Source Making (2019) *Software Architectural Antipatterns* [Online]. Available at https://sourcemaking.com/antipatterns/jumble (Accessed 22 Jun 2019).

Tsonev, K. (2014) *Node.js Blueprints*, Olton, UNITED KINGDOM, Packt Publishing Ltd [Online]. Available at http://ebookcentral.proquest.com/lib/open/detail.action?docID=1674844 (Accessed 1 April 2019).

Tsonev. K, (2015) Node.js By Example - Node.js By Example [Online]. Available at https://learning-oreilly-com.libezproxy.open.ac.uk/library/view/nodejs-by-example/9781784395711/index.html (Accessed 1 April 2019).

Young, R. R. (2003) The requirements engineering handbook, Artech House technology management and professional development library, Boston, Artech House [Online]. Available at https://ebookcentral.proquest.com/lib/open/reader.action?docID=227599 (Accessed 18 Jun 2019).

v8.dev (n.d) *Documentation* [Online]. Available at https://v8.dev/docs (Accessed 20 Aug 2019).

Volere.org (n.d) *Templates* [Online]. Available at https://www.volere.org/templates/ (Accessed 19 June 2019).

Volere.org (n.d, a) *Templates* [Online]. Available at https://www.volere.org/templates/ (Accessed 19 June 2019).

Annexe A – The Volere Template

Volere Template

Progress Management System

By Paul John Sharrock

Table of Contents

1. The purpose of this project

1a. Background of the project

The client for this project is the HR management of a software development company. The business being done within this project is to develop a performance management system.

The client identified the need to ensure that individuals are performing in line with the over all company strategic objectives. The end product will allow the user to track the progress of individuals and teams in this aspect.

Being able to identify under-performing, preforming, and up coming talent within the employees will allow the company to implement processes, where appropriate, that aim to maintain and improve employee performance in line with an organization's objectives.

It has been decided that a product that allows for the recording and querying of objectives and employee progress in line with these shall be designed for in house development.

1b. Goals of the project

Purpose. The new system will enable the company to record individual objects and respective comments from employees and their respective line managers. Thus allowing the company to identify under-performing/ preforming / and upcoming talent. In addition, the software will allow for reports to be generated in order for the HR team to visualize the overall performance of the company.

Advantage. Being able to manage the performance of the company will allow for company deadlines to be met, therefore, reducing delays on the time to market for products being developed. Designing such a product out of house will reduce the burden on company resources.

Measure. A system that captures, records and displays employee performance.

2. The stakeholders

2a. The Client

The client for this project is the HR manager of the company

2b. The customer

The HR team, who query the software to generate quarterly reports of company performance.

Department heads, who record objectives and key results of the team as a whole and monitor and manage the objectives and key results of the team members.
Department members, who record objectives and key results.

2c. Other stakeholders

These include the programmers and testers within the company. The knowledge needed by the project is the architecture that the programmers will develop the software on. This will influence the design of the delivered product.

2d. Hands-on users of the product

These include the employees with a positive approach to technology, however, their capacity to learn a new software system and subsequently interact with it will be limited due to current workloads.

3. Mandated constraints

3a. Solution constraints

Description. The performance management system will require minimal effort to learn and use.

Rationale. Hands-on users will have a limited capacity to be learning how to use and navigate a new system.

Fit criterion. The system user interface shall be familiar and it shall be easy to learn and use.

Description. The system shall be portable and non-operating systems dependant.

Rational. This will allow the software to be installed and used on existing and future frameworks will minimal effort.

Fit criterion. The system is operating software independent.

4. Naming conventions and terminology

OKR - Objectives and key results. This is a term used within HR management that encapsulates the process of setting objectives and recording progress on said objective. This, therefore, allows management a measurement for both progress and subsequently the performance of its employees.

Quarterly Milestone - Set by HR and signifies a point in time where departments are to consolidated and submit their progress to date

Objective/Key Result Milestone – Set internally by each department as a means to set deadlines and/or review progress to date.

User – A user of the system.

Initiator – A user that has initiated an event within the system.

Strategic Objective – An objective set by the CEO of the company.

Tactical Objective – An objective derived from Strategic Objectives which when complete will serve to achieve the Strategic Objective from where it is derived. Also referred to within this document as **Objective**

5. Relevant fact and assumptions

5a. Facts

The OKR system consists of the following components:

- **Objectives.** Defined by 3-5 key objectives department or individual levels. Objectives should be ambitious, qualitative, time bound and actionable by the person or team.

- **Results.** Under each Objective, 3-5 measurable results are defined. Key results should be quantifiable, achievable, lead to objective grading and be difficult, but not impossible.

  OKR results can be based on growth, performance, or engagement. Often they are numerical, but they can also show if something is done or undone, so a binary 0 or 1.

5b. Assumptions

Once defined, Objectives and Key Results will be communicated to everyone. The wording will be tuned and agreed by all, so everyone has a common understanding.

As people start working, they will update their result indicators regularly - preferably weekly. An objective is considered done when 70-75% of its results have been achieved. Employee performance is measured by the objectives that they complete within a given quarter (set by HR – see fig 2)

- Under Performing – 50% or less of objectives achieved

- As Expected – 51% to 79% of objectives achieved

- Exceeding – 80% + of objectives achieved

OKRs shall be reviewed regularly, as needed. If company, team, or personal goals change, that change can be cascaded to OKRs already set.

6. The scope of the work

6a. The current situation

Currently, the progress is recorded on a paper file system. This is cumbersome and does not allow for performance to be measured. In addition, these records are only kept for new employees within their first year of employment, as a result, employees that have been in the company for a longer period have no records at all.

6b. The context of the work

The following models illustrate the performance management process that is to be implemented within the company.

Fig 1 illustrates how the strategic objective, set by the CEO, is cascaded down to the individual. Each level within the company will set their own implicit objectives and key results on said objectives. This is done on a quarterly basis.

Fig 2 illustrates how progress will be recorded and reported on for each quarter. Progress is iteratively updated by the teams using internal milestones. Team progress is consolidated each quarter and an update given to the CEO.

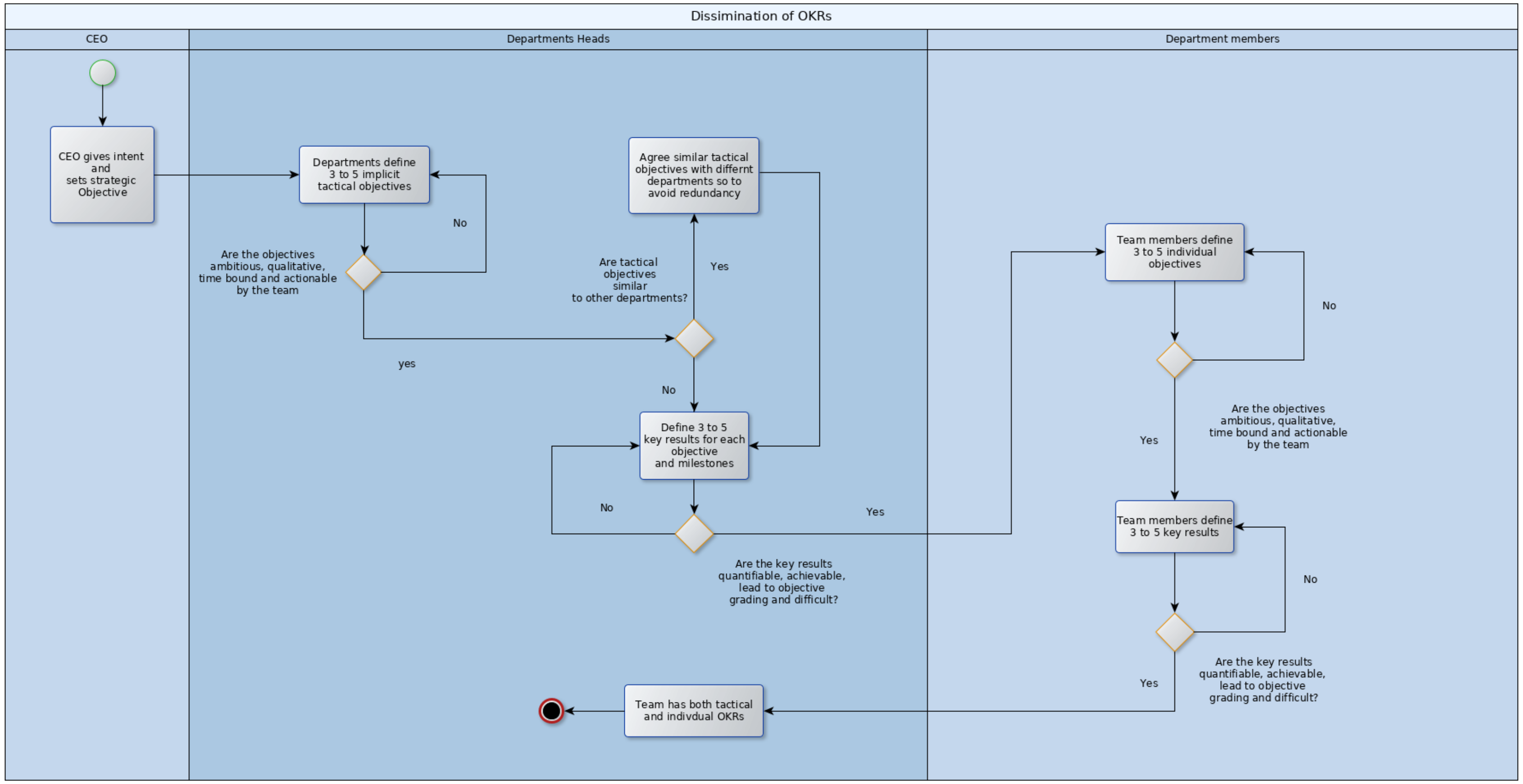

Dissimination of OKRs
CEO
Departments Heads
Department members
CEO gives intent and sets strategic Objective
Departments define 3 to 5 implicit tactical objectives
No
Are the objectives ambitious, qualitative, time bound and actionable by the team
yes
Agree similar tactical objectives with differnt departments so to avoid redundancy
Yes
Are tactical objectives similar to other departments?
No
Define 3 to 5 key results for each objective and milestones
No
Yes
Are the key results quantifiable, achievable, lead to objective grading and difficult?
Team members define 3 to 5 individual objectives
No
Yes
Are the objectives ambitious, qualitative, time bound and actionable by the team
Team members define 3 to 5 key results
No
Yes
Are the key results quantifiable, achievable, lead to objective grading and difficult?
Team has both tactical and indivdual OKRs

Fig 1 - the dissemination of OKRs per quarter

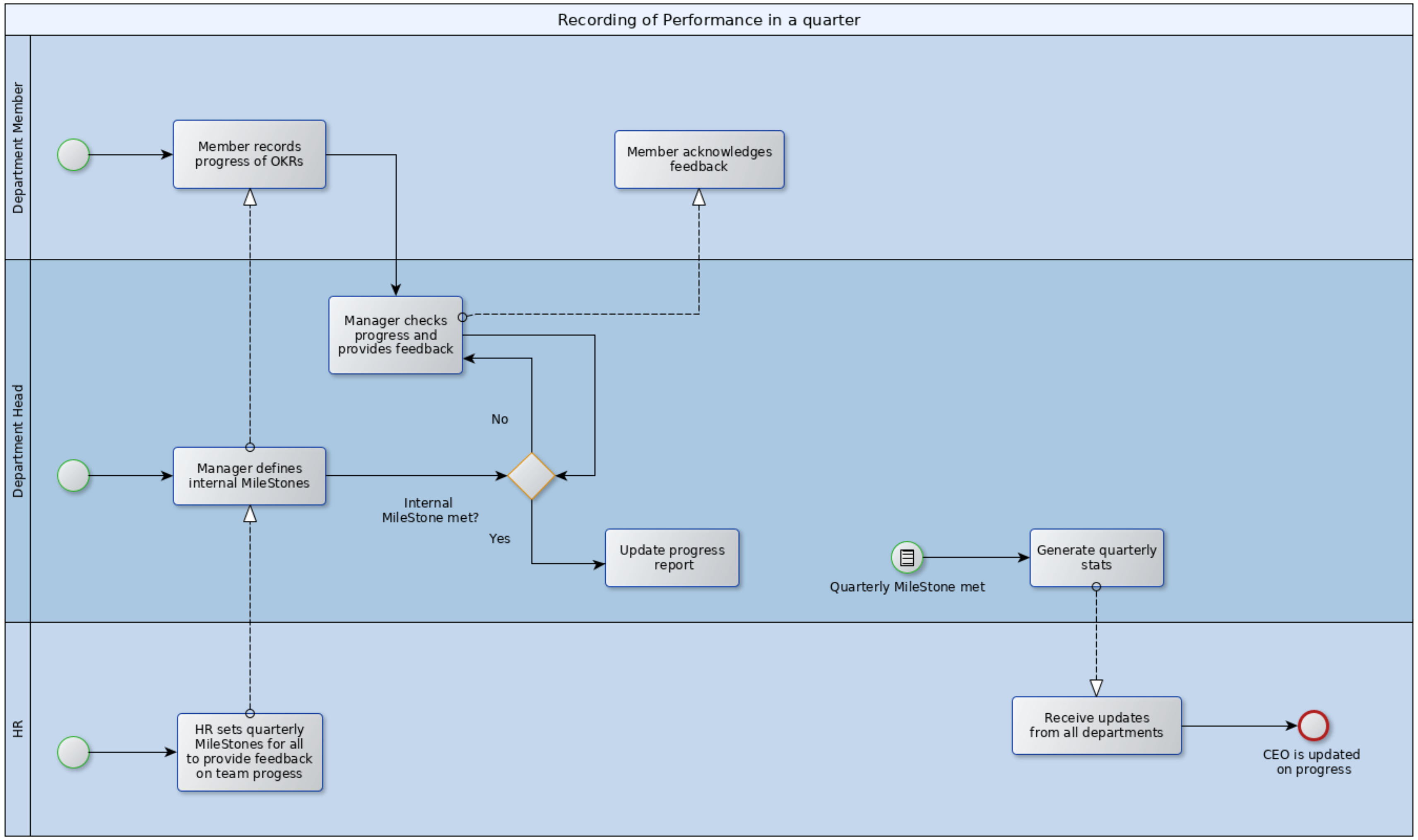

Recording of Performance in a quarter
Department Member
Department Head
HR
Member records progress of OKRs
Member acknowledges feedback
Manager checks progress and provides feedback
No
Manager defines internal MileStones
Internal MileStone met?
Yes
Update progress report
Quarterly MileStone met
Generate quarterly stats
HR sets quarterly MileStones for all to provide feedback on team progess
Receive updates from all departments
CEO is updated on progress

Fig 2 - how progress is recorded

6c. Work partitioning

No	Event Name	Input/Trigger	Input and Output	Business Use case summary
1	Define tactical objective	Strategic objectives given	Tactical objectives set (in)	Determine if the user has the right authorizations and set objectives accordingly associated to a given user. Objectives are then stored to the database
2	Define key results	Tactical objectives set	Key results set (in)	Determine if the user has the right authorizations and set key result for a given objective accordingly. Key Results are then stored to the database
3	Set Milestone	External milestone set	Milestone set (in)	Determine if the user has the right authorizations and set milestone for a given Key Result accordingly. The given Key Result is updated and the update is stored to the database.
4	Record progress	*none*	Progress incremented (in)	Determine if the user has the right authorizations and increment progress for a given Key Result. The Key Result is updated and the

				update is stored to the database.
5	Provide feedback	Progress recorded	Feedback given(in)	Manager receives notification of progress and provides feedback for a given user.
6	Generate report	Quarterly milestone met	Report generated (out)	Manager consolidates his teams progress and submits to HR
7	Edit objective and or key results	*none*	Objectives/key results edited (in)	Determine if the user has the right authorizations and edit an objective/key results for a given user. Changes are then stored to the database.

7. Business Data Model

7a. Data Model

A initial domain structural model is illustrated within Fig 3.

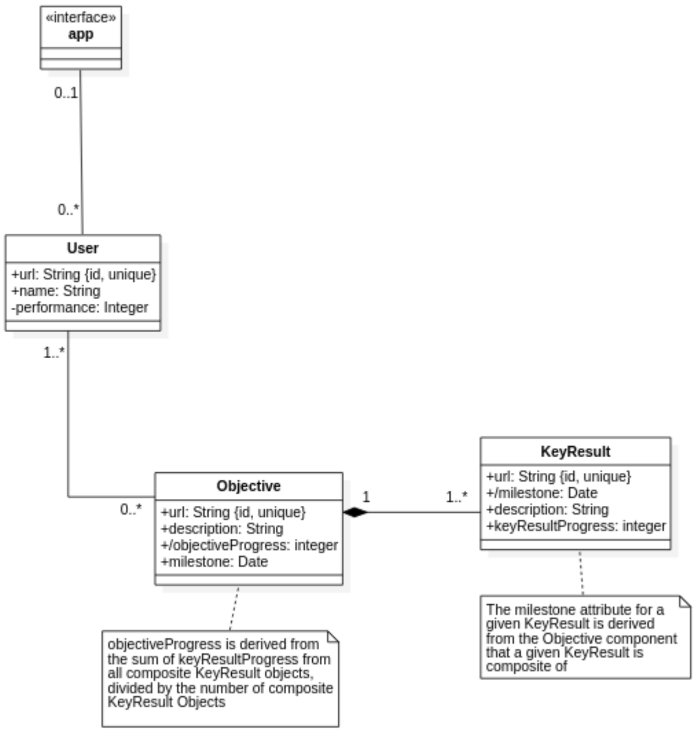

«interface»
app
0..1
0..*
User
+url: String {id, unique}
+name: String
-performance: Integer
1..*
0..*
Objective
+url: String {id, unique}
+description: String
+/objectiveProgress: integer
+milestone: Date
1
1..*
KeyResult
+url: String {id, unique}
+/milestone: Date
+description: String
+keyResultProgress: integer
objectiveProgress is derived from the sum of keyResultProgress from all composite KeyResult objects, divided by the number of composite KeyResult Objects
The milestone attribute for a given KeyResult is derived from the Objective component that a given KeyResult is composite of

Fig 3

7b. Data Dictionary

The following is an elaboration on the Classes and their attributes.

Name	**Content**	**Type**
System		Class
User	ID + Name + Performance	Class
Objective	ID + Description + Objective Progress + Milestone + title	Class
Key Result	ID + Milestone + Description + Key Result Progress + title	Class
App - User	The app will contain a number of Users	Association
User - Objective	A User will contain a number of Objectives	Association
Objective – KeyResult	An Objective will contain a number of KeyResults	Association
Url	A unique string that allows for the class instance to be recorded to a Database	Attribute
User.name	The users name that allows for human readable identification	Attribute
User.performance	Determined per quarter as per the quarterly milestones set by HR (see Fig 2). Thresholds are set as per the assumptions articulated. The value is calculated as the sum of Objective.progress attribute for all Objectives associated for a given User, divided by the number of Objectives associated to a given User.	Attribute
Objective.description	A textual description of the objective.	Attribute
Objective.objectiveProgress	An integer, Objective.objectiveProgress is derived from the sum of KeyResult.keyResultProgress from all composite KeyResult objects for a given Objective, divided by the number of composite KeyResult Objects for a given Objective.	Attribute

Objective.milestone	A date set internally by departments as a deadline/point in time for a review.	Attribute
Objective.title	A string that denotes the title of the objective	Attribute
KeyResult.milestone	A date that is derived from the Objective object to which a given KeyResult is a component of.	Attribute
KeyResult.description	A textual description of a given KeyResult.	Attribute
KeyResult.keyResultProgress	An integer which is a percentage from 1 to 100 and measures how complete a given KeyResult object is.	Attribute
KeyResult.title	A string that denotes the title of the keyResult	Attribute

8. The Scope of the Product

8a. Product Boundary

Fig 5 illustrates the use cases that will facilitate the application logic for this product.

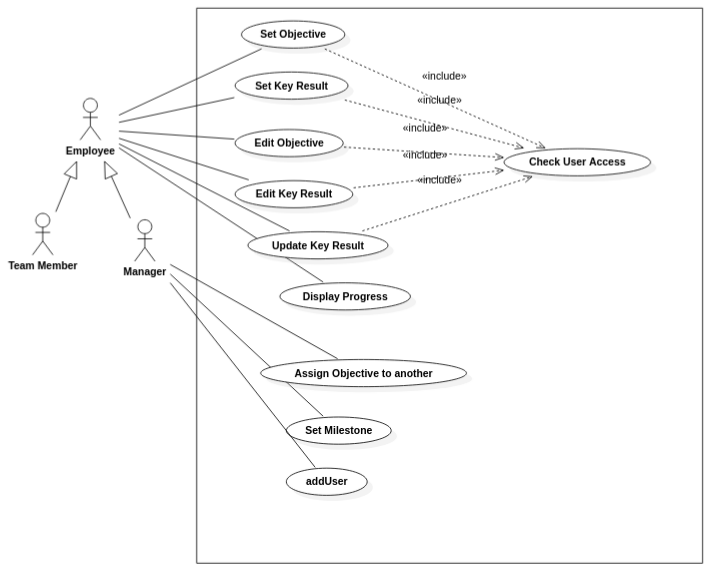


Fig 5

8b. Elaborate Use Cases

Identifier and name:	UC1 *set objective*
Initiator:	*Employee*
Goal:	An objective is set at a level appropriate to the initiator's authorization.
Precondition:	none
Postcondition:	An objective object is created with properties as set by the initiator.
Assumptions:	The quality of the objective is ascertained by the initiator. The initiator has already log into the system with their credentials.
Main success scenario:	1. The initiator selects to set an objective. 2. The initiator selects the scope of the objective (department or individual). 3. The system checks the initiator's authorization. 4. The initiator enters the properties of the objective and submits them to the system. 5. The system accepts the new objective and its properties, which are then stored.
Extension:	3.a.1 *initiator does not have authorization.* A new objective is not set and the initiator is informed

Identifier and name:	UC2 *set key results*
Initiator:	*Employee*
Goal:	A key result is set.
Precondition:	A composite objective exists.
Postcondition:	A key result object is created and it is a component a composite objective object
Assumptions:	The initiator has already log into the system with their credentials.
Main success scenario:	1. The initiator selects an existing objective. 2. The system checks the initiator's authorization to set key results for the given objective. 3. The initiator enters the properties of the key result for the given objective and submits them to the system. 4. The system accepts the key result and its properties, which are then stored.
Extension:	2.a.1 *initiator does not have authorization.* A new key result is not set and the initiator is informed

Identifier and name:	UC3 edit *objective*
Initiator:	*Employee*
Goal:	An existing objective is edited.
Precondition:	An objective exists.
Postcondition:	The properties for a given objective are edited.
Assumptions:	The initiator has already log into the system with their credentials.
Main success scenario:	1. The initiator selects an existing objective 2. The system checks the initiator's authorization to edit the objective. 3. The initiator edits the properties for a given objective 4. The system accepts the changes to a given objectives properties and the objective is stored.
Extension:	2.a.1 *initiator does not have authorization.* Changes to a given objective are rejected and the initiator is informed

Identifier and name:	UC4 *edit key results*
Initiator:	*Employee*
Goal:	An existing key result is edited
Precondition:	A key result that is composite to an objective exists.
Postcondition:	The properties of a given Key Result are edited.
Assumptions:	The initiator has already log into the system with their credentials.
Main success scenario:	1. The initiator selects an objective. 2. The system displays all associated key result for a given objective. 3. The initiator selects to edit a key result. 4. The system checks the initiator's authorization to edit the key result. 5. The initiator edits the properties of the key result. 6. The system accepts the edit and it is stored.
Extension:	2.a.1 *initiator does not have authorization.* Changes to a given key result are rejected and the initiator is informed

Identifier and name:	UC5 *update key results*
Initiator:	*Employee*

Goal:	The progress of a key result is updated
Precondition:	A key result exists
Postcondition:	The progress for a given key result is updated.
Assumptions:	The initiator has already log into the system with their credentials.
Main success scenario:	1. The initiator selects an objective. 2. The system displays all associated key result for a given objective. 3. The initiator selects to update a key result's progress. 4. The system checks the initiator's authorization to update the key result 5. The initiator updates the key result's progress 6. The system accepts the update and it is stored.
Extension:	2.a.1 *initiator does not have authorization.* Updates to a given key result are rejected and the initiator is informed.

Identifier and name:	UC6 *assign objective to another*
Initiator:	*Manager*
Goal:	An objective is assigned to another employee
Precondition:	An objective exists
Postcondition:	A new association between a given objective and another user object is created.
Assumptions:	The initiator has already log into the system with their credentials.
Main success scenario:	1. The initiator selects a given objective. 2. The system checks the initiator's authorization to assign an objective to another. 3. The initiator assigns the objective to another user 4. The system accepts the assignment and it is stored
Extension:	2.a.1 *initiator does not have authorization.* The assignment of a given objective is rejected and the user is informed

Identifier and name:	UC7 *display progress*
Initiator:	*Employee*
Goal:	The progress of key results for a given objective is displayed.
Precondition:	An objective with key results exits
Postcondition:	The key results for a given objective are displayed to the user.
Assumptions:	The initiator has already log into the system with their credentials.
Main success scenario:	1. The initiator selects a given objective 2. The system displays the objective and the progress of all its associated key results

Identifier and name:	UC8 *set milestone*
Initiator:	*Manager*
Goal:	A milestone for a given objective is set.
Precondition:	An objective exists.
Postcondition:	A milestone property for a given objective is set
Assumptions:	The initiator has already log into the system with their credentials.
Main success scenario:	1. The initiator selects a given objective 2. The system checks the initiator's authorization to set a milestone 3. The initiator sets a milestone for a given objective 4. The system accepts the milestone and it is recorded
Extension:	2.a.1 *initiator does not have authorization.* The setting of a milestone is rejected and the user is informed

Identifier and name:	UC9 *add User*
Initiator:	*Manager*
Goal:	A new user is added to the system
Precondition:	None
Postcondition:	A new user object is created and stored to the system.
Assumptions:	The initiator has already log into the system with their credentials.
Main success scenario:	1. The initiator selects to add a new user. 2. The system checks the initiator’s authorization to add a new user. 3. The initiator set the user properties. 4. The system accepts the new user object and its properties and it is recorded.
Extension:	2.a.1 *initiator does not have authorization.* The creation of a new user is rejected and the initiator is informed.

9. Functional and non-functional; Requirements

Reqt #	Reqt type	Description	Rationale	Fit criterion
SFR1: UC1, Step 1	Functional	The system shall allow a user to initiate a new objective	This will allow employees to set their objectives or those of the department	The set objective page is displayed
SFR2: UC1, Step 2	Functional	The system shall allow the user to select the scope of the new objective (department or individual)	This will allow employees to allocate objectives to themselves of other employees	The scope of the objective is accepted if the user is authorized to see S1
SFR3: UC1, Step 3	Security	See Reqt S1		
SFR4: UC1, Step 4	Functional	The system will allow a user to record objectives properties.	This will allow employees to record their objectives or those of the department	The objective details are accepted by the system
SFR4: UC1, Step 5	Functional	The system will store the objective properties set by the initiator.	This will allow for objectives to be stored	The objective details are stored by the system
SFR5: UC2, Step 1	Functional	The system shall allow a user to selected an objective to set a new associated key result with said objective	This will allow employees to assign key results to an objective	If the user is authorized, objects associated with the user are displayed, once an objective is selected the set new key result page is displayed
SFR6: UC2, Step 2	Security	See Reqt S1		
SFR7: UC2, Step 3	Functional	The system will allow a user to	This will allow employees to	The key result details are

		record key result properties.	record key results for a given objective	accepted by the system
SFR8: UC2, Step 4	Functional	The system will store the key result properties set by the initiator.	This will allow for key results to be stored	The key result details stored by the system
SFR9: UC3, Step 1	Functional	The system shall allow a user to selected an objective to edit its properties.	This will allow a user to change an objective in the future	If the user is authorized, objects associated with the user are displayed, once an objective is selected the edit objective page is displayed
SFR10: UC3, Step 2	Security	See Reqt S1		
SFR11: UC3, Step 3	Functional	The system shall allow a user to edit the properties of an associated objective	This will allow a user to change an objective in the future	Edits to an objective are accepted by the system
SFR12: UC3, Step 4	Functional	The system shall store the edits to a given objective.	This will allow for changes to an objective to be saved	The system stores the changes for a given objective
SFR13: UC4, Step 1, Step 2, Step 3	Functional	The system shall allow a user to selected an objective to edit the key result associated with said objective	This allows users to change key objectives in the future	If the user is authorized, objects associated with the user are displayed, once an objective is selected the edit key results page is displayed
SFR14: UC4, Step 4	Security	See Reqt S1		

SFR15: UC4, Step 5	Functional	The system shall allow a user to edit the properties of an associated key result	This allows users to change key objectives in the future	Edits to a key result are accepted by the system
SFR16: UC4, Step 6	Functional	The system shall store the edits to a given key result	This allows changes to key results to be stored	The system stores the changes for a given key result
SFR17: UC5, Step 1, Step 2, Step 3	Functional	The system shall allow a user to selected an objective to update the progress of the key result associated with said objective	This will allow employees to update their progress	If the user is authorized objects associated with the user are displayed, once an objective is selected the update key results progress page is displayed
SFR18: UC5, Step 4	Security	See Reqt S1		
SFR19: UC5, Step 5	Functional	The system shall allow a user to update the progress of an associated key result	This will allow employees to update their progress	Updates to a key result are accepted by the system
SFR20: UC5, Step 6	Functional	The system shall store the updates to a given key result	This allows updates to key results to be stored	The system stores the updates for a given key result
S1	Security	The system will ensure users have the right access to edit objectives, milestones and/or key results as well as the progress of a key result	This will ensure that users with the right authorization are editing objectives, milestones and/or key results. Objectives set	Scenario 1: A user attempts to edit an objective, milestone and/or key result that they are not authorized to do so and the system rejects the change.

			by a manager cannot be edited by an employee	Scenario 2: A user attempts to edit an objective, milestones and/or key result that they are authorized to do so and the system accepts the change
SFR21: UC6, Step 1	Functional	The system shall allow a user to selected an objective to assign it to another	Some objectives will be similar between departments. Managers can reallocate objectives and by extension its key results between departments. Some objectives may be applicable to two employees from different departments	If the user is authorized objects associated with the user are displayed, once an objective is selected the assign objective to another page is displayed
SFR22: UC6, Step 2	Security	See Reqt S2		
SFR23: UC6, Step 3	Functional	The system shall allow a user to assign an objective to another	This will allow managers to assign objectives and by extension its key results between departments	The selected objectives and its associated key results are associated with the users selected by the initiator and the system accepts the changes
SFR24: UC6, Step 4	Functional	The system shall store new user/objective and key result associations.	This will allow assignments made by managers to be stored	The system stores the new association for a given objective and by extension

				it key results
SFR25: UC7, Step 1	Functional	The system shall allow a user to select objectives associated with the user and display it progress	This will allow employees to view the progress on a given objective	The display progress page is displayed
SFR26: UC7, Step 2	Functional	The system shall display the progress for a given objective and its associated key results	This will allow employees to view the progress on a given objective	The system shall calculate the progress of a given objective which is derived from the total progress of all associated key results divided by the number of associated key results and display the progress for a given objective and all of its associated key results
SFR27: UC8, Step 1	Functional	The system shall allow a user to select objectives associated with the user to set a milestone for a given objective	This will allow users to set milestones	If the user is authorized objects associated with the user are displayed, once an objective is selected the set milestone page is displayed
SFR28: UC8, Step 2	Security	See Reqt S2		
SFR29: UC8, Step 3	Functional	The system shall allow a user to set a milestone for a given objective	This will allow users to set milestones	The system accepts the milestone for a given objective

SFR30: UC8, Step 4	Functional	The system shall store milestones for a given objective	This will allow milestones to be stored	The system stores the milestone for a given objective
SFR31: UC9, Step 1	Functional	The system shall allow a user to add a new user	This will allow managers to add new users to the system	If the user is authorized the add new user page is displayed.
SFR32: UC9, Step 2	Security	See Reqt S2		
SFR33: UC9, Step 3	Functional	The system will allow a user to set the properties of a new user.	This will allow managers to set the properties of a new user object	A new user object is created and its properties are set.
SFR34: UC9, Step 4	Functional	The system shall store new user objects and their properties.	This will allow for new users to be recorded on the system	A new user object is stored in the database.
S2	Security	The system will ensure that users have the right access to set objectives, and/or milestones between teams and between individuals from different teams. In addition, the system will ensure that users have the right access to add new users.	Managers will be responsible for reallocating objectives, and/or milestones between teams. This will ensure that those without the right access cannot do so.	Scenario 1: A user attempts to reallocate an objective and/or milestone that they are not authorized to do so and the system rejects the change. Scenario 2: A user attempts to reallocate an objective and/or milestone that they are authorized to do so and the system accepts the change
A1	Style requirements	Progress shall be amalgamated by the system	This will allow users to visualize the	The system automatically shows progress

		into a dashboard showing a hierarchical view of objectives and corresponding key results.	progress	in a hierarchical view
A2	Style requirements	The system will show how OKRs feed into the strategic OKRs and calculate the overall progress of the company.	This will allow the user to visualize how the progress of the overall company and that of the individuals are linked	The system generates a view that allows a user to visualize how OKRs are connected.
A3	Style requirements	The system will highlight areas that are behind a set level of progress.	This allows managers and HR to pre-empt delays within the company progress	Given a threshold, the system highlights those areas that are behind
A4	Style requirements	The system will show visually whether the company is on track of delivering its strategic OKRs, whether it is exceeding or not.	This allows managers and HR to pre-empt delays within the company progress	Given a threshold, the system highlights those areas that are behind, ahead or on schedule
A5	Style requirements	The system shall highlight teams and/or individuals that are top-performers, deliver solid and on track every time and/or under-performing.	This allows managers and HR to reward top-performers of to manage under-performers	Given a scope and a threshold the system highlights individuals that are top-performers, deliver solid and on track every time and/or under-performing
A6	Audit	The system will	This will allow for	The system

	requirements	store the data in a database and allow for backups to be made	the integrity of the data	stores data to a database automatically as a user submits their changes
M1	Maintainability	The system shall be able to be ported to other operating systems	This will allow for the maintainability of the system.	The system is operating system independent
C1	Convenience requirements	The system shall allow users to provide feedback on another’s OKRs	This will allow users to provide constructive criticism	A user provides feedback on another’s OKRs and the system accepts the feedback and the feedback is displayed to the other user
U1	Usability	The system will be simple to use	Employee capacity should not be reduced by the system – either using it or learning how to use it	A user finds the system intuitive and familiar to use

10. Design Analysis

10a. System Operations

UC1 *New Objective*

context user::newObjective(title: String, description : String, milestone : Date) : String

pre:

- - milestone is not in the past

post:

- - a new Objective object, objective, will have been created (object creation)
- - user will be linked to objective
- - a string indicating success will have been returned
- - otherwise, a string indicating failure will have been returned

UC2 *New Key Results*

context objective::newKeyResult(title : String, description : String) : String

pre:

- - objective exists

post:

- - a new KeyResult object, keyResult, will have been created (object creation)
- - objective will be linked to keyResult
- - a string indicating success will have been returned
- - otherwise, a string indicating failure will have been returned

UC3 *Edit Objective*

context user::editObjective(title : String, description : String, milestone : Date) : String

pre:

- - objective exists

post:

- - objective properties will have been edited
- - a string indicating success will have been returned
- - otherwise, a string indicating failure will have been returned

UC4 *Edit Key Result*

context objective::(title : String, description : String) : String

pre:

- - keyResult exists

post:

- - keyResult properties will have been edited
- - a string indicating success will have been returned
- - otherwise, a string indicating failure will have been returned

UC5 *Update Progress*

context: keyResult::(amount : Integer) : String

pre:

- - keyResult exists

post:

- - keyResult’s keyResultProgress will have been updated
- - a string indicating success will have been returned
- - otherwise, a string indicating failure will have been returned

UC6 *Reassign Objective*

context: System::(objective : Objective, user : User, anotherUser : User) : String

pre:

- - user exists
- - anotherUser exits
- - objective exits

post:

- - a new link between objective and anotherUser will have been created
- - a string indicating success will have been returned
- - otherwise, a string indicating failure will have been returned

UC7 *Display progress*

context: System::(user : User, objective : Objective) : Integer

pre:

- - user exists
- - objective exists

post:

- - for all keyResults objects associated with objective, the average of the keyResultProgress is displayed as an integer

UC8 *Edit Milestone*

context objective::setMilestone(milestone : Date) : String

pre:

- - objective exists

post:

- - objective properties will have been edited
- - a string indicating success will have been returned
- - otherwise, a string indicating failure will have been returned

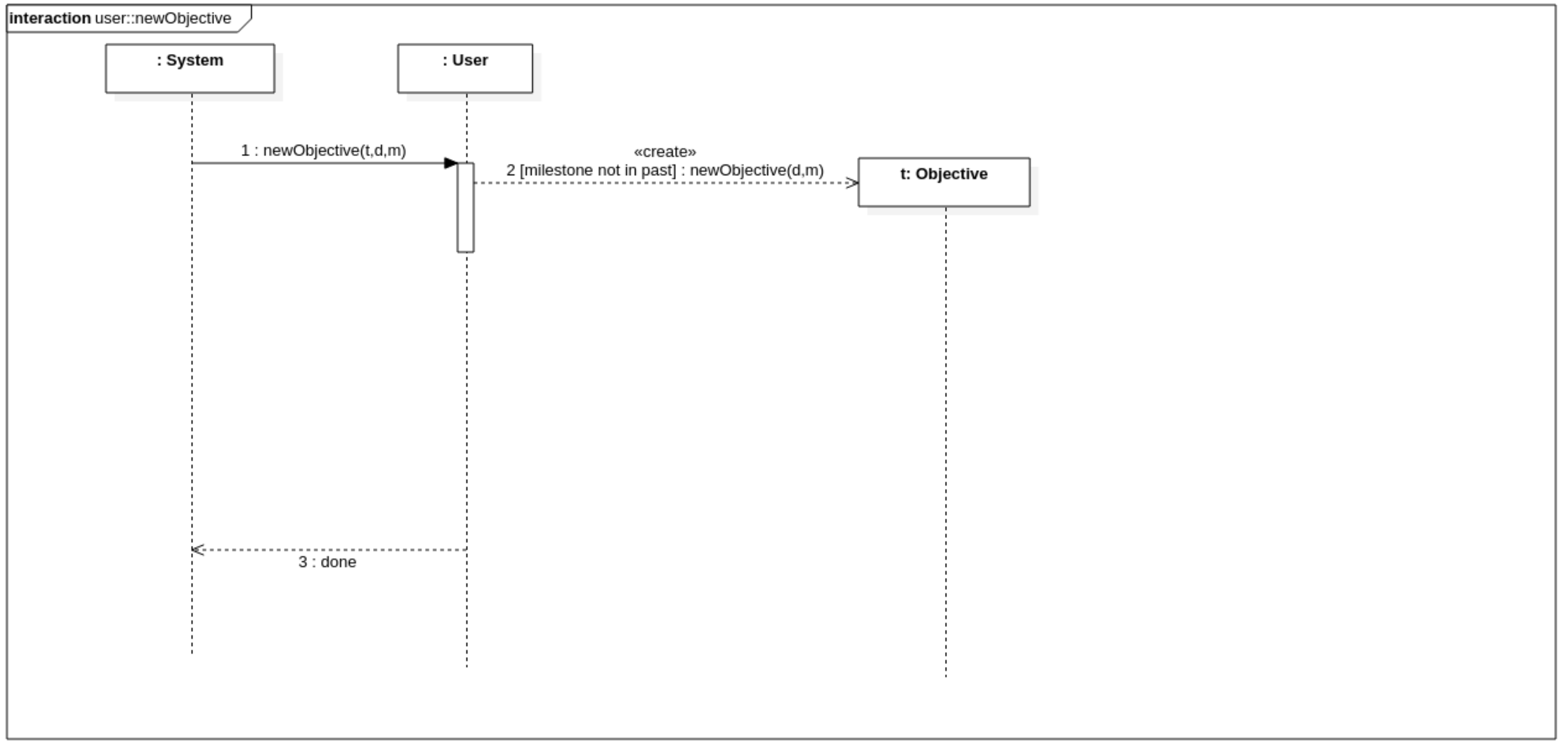
UC9 *Add User*

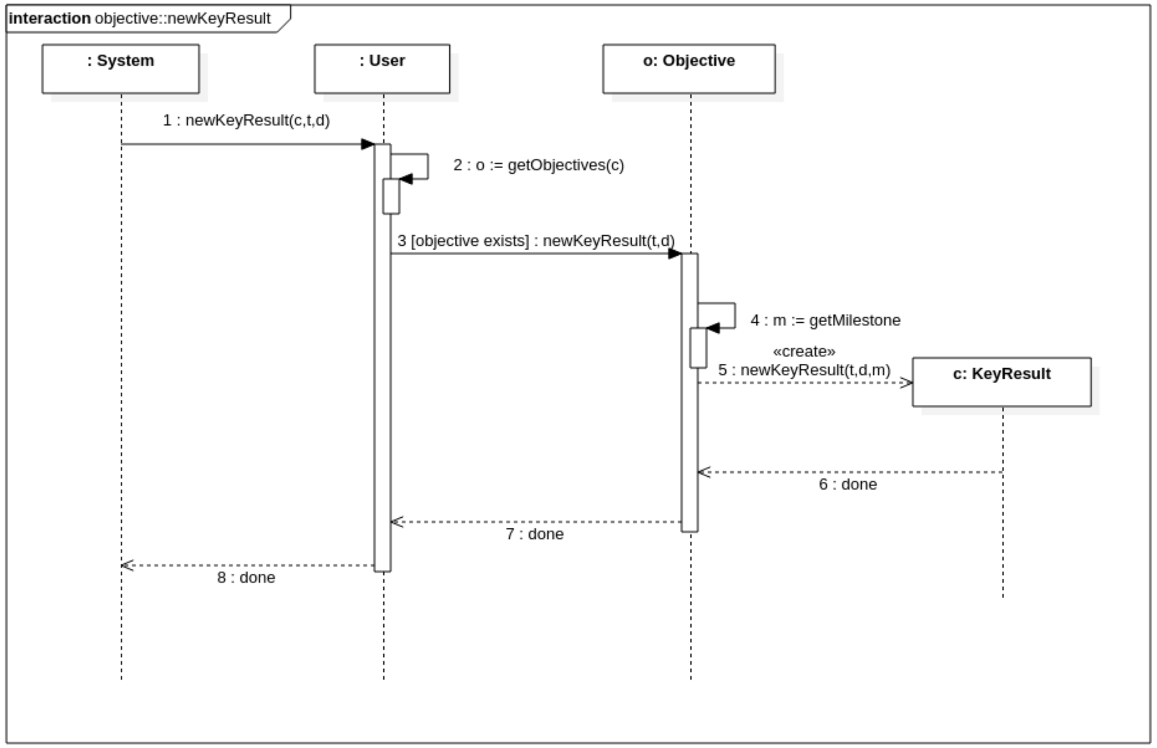
context System::(name : String) : String

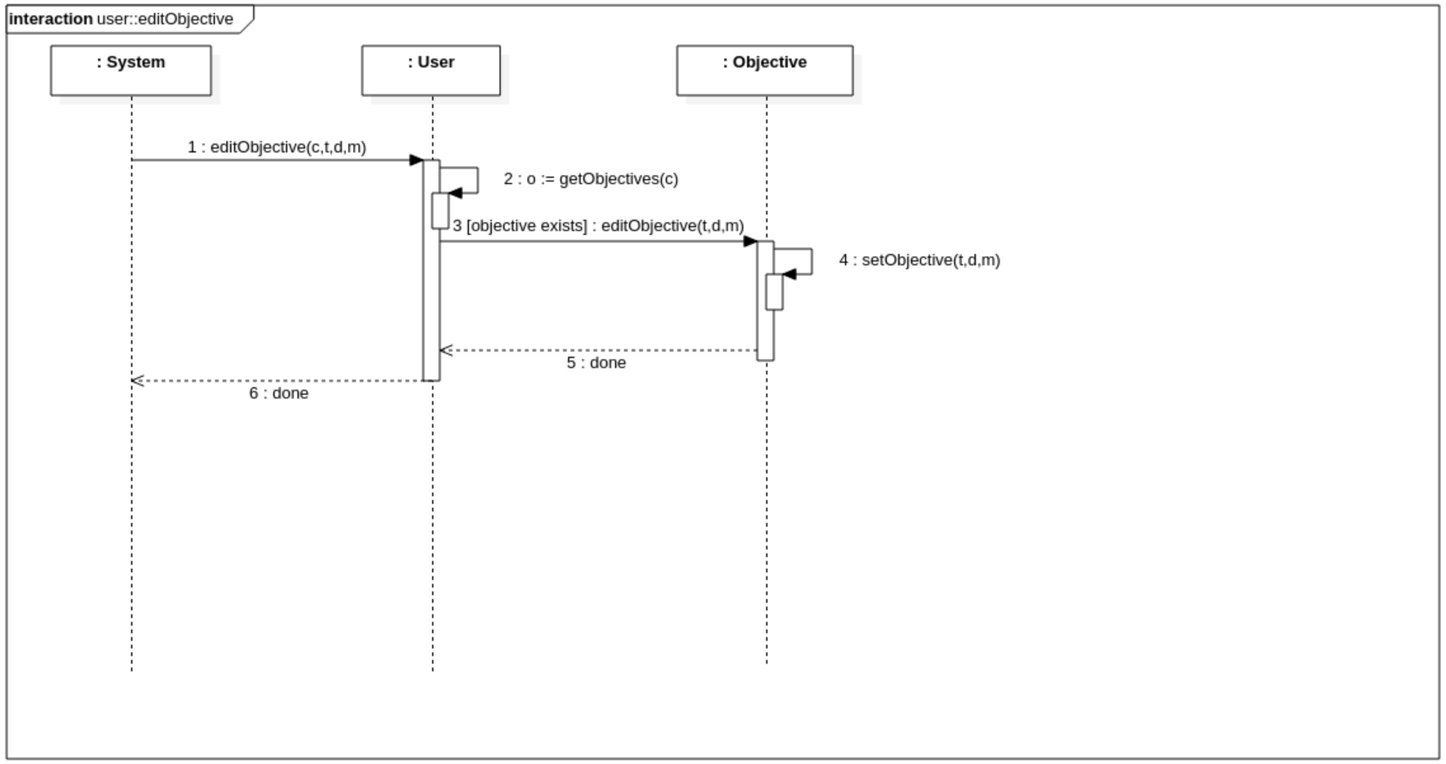
post:

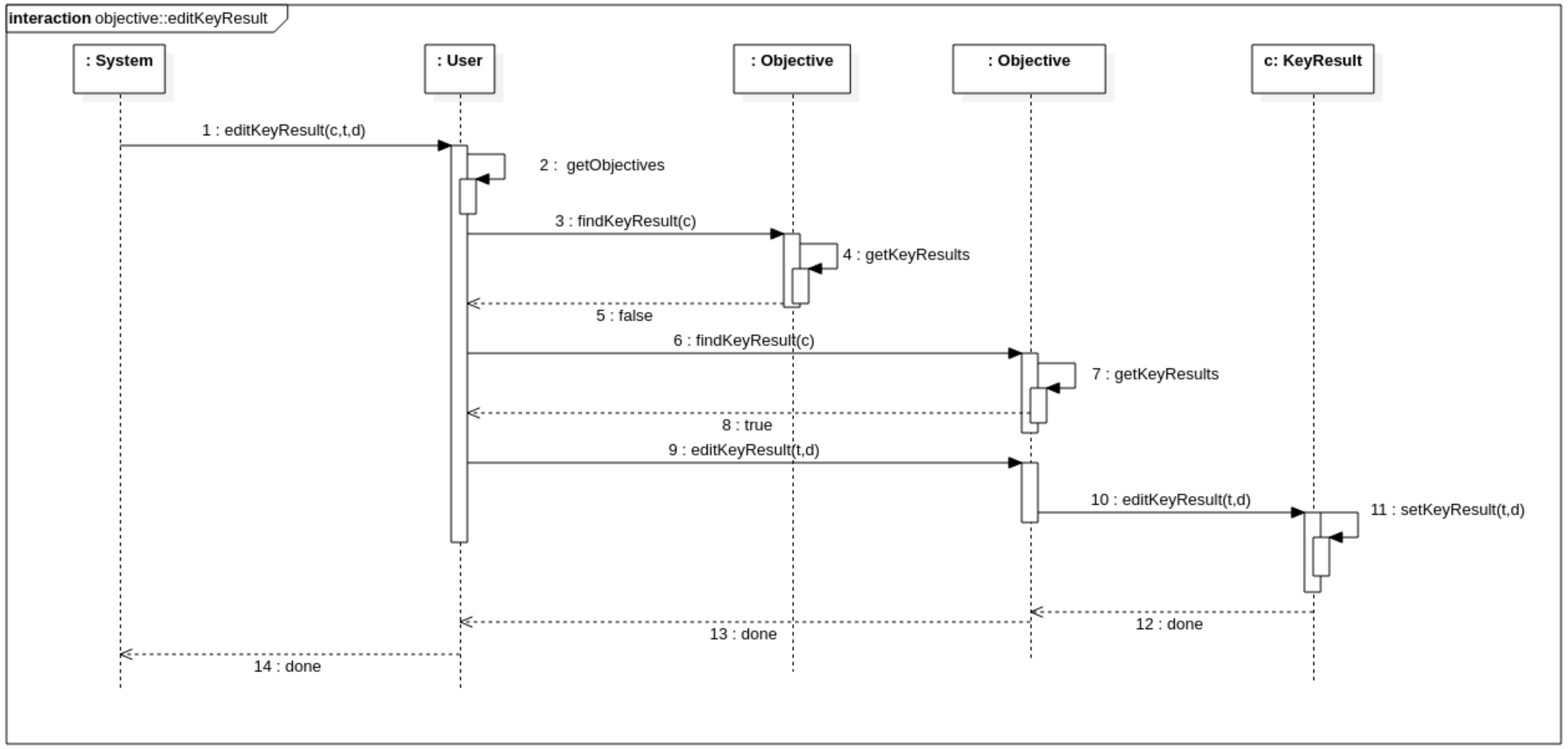
- - a new user object User, will have been created
- - User will be linked to System
- - a string indicating success will have been returned
- - otherwise, a string indicating failure will have been returned

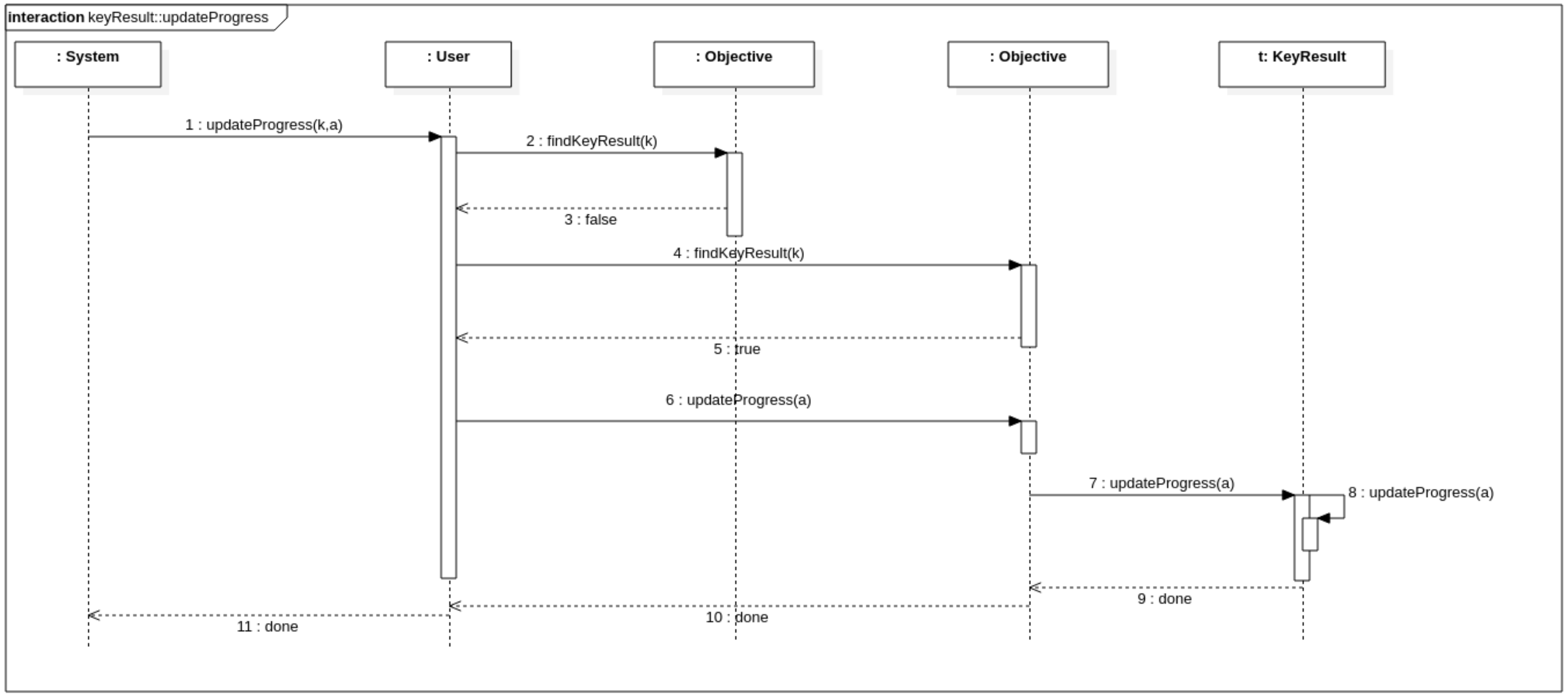
10b. Sequence models

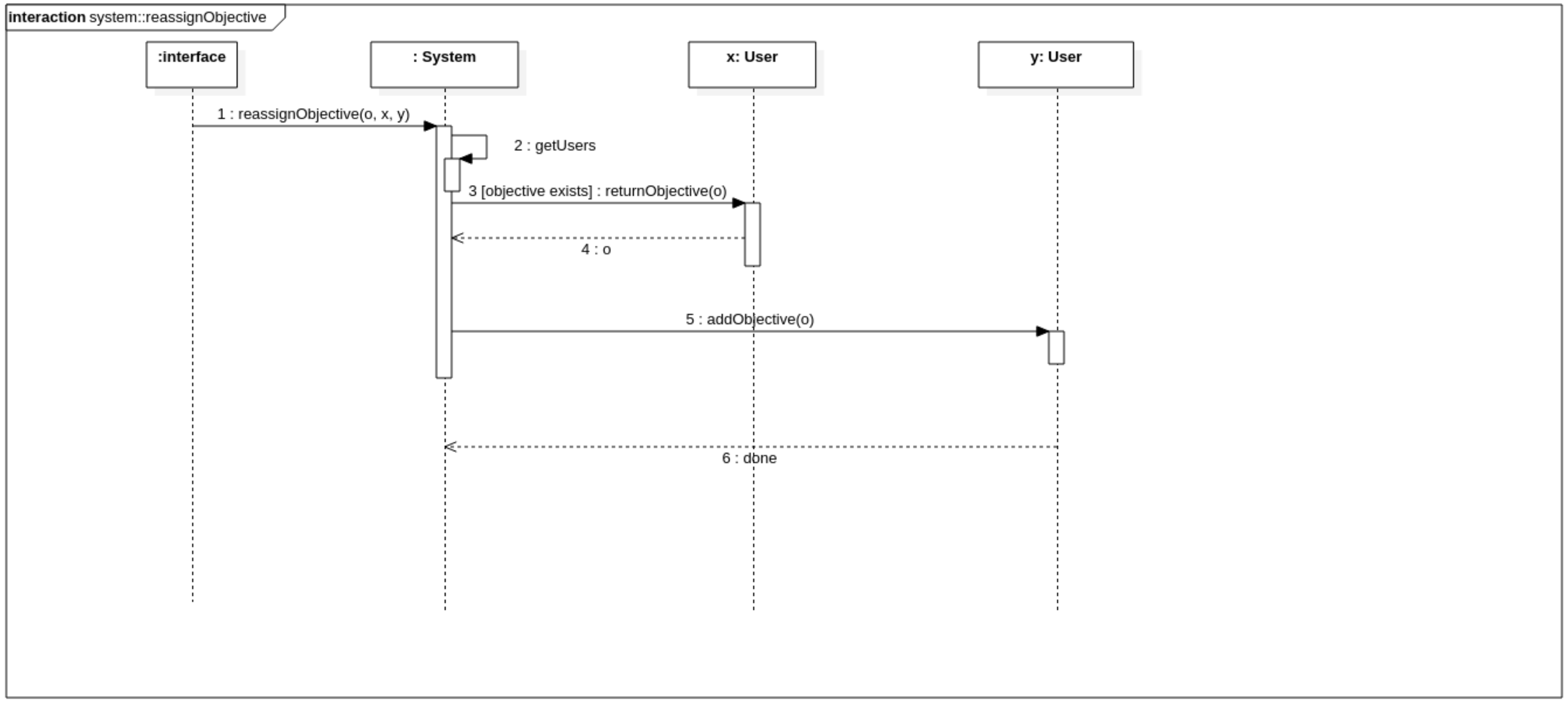

interaction user::newObjective
: System
: User
1 : newObjective(t,d,m)
«create»
2 [milestone not in past] : newObjective(d,m)
t: Objective
3 : done

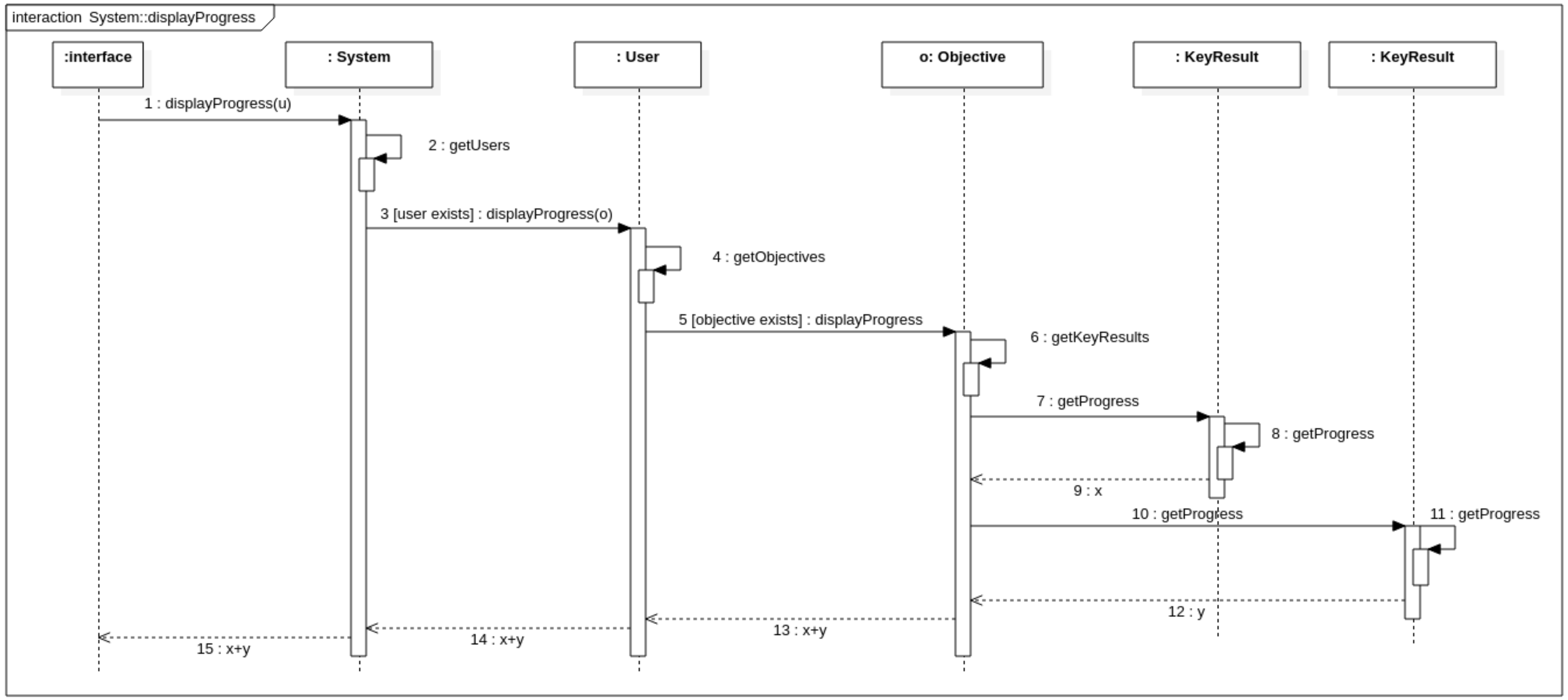

interaction objective::newKeyResult
: System
: User
o: Objective
1 : newKeyResult(c,t,d)
2 : o := getObjectives(c)
3 [objective exists] : newKeyResult(t,d)
4 : m := getMilestone
«create»
5 : newKeyResult(t,d,m)
c: KeyResult
6 : done
7 : done
8 : done

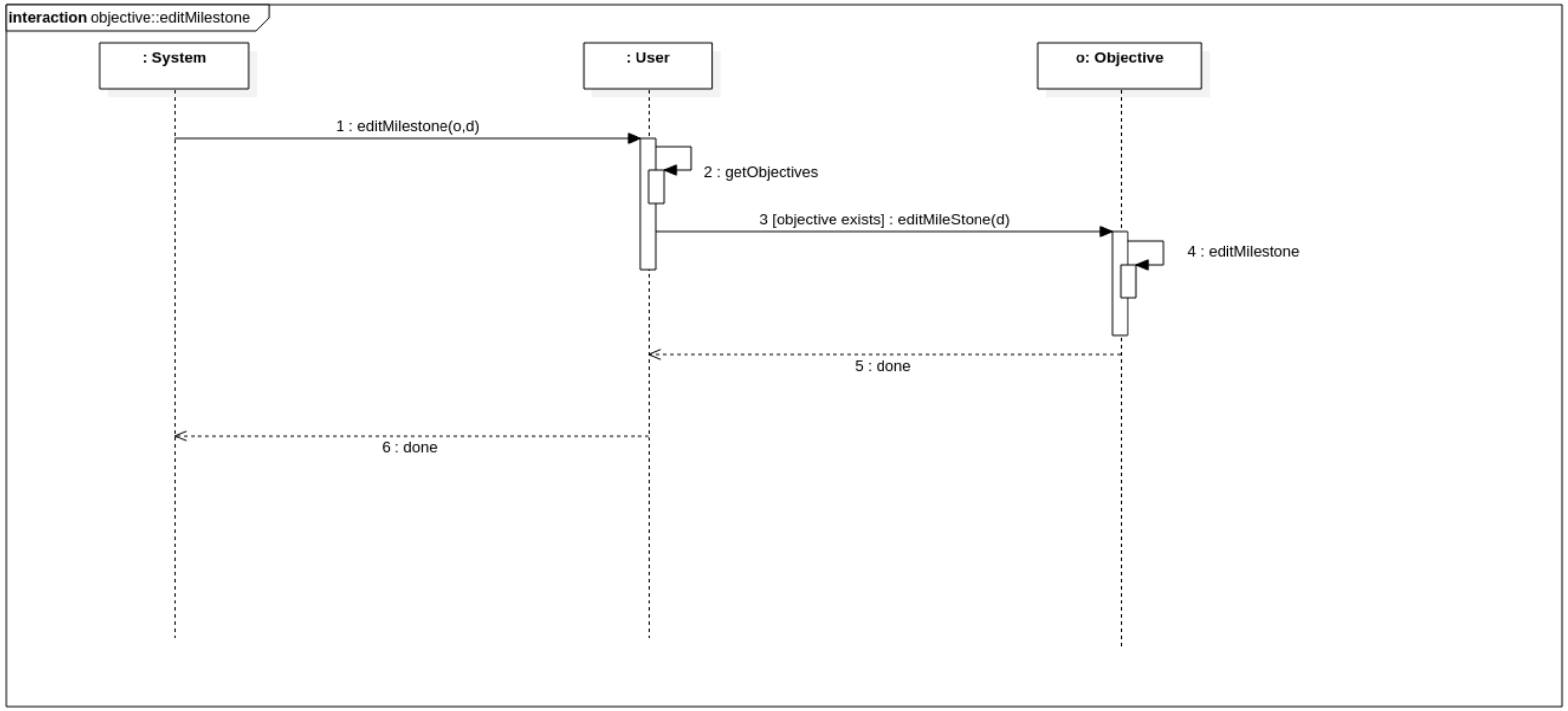

interaction user::editObjective
: System
: User
: Objective
1 : editObjective(c,t,d,m)
2 : o := getObjectives(c)
3 [objective exists] : editObjective(t,d,m)
4 : setObjective(t,d,m)
5 : done
6 : done

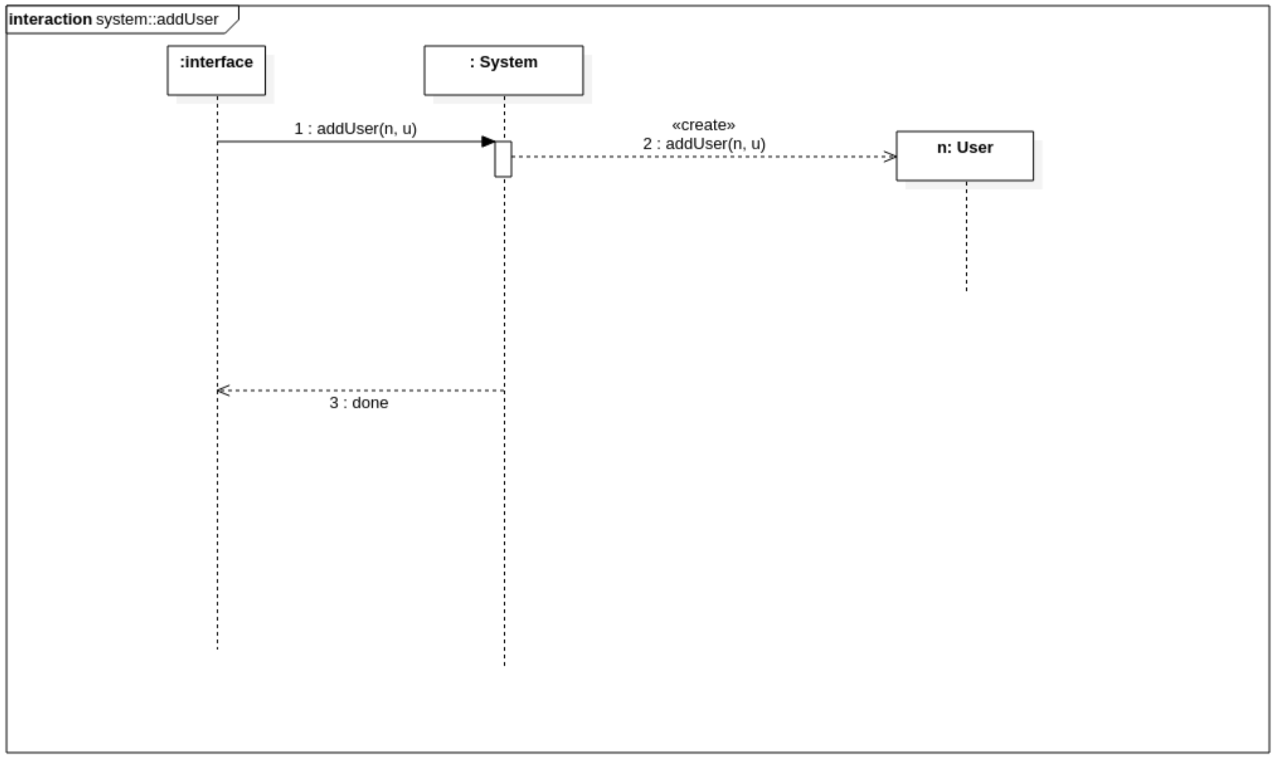

interaction objective::editKeyResult
: System
: User
: Objective
: Objective
c: KeyResult
1 : editKeyResult(c,t,d)
2 : getObjectives
3 : findKeyResult(c)
4 : getKeyResults
5 : false
6 : findKeyResult(c)
7 : getKeyResults
8 : true
9 : editKeyResult(t,d)
10 : editKeyResult(t,d)
11 : setKeyResult(t,d)
12 : done
13 : done
14 : done


interaction keyResult::updateProgress
: System
: User
: Objective
: Objective
t: KeyResult
1 : updateProgress(k,a)
2 : findKeyResult(k)
3 : false
4 : findKeyResult(k)
5 : true
6 : updateProgress(a)
7 : updateProgress(a)
8 : updateProgress(a)
9 : done
10 : done
11 : done

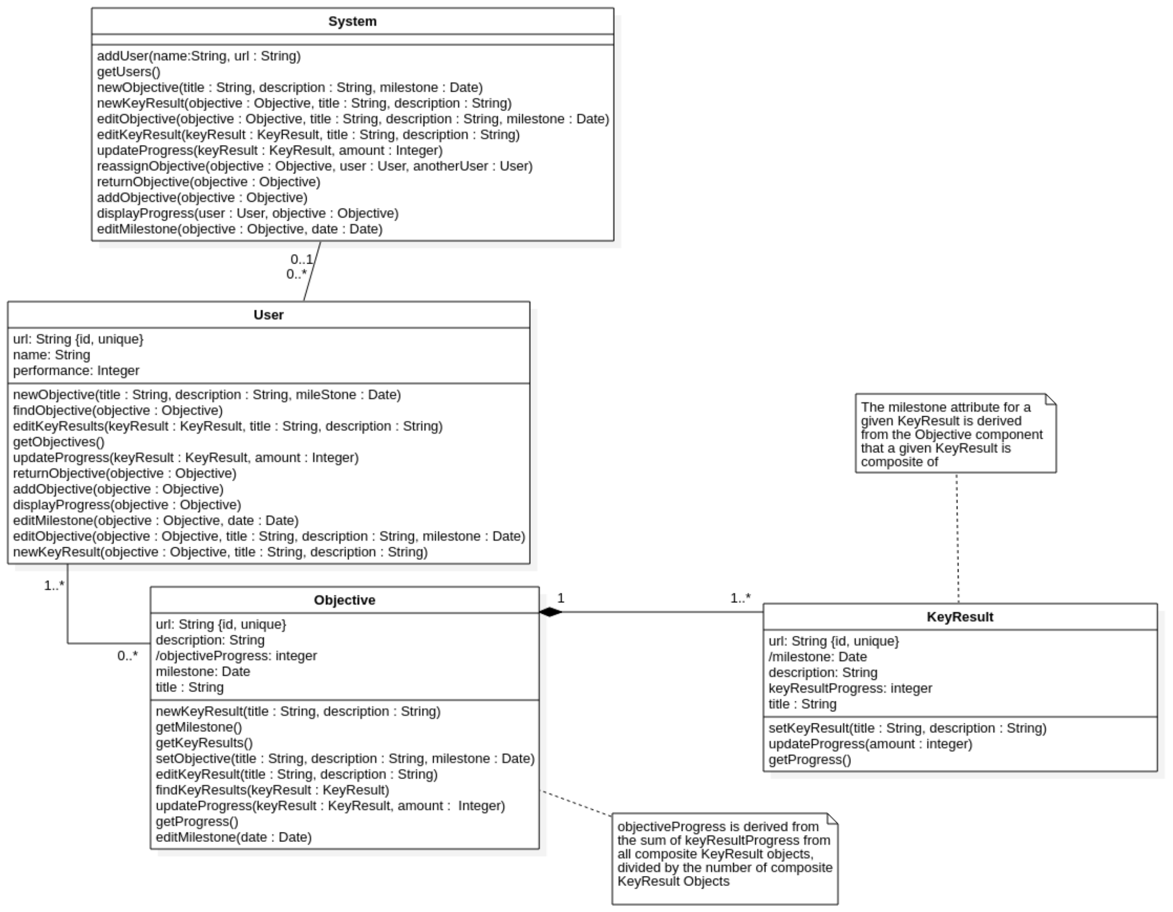

interaction system::reassignObjective
:interface
: System
x: User
y: User
1 : reassignObjective(o, x, y)
2 : getUsers
3 [objective exists] : returnObjective(o)
4 : o
5 : addObjective(o)
6 : done


interaction System::displayProgress
:interface
: System
: User
o: Objective
: KeyResult
: KeyResult
1 : displayProgress(u)
2 : getUsers
3 [user exists] : displayProgress(o)
4 : getObjectives
5 [objective exists] : displayProgress
6 : getKeyResults
7 : getProgress
8 : getProgress
9 : x
10 : getProgress
11 : getProgress
12 : y
13 : x+y
14 : x+y
15 : x+y


interaction objective::editMilestone
: System
: User
o: Objective
1 : editMilestone(o,d)
2 : getObjectives
3 [objective exists] : editMileStone(d)
4 : editMilestone
5 : done
6 : done


interaction system::addUser
:interface
: System
1 : addUser(n, u)
«create»
2 : addUser(n, u)
n: User
3 : done

10c. Design Structural Model


System
addUser(name:String, url : String)
getUsers()
newObjective(title : String, description : String, milestone : Date)
newKeyResult(objective : Objective, title : String, description : String)
editObjective(objective : Objective, title : String, description : String, milestone : Date)
editKeyResult(keyResult : KeyResult, title : String, description : String)
updateProgress(keyResult : KeyResult, amount : Integer)
reassignObjective(objective : Objective, user : User, anotherUser : User)
returnObjective(objective : Objective)
addObjective(objective : Objective)
displayProgress(user : User, objective : Objective)
editMilestone(objective : Objective, date : Date)
0..1
0..*
User
url: String {id, unique}
name: String
performance: Integer
newObjective(title : String, description : String, mileStone : Date)
findObjective(objective : Objective)
editKeyResults(keyResult : KeyResult, title : String, description : String)
getObjectives()
updateProgress(keyResult : KeyResult, amount : Integer)
returnObjective(objective : Objective)
addObjective(objective : Objective)
displayProgress(objective : Objective)
editMilestone(objective : Objective, date : Date)
editObjective(objective : Objective, title : String, description : String, milestone : Date)
newKeyResult(objective : Objective, title : String, description : String)
1..*
0..*
Objective
url: String {id, unique}
description: String
/objectiveProgress: integer
milestone: Date
title : String
newKeyResult(title : String, description : String)
getMilestone()
getKeyResults()
setObjective(title : String, description : String, milestone : Date)
editKeyResult(title : String, description : String)
findKeyResults(keyResult : KeyResult)
updateProgress(keyResult : KeyResult, amount : Integer)
getProgress()
editMilestone(date : Date)
1
1..*
KeyResult
url: String {id, unique}
/milestone: Date
description: String
keyResultProgress: integer
title : String
setKeyResult(title : String, description : String)
updateProgress(amount : integer)
getProgress()
The milestone attribute for a given KeyResult is derived from the Objective component that a given KeyResult is composite of
objectiveProgress is derived from the sum of keyResultProgress from all composite KeyResult objects, divided by the number of composite KeyResult Objects

11. MEAN Stack Evaluation

With the abstract design of a solution captured in the previous chapters of this template, the rest of this template will seek to evaluate developing a solution using a MEAN stack (MongoDB, Express, Angular and Node.js).

The rest of this chapter shall focus on the implementation of the first iteration of this design within a MEAN stack. Of note is that the development done so far is in order to evaluate the proposed solution within a MEAN stack. This is not a fully functional system and system tests will need to be conducted. As a result, logical flaws within the first iteration given are highly likely.

In implementing the design into a MEAN stack the main obstacle that was immediately apparent was how to translate the abstract design into a tangible solution within a MEAN stack. This was primarily due to the fact that in contrast to more classical object orientated languages, such as Java, a class as a concept could not be implemented as one encapsulated file.

When using classical object oriented engineering methods for a system that uses a language such as JavaScript, although the methods prove effective, how the resultant code derives from the design is less apparent. For example, the design structural model and the functions required for each class cannot be easily translated to an Express application. As the concept of a class does not really exist within an Express application. Each one of those functions is an abstract concept as to what a specific combination of Express modules can achieve.

Chapters 11a.ii to 11b.iv will seek to explain how the concept of a class is implemented by discussing the code elements and layout. Whereas, chapter 11a.i will seek to evaluate the pros and cons of a MEAN stack architecture.

A live demonstration of this proposed solution is available here https://tm470-ema.herokuapp.com/modelRouter/user/create

11a. Suggested Architecture

A suggested architecture for a solution that uses a MEAN stack is illustrated within fig 4.

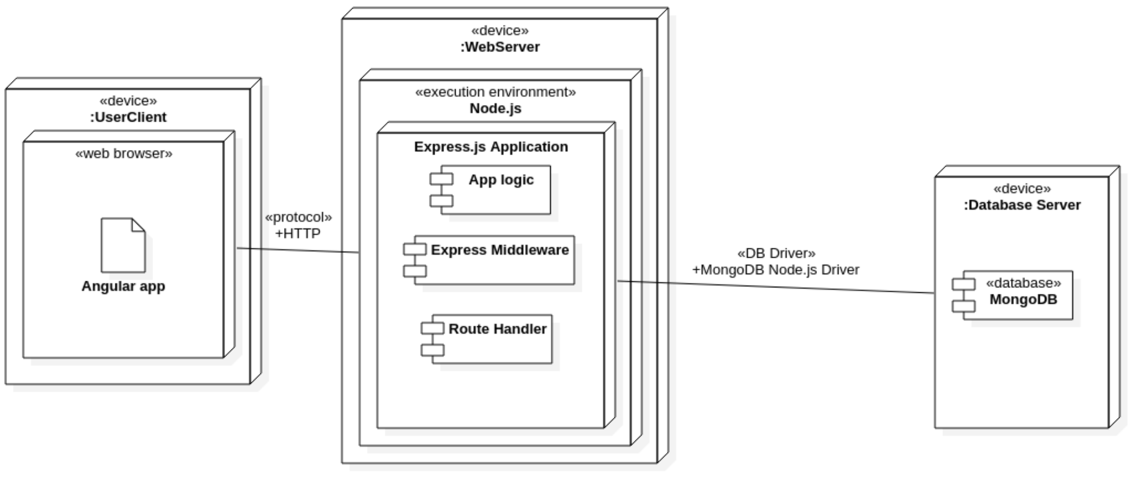


Fig 4

11a.i Node.js and Express evaluation

Introduction

The evaluation of the solution defines a set of criteria that will be used. In exploring these criteria, what the components of a MEAN stack are, and how they operate, shall be articulated. Which will subsequently allow the reader to be in a position to better understand how the proposed solution could be implemented within a MEAN stack architecture.

The selected criteria were placed into two categories: the “ilities” of the system and how the system addresses some of the more common problems faced by development companies.

Looking at the first set of criteria in detail, the following “ilities” were considered.

- Maintainability.
- Scalability.
- Extensibility.
- Security.
- Portability.
- Performance.

Looking at the second set of criteria in detail some of the challenges facing the software industry were considered (CIO, 2019).

- Time to market.
- Required expertise for the system.
- Integration and interfacing issues.
- Acquiring and retaining talent.
- Internal sourcing and outsourcing required for the system.
- Adoption rates.

Some of the latter criteria are intrinsically tied with the "ilities". For instance, time to market is relative to the maintainability, portability, extensibility and scalability of software. The expertise required, and acquiring and retaining talent in relation to the system is relative to nearly all the "ilities" articulated. Integration and interfacing issues are relative to the extensibility of a system. Adoption rates are generally relative to performance as this is the most common complaint from end-users that can be addressed by the system itself.

What is a MEAN stack?

A MEAN stack is an architecture that uses Angular as the front end. Uses Node.js as the server side execution environment, within which Express is the web framework that handles the logic of the system. Finally, MongoDB serves as the back end of the stack, although Node.js and Express can cater for other types of databases.

Node.js and Express are what form the core of a MEAN stack. As is evidential in the prototype, the front end can be replaced with other templating languages. Replacing the front end (as in this instance with Pug) still allows for the main benefits, as articulated within the set criteria, to be present within the stack as a whole.

Although Angular does elevate the benefits of a MEAN stack, the choice to use Pug was a deliberate decision to allow for a prototype to be deployed within the time frame of this project.

What is Node.js?

According to Node.js.org (n.d), Node.js is a server side execution environment that allows users to build scalable network applications quickly. It is written in JavaScript and is open source and cross-platform. Reinforcing this statement, Holmes (2015) highlights that a MEAN stack allows for an end-to-end framework using the best of the best of JavaScript. Ergo, both the client and server side will consist of the same language. The statements made by Node.js.org (n.d) suggest that the criteria of maintainability, portability and scalability can be addressed with the use of Node.js.

One of the benefits of having a system with the vast majority of its components written in the same language is highlighted by Cappelli (2000), who addresses the woes software development companies have when acquiring and retaining talent. In essence, he states that the more niche an employee is, the more at risk a company is of having said employee poached by other companies. He goes on to state companies that recruit employees *“whom can do the job but are not in high demand… may be able to shelter themselves from market forces”.* Therefore, using a system that utilises the same language for both the client and server side address not only the requirement for expertise (as it mitigates against niche specialities), but also the woes of talent being poached.

Equally so, it addresses a common concept known as a “skill-set silo” (Businessdictionary.com, 2019) which within the software development industry refers to teams that do not share goals, tools, priorities and processes with other departments and is detrimental to a company’s overall performance.

In contrast, having the same language within the client and server side of the system will bring together employees whose contribution will span the full development cycle. Thus, allowing for cross-functional teams where each member, drawn from different areas of an organisation, maintain responsibility for the system throughout its entire life cycle. Pinto. M, et al (1993) have attributed cross-functional teams to a company’s overall performance and effectiveness and, therefore, from the perspective of this project a decrease in time to market for products.

In addition, having a whole stack that is written in the same language allows for insourcing from other departments to meet required deadlines. Equally so, owing to the simplicity of JavaScript, the recruitment pool for potential outsourcing tasks becomes larger.

However, owing to the fact that Node.js is unopinionated with little guidance on how the components of the system could be implemented a decrease in productivity is likely to be seen during the initial adoption period of the technology. This is mainly due to the fact that the developers will need to define and agree on good processes for the development and maintenance of code.

This statement, therefore, brings into question the concept of niche specialities. In general, a niche speciality refers to expertise in a specific language within a specific area of a system. With Node.js being unopinionated, a company that adopts said technology and defines its own best practices has, in essence, created its own niche speciality amongst its employees. Two hypothetical companies that adopt a MEAN stack may develop its application in different ways.

Therefore, the notion that a freshly recruited employee will immediately begin to contribute to the company's overall productivity is flawed. As a degree of in house training will be required. However, owing to the simplicity of Node.js said training is likely to consume less time than alternative technologies.

With regard to the concept of cross-functional teams, Santa & Pun (2009), highlight in their study that simply having cross-functional teams does not automatically result in better operational performance. They observed that an increase in operational performance was as a result of quality interaction and communication between teams. Which although the concept of cross-functional facilitates this, it does not enforce it.

Looking at the performance of Node.js and by extension the potential adoption rates of the system, it is an asynchronous, single-threaded, event driven JavaScript runtime, using Google’s V8 engine.

The single-threaded, event-driven architecture approach of Node.js allows for multiple simultaneous connections efficiently. In contrast, a multi-threaded application will create an additional thread for each new process which will consume a hosting machine’s resources.

It achieves this by asynchronously using event loops and callbacks. Ultimately, allowing it to execute non-blocking methods. The most common of which are I/O operations which when run synchronously will block all other events until said operation is completed. In deciding on an architecture for a full stack this is an important consideration with regard to the performance of a system.

Google's V8 is responsible for not only compiling and executing the code but for handling the call stack, memory management, garbage collection and providing the data types, operators, objects and functions (v8.dev, n.d). Owing to the fact that performance is a key driver in the competition between web browsers V8 is constantly being optimised. Therefore, this competition will ultimately benefit developers who use Node.js in their applications.

Regarding scalability, Node.js derives its name from the fact that its developers wanted to emphasis that a Node application comprises of multiple nodes which communicate with one another. This approach is in essence what makes Node.js scalable. Although the scalability of Node.js cannot be achieved by means of the prototype, a number of case studies are available by large enterprise organisations that have use Node.js for its scalability properties, such as NASA (foundation.nodejs.org, n.d) and Netflix (Alex Liu – Scaling A/B testing on Netflix.com with Node.js, 2014).

Node.js’ modular approach equally addresses integration and interfacing issues, as well as security and extensibility. Integration and interfacing issues are addressed by the fact that as alluded to, a MEAN stack utilises the best of the best of JavaScript (Holmes, 2015). Node.js is not restricted to use only the elements of a MEAN stack, there are numerous modules that allow it to integrate and interface with other technologies, which is also what makes Node.js extensible.

Regarding security, Node.js is just a vulnerable as the next programming language, and it is up to the developers to addresses known vulnerabilities. Node.js' module approach allows developers to address these with relative ease with modules that implement protection against known flaws. Within the prototype two such modules have been utilised: Helmet, which sets HTTP headers and provides 14 functions such as enforcing HTTPS, prevent clickjacking and hiding MIME types; and express-validator which provides a number of sanitation functions, for instance, HTML characters are automatically escaped, as well as validation functions such as checking for correct input type. The later can be customised with a built in function that takes specific parameters, for instance, checking if an email already exists or if a password is valid.

The aforementioned modules were specifically highlighted as an example of how this modular approach contributes to productivity and subsequently time to market. The full extent of this will be fully appreciated by someone who has had the pleasure of debugging another developers validation function littered with Regex and is a prime example of common functions that are required within the majority of systems.

Returning back to Node.js' performance. Node.js' ability to work asynchronously does not come packaged in the initial installation. A utility module must be installed with the Node Package Manager and imported into the relevant code logic that will utilise it.

With this in mind, we shall address the flaws of Node.js' modular approach. Although the core Node.js is stable the fact that it is an open source ecosystem results in some packages not being supervised and as a result, being of poor quality or lacking proper documentation.

As alluded to early, the Node Package Manager allows for easy installation of modules, however, it gives no indication of the quality or source of such a package. This, therefore, requires experienced developers who can find and know what tools can be trusted. Which brings us back to the concept of niche specialities and a company’s best practices. Equally so, it highlights that until best practices are established, employee time is likely to be consumed with corroborative research into the system itself.

The implementation

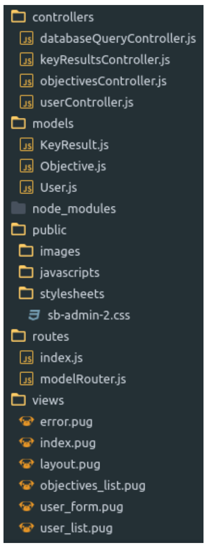
One of the main concerns with developing this proposed solution was whether the design, which follows an object orientated approach, could be implemented within a MEAN stack. Recent updates to JavaScript have made it more object orientated, however, how it achieves this differs from other programming languages. For instance, JavaScript does not allow for access modifiers, however, within a Node.js environment it achieves encapsulation and data-hiding by its modular approach to where code resides within the system.

Exploring this in more detail, as Node.js is event driven this allows for the control flow of a program to be determined by an event. This is generally done with the use of event listeners which in turn trigger event handlers, which subsequently respond with some function that may or may not trigger further events. This is what allows for the encapsulation of code with event listeners only needing to know which event handlers to trigger. Before elaborating on this further what Express is, needs to be articulated.

What is Express?

As articulate within the previous paragraphs, Express is a web framework that allows for a web application to be structured to handle different http requests at a specific URL.

A class within the Express framework is defined by a file for its logic (controller), a separate file that in essence makes the logic within the controller encapsulated (route) and yet another file that essentially defines the persistence of an instance of said class within the database (model). Fig 5 illustrates the file structure for the prototype.



```
controllers
  databaseQueryController.js
  keyResultsController.js
  objectivesController.js
  userController.js
models
  KeyResult.js
  Objective.js
  User.js
node_modules
public
  images
  javascripts
  stylesheets
    sb-admin-2.css
routes
  index.js
  modelRouter.js
views
  error.pug
  index.pug
  layout.pug
  objectives_list.pug
  user_form.pug
  user_list.pug
```

Fig 5

From the folder structure that an Express application requires five files/folders are of particular importance and shall be explored in further detail:

app.js – (not captured within fig 5, however, it sits in the root of the folder structure) which initialises the application and in essence is equivalent to a main.java class.

Controllers – these files are the application logic, with each file corresponding to a class within the Domain Structural Model

Routes – routing refers to determining how an application responds to a clients request which uses a URI. The URI is appended with a specific HTTP request method, such as GET and POST. The route will also define a handler function which is executed if a route is matched

Models – contains files each which will correspond to a database schema

Views – contains files containing the templates that display information to the user.

11b. Code in detail

11b.i. App.js

Looking first at app.js, as alluded to this is the entry point for the application and initialises the app. Lines 1-7 are the middleware imports; Lines 10 & 11 import the route files; Line13 initialises the middleware express, whereas, line 56 makes it public to other modules; Lines 16 – 22 initialise mongoose, which is an API to the MongoDB database that is used for this proposed solution; Lines 25 & 26 set the desired templating language to be used for the user interface; Lines 28 - 34 initialises the middleware that was imported; To begin the inter module communication lines 37 & 38 call the route files.

Calling the route files in this manner is another benefit of using a MEAN stack, as once initiated the execution environment never sleeps and is constantly listening for an event.

```
 1: var createError = require('http-errors')
 2: var express = require('express')
 3: var path = require('path')
 4: var cookieParser = require('cookie-parser')
 5: var logger = require('morgan')
 6: var helmet = require('helmet')
 7: var compression = require('compression')
 8:
 9: // import routes
10: var indexRouter = require('./routes/index')
11: var modelRouter = require('./routes/modelRouter')
12:
13: var app = express()
14:
15: // Set up the mongoose connection
16: var mongoose = require('mongoose')
17: var Url = 'mongodb+srv://jps462:H3l1c0pt3r123@cluster0-
cgcsu.mongodb.net/projectDB?retryWrites=true&w=majority'
18: var mongoDB = process.env.MONGODB_URI || Url;
19: mongoose.connect(mongoDB, { useNewUrlParser: true })
20: mongoose.Promise = global.Promise
21: var db = mongoose.connection
22: db.on('error', console.error.bind(console, 'MongoDB connection error:'))
23:
24: // view engine setup
25: app.set('views', path.join(__dirname, 'views'))
26: app.set('view engine', 'pug')
27:
28: app.use(logger('dev'))
29: app.use(express.json())
30: app.use(express.urlencoded({ extended: false }))
31: app.use(cookieParser())
32: app.use(express.static(path.join(__dirname, 'public')))
33: app.use(helmet())
34: app.use(compression()) // Compress all routes
35:
36: // add the routes
37: app.use('/', indexRouter)
38: app.use('/modelRouter', modelRouter)
39:
40: // catch 404 and forward to error handler
41: app.use(function (req, res, next) {
42:   next(createError(404))
43: })
44:
45: // error handler
46: app.use(function (err, req, res, next) {
47:   // set locals, only providing error in development
48:   res.locals.message = err.message
49:   res.locals.error = req.app.get('env') === 'development' ? err : {}
50:
51:   // render the error page
52:   res.status(err.status || 500)
53:   res.render('error')
54: })
55:
56: module.exports = app
```

```
// anything that is exported becomes pubic.
```

11b.ii. Routes

Two route files are used within this proposed solution:

index.js

The first route file simply redirects to the modelRouter file. Although this may seem redundant, it allows for a different home page to be set and displayed in the future.

```
//node_module imports
var express = require('express')
var router = express.Router()

// GET home page. This calls the modelRouter.
router.get('/', function (req, res) {
  res.redirect('/modelRouter')
})

module.exports = router
```

modelRouter.js

Within this file lines 6 – 9 import the controller files. As articulated above these are the files that deal with the application logic. The remaining of the file defines the URIs and http methods appended to them. Each URI will call a function from the relevant controller. For instance line 14 calls the function `index` from the `databaseQueryContoller`.

```
 1: //node_modules imports
 2: var express = require('express')
 3: var router = express.Router()
 4:
 5: // Required controller modules.
 6: var userController = require('../controllers/userController')
 7: var keyresultsController = require('../controllers/keyResultsController')
 8: var objectivesController = require('../controllers/objectivesController')
 9: var databaseQueryController = require('../controllers/databaseQueryController')
10:
11:
12:
13: // GET the function that returns the number of entries within the database. Which are then
populated on the index page.
14: router.get('/', databaseQueryController.index)
15:
16: /// USER ROUTES ///
17:
18: // GET request for creating USER. NOTE This must come before route for id (i.e. display user).
19: router.get('/user/create', userController.user_create_get)
20:
21: // POST request for creating USER.
22: router.post('/user/create', userController.user_create_post)
23:
24: // GET request for list of all USERs.
25: router.get('/users', userController.user_list)
26:
27: /// KeyResults ROUTES ///
28:
29: // GET request for creating a KeyResults. NOTE This must come before route that displays
KeyResults (uses id).
30: router.get('/KeyResult/create', keyresultsController.keyresult_create_get)
31:
32: // POST request for creating KeyResults.
33: router.post('/KeyResult/create', keyresultsController.keyresult_create_post)
34:
35:
36: // GET request for list of all KeyResults.
37: router.get('/KeyResults', keyresultsController.keyresult_list)
38:
39: /// OBJECTIVES ROUTES ///
40:
```

```
// GET request for creating an objective. NOTE This must come before route that displays objectives (uses id).
router.get('/objectives/create', objectivesController.objectives_create_get)

// POST request for creating objectives.
router.post('/objectives/create', objectivesController.objectives_create_post)

// GET request for list of all objectives.
router.get('/objectives', objectivesController.objectives_list)

module.exports = router
```

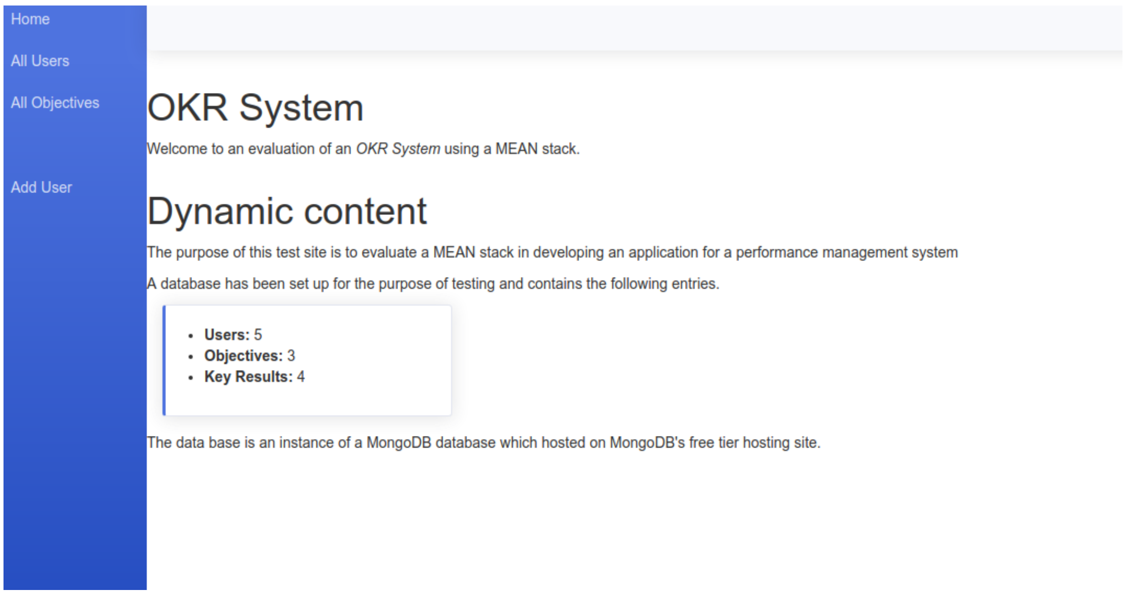
11b.iii. Controllers

Moving forward the controllers shall be explored, starting with the highlighted function called from the `modelRouter` to the `databaseQueryController`.

databaseQueryController.js

Within this controller file, lines 5 – 7 import the model schemas that have been defined within the model files. Line 9 is the `index` function that was called in the previously explored `modelRouter` file. This function calls MongoDB's count function via the model files that were imported. Finally, on line 21 it renders the data using the index.pug file. This whole process tests all elements of the MEAN stack and produces a result as illustrated in Fig 6, which indicates the total number of entries within the MongoDB database. For the purpose of this evaluation, MongoDB's free tier hosting site was used.

```
 1: // Here we import some middleware that allows for asynchronous operations
 2: var async = require('async')
 3:
 4: // These are the imports that are require for this app to function.
 5: var User = require('../models/User')
 6: var Objective = require('../models/Objective')
 7: var KeyResult = require('../models/KeyResult')
 8:
 9: exports.index = function (req, res) {
10:     async.parallel({
11:       user_count: function (callback) {
12:         User.countDocuments({}, callback) // Pass an empty object as match condition to find all documents of this collection
13:       },
14:       objective_count: function (callback) {
15:         Objective.countDocuments({}, callback)
16:       },
17:       keyResult_count: function (callback) {
18:         KeyResult.countDocuments({}, callback)
19:       },
20:
21:     }, function (err, results) {
22:       res.render('index', { title: 'OKR System', error: err, data: results })
23:     })
24:   }
```



OKR System

Welcome to an evaluation of an *OKR System* using a MEAN stack.

Dynamic content

The purpose of this test site is to evaluate a MEAN stack in developing an application for a performance management system

A database has been set up for the purpose of testing and contains the following entries.

- **Users:** 5
- **Objectives:** 3
- **Key Results:** 4

The data base is an instance of a MongoDB database which hosted on MongoDB's free tier hosting site.

Fig 6

11b.iv. Models

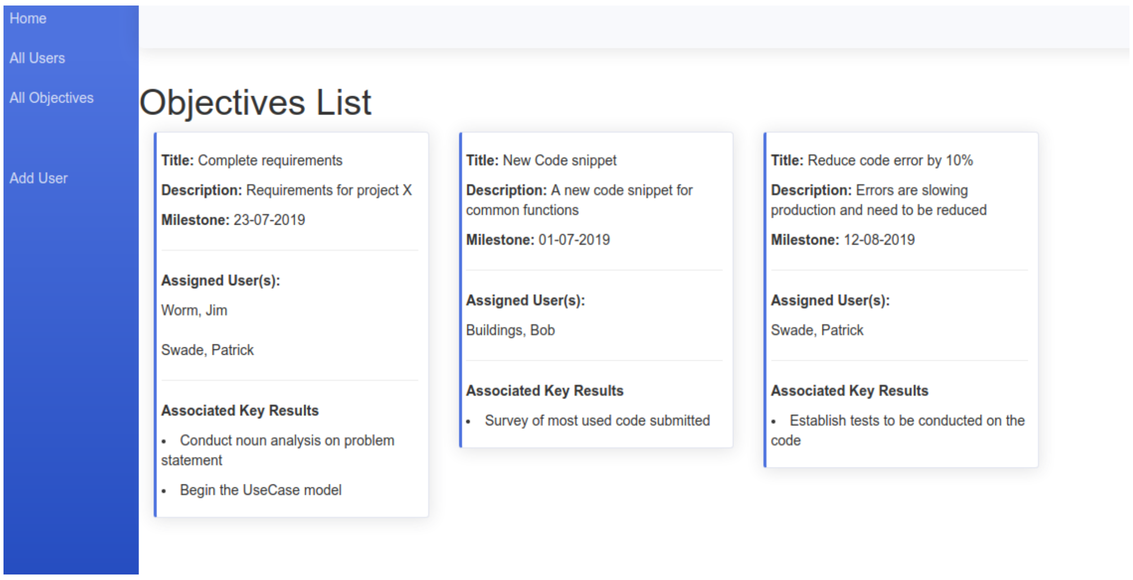
User Model

Within this file, the middleware mongoose is imported and the schema for a `User` object (lines 3 – 10) is defined. Line 13 & 20 are virtual properties of a database entry. This allows for client side manipulation of the data. For instance, future interactions of this proposed solution could use a virtual property for an `Objective` Class' property `objectiveProgress` in which as alluded to within the data dictionary Para 7b, is the total progress of composite `keyResults` divided by the total number of composite `keyResults` for a given `objective`.

```
 1: var mongoose = require('mongoose')
 2: var Schema = mongoose.Schema
 3: var UserSchema = new Schema(
 4:   {
 5:     first_name: { type: String, required: true, max: 100 },
 6:     family_name: { type: String, required: true, max: 100 },
 7:     objectives: [{ type: Schema.ObjectId, ref: 'Objective' }],
 8:     performance: { type: Number }
 9:   }
10: )
11:
12: // Virtual for users' full name
13: UserSchema
14:   .virtual('name')
15:   .get(function () {
16:     return this.family_name + ', ' + this.first_name
17:   })
18:
19: // Virtual for users' URL
20: UserSchema
21:   .virtual('url')
22:   .get(function () {
23:     return '/modelRouter/user/' + this._id
24:   })
25:
26: // Export model
27: module.exports = mongoose.model('User', UserSchema)
```

11b.v. Views and inter module communication from user interactions

Here inter module communication will be explored in further detail. Fig 7 is the display that results from a user selecting the “All Objectives” link.



Objectives List

Title: Complete requirements
Description: Requirements for project X
Milestone: 23-07-2019

Assigned User(s):
Worm, Jim
Swade, Patrick

Associated Key Results
- Conduct noun analysis on problem statement
- Begin the UseCase model

Title: New Code snippet
Description: A new code snippet for common functions
Milestone: 01-07-2019

Assigned User(s):
Buildings, Bob

Associated Key Results
- Survey of most used code submitted

Title: Reduce code error by 10%
Description: Errors are slowing production and need to be reduced
Milestone: 12-08-2019

Assigned User(s):
Swade, Patrick

Associated Key Results
- Establish tests to be conducted on the code

Fig 7

This particular function was implement as it demonstrates an event flow that navigates the full stack. Ergo from the interface via the application logic to the database and back. Illustration 1 gives an abstract view of the event trace that shall be followed, although not technically accurate, this level of abstraction suffices for the following evaluation.

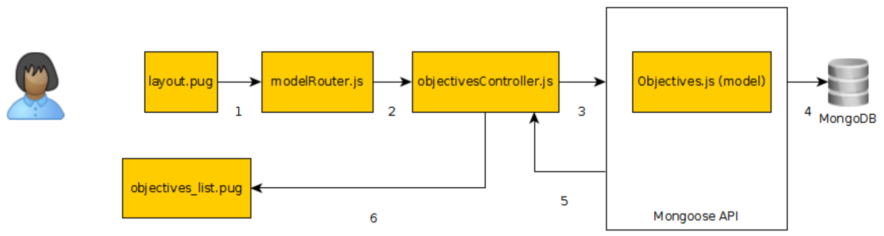


Illustration 1

Step 1

The user is displayed with a view that contains navigational links to list the objectives. This is provided with the `layout.pug` file. Note that this evaluation does not seek to elaborate on Pug, however, to avoid potential confusion Pug is templating engine, primarily used for server-side templating in NodeJS and uses indentation to define HTML elements.

```
    body
          div(id="content-wrapper")
            block sidebar
              ul(class='navbar-nav bg-gradient-primary sidebar sidebar-dark accordion')
                li(class='nav-item')
                  a(class='nav-link', href='/modelRouter') Home
                li(class='nav-item')
                  a(class='nav-link', href='/modelRouter/users') All Users
                li(class='nav-item')
```

```
23:                   a(class='nav-link', href='/modelRouter/objectives') All Objectives
24:                 hr
25:                 li(class='nav-item')
26:                   a(class='nav-link', href='/modelRouter/user/create') Add User
```

Of note is that each link has a reference to the `modelRouter` and provides a URI. Of relevance to this demonstration line 23 calls the `objectives_list` function.

Step 2

This particular URI is listed in the `modelRouter` in line 48 below. As the URI given matches a function, this function is called within the `objectivesController`.

```
47: // GET request for list of all objectives.
48: router.get('/objectives', objectivesController.objectives_list)
```

Step 3, 4, 5 & 6

This particular function is listed on line 5. Steps 4 & 5 are then facilitated by the Mongoose API which is accessed via the import on line 2 (see Objectives Model below for details). Looking at line 6 this calls a function to find the relevant data from the database. Lines 7 & 8 allows for recursive data to be returned. In this case `assignedEmployees` and `keyResults` are relational elements from the `User` and `KeyResult` document databases.

Within the Objectives document database, these are recorded by their `ObjectID`. The populate function allows for their actual details to be returned.

Finally, the `objectivesController` renders the data within the `objectives_list.pug` view, line 12 (note the express allows for file extensions to be omitted and will automatically decide on the files type by the context. In this case, the function `render` is called, therefore, Express can differentiate between `objectivesController.js` and `objectivesController.pug`

objectivesController.js

```
 2: var Objective = require('../models/Objective')
 3:
 4: // Display list of all Objectives.
 5: exports.objectives_list = function (req, res, next) {
 6:   Objective.find({}, 'title description milestone assignedEmployees keyResults')
 7:     .populate('assignedEmployees')
 8:     .populate('keyResults')
 9:     .exec(function (err, listObjct) {
10:       if (err) { return next(err) }
11:       // Successful, so render
12:       res.render('objectives_list', { title: 'Objectives List', objectives_list: listObjct })
13:     })
14: }
```

objectives model

```
 6: var ObjectiveSchema = new Schema(
 7:   {
 8:     title: { type: String, required: true },
 9:     description: { type: String, required: true },
10:     assignedEmployees: [{ type: Schema.ObjectId, ref: 'User' }],
11:     milestone: { type: Date, required: true },
12:     // objectiveProgress :{},
13:     keyResults: [{ type: Schema.ObjectId, ref: 'KeyResult' }]
14:   }
15: )
```

objectives_view.pug

```
  div
    - objectives_list.sort(function(a, b) {let textA = a.title.toUpperCase(); let textB =
b.title.toUpperCase(); return (textA < textB) ? -1 : (textA > textB) ? 1 : 0;});
    each objct in objectives_list
      div(class="col-xl-3 col-md-6 mb-4")
        div(class="card border-left-primary shadow h-100 py-2")
          div(class="card-body")
            div(class="row align-items-center")
              div(class="col mr-2")
                p #[strong Title: ]#{objct.title}
                p #[strong Description: ]#{objct.description}
                p #[strong Milestone: ]#{objct.mileStone}
                hr
                p #[strong Assigned User(s): ]
                each val in objct.assignedEmployees
                  dl
                    dd #{val.name}
                hr
                p #[strong Associated Key Results]
                each kr in objct.keyResults
                  ul
                  li #{kr.title}
```

Conclusion

With a better understanding of how an application written in Express operates, the report will seek to address considerations and concerns with implementing the proposed design into a MEAN stack architecture.

As is evidential within the prototype, Express handles data-hiding and encapsulation in a different way to more classical object-orientated programming languages. More obviously is how it allows for Objects, Messaging, Methods and Abstraction. However, how does it handle other characteristics of object-oriented programming?

From the case studies examined during this project, Express applications were generally written in a hybrid of functional, object-orientated and procedural styles. Equally so, functions are objects that can be created outside of a class. Therefore, although implementing classes is possible within Express, it is likely to add overhead and complexity that is not necessarily required.

In implementing the solution into Express, the classes identified within the design phase were used as an abstract idea of what a combination of specific Model, Controller and Route files could achieve functionally.

Looking at other aspects of object-orientated programming adopting polymorphism is likely to be detrimental to how Express handles data-hiding via the unique URIs within a route file. Therefore, by extension inheritance is not necessarily needed.

Looking at composition, on the other hand, Express employs functional composition. Where the results of one function are pipelined into another to create a new function altogether. The `objectives_list` function gives an example of this by drawing on three separate models, executing the `populate` function to create a data object that is passed to the view and rendered.

With regard to using a Volere template to develop an application within a MEAN stack. When using a language such as JavaScript within an Express framework, although the methods prove effective, how the resultant code derives from the design is less apparent. For example, the design structural model defines the functions required for each class.

However, as the concept of a class does not really exist within an Express application. Each one of those functions is an abstract concept as to what a specific combination of Controller, Model and Route files will achieve.

Therefore, in adopting a MEAN stack technology, the use of more classical development processes are likely to be more of a hindrance than serving their intended purpose.

Source Making (2019), articulates a common antipattern within the software development industry known as the “The Grand Old Duke of York” which in essence captures the frictions between abstractionist and implementationists. In essence, abstractionists are comfortable discussing software design concepts. Whereas, implementationists, often require source code examples before they can grasp abstract concepts.

He goes on to state that implementationists outnumber abstractionist within the software development industry, therefore, the use of complex abstract models could be detrimental to a company’s performance as more time will be required in clarifications and explanations.

References

Alex Liu – Scaling A/B testing on Netflix.com with Node.js (2014) YouTube video, added by Cian O'Maidin [Online]. Available at https://www.youtube.com/watch?v=gtjzjiTI96c& (Accessed 23 August 2019).

Businessdictionary.com (2019) *Silo Mentality Definition* [Online]. Available at http://www.businessdictionary.com/definition/silo-mentality.html (Accessed 23 Aug 2019).

Cappelli, P (2000) *A Market-Driven Approach to Retaining Talent*. Harvard Business Review, vol. 78, no. 1, 2000, p. 103. Gale Academic Onefile [Online]. Available at https://link-gale-com.libezproxy.open.ac.uk/apps/doc/A59023945/AONE?u=tou&sid=AONE&xid=fa8774fe (Accessed 19 August 2019).

CIO (2019) 8 Challenges affecting software project management. [Online]. Available at https://www.cio.com/article/3065984/8-challenges-affecting-software-project-management.html (Accessed 18 August 2019).

foundation.nodejs.org (n.d) *Node.js Helps NASA Keep Astronauts Safe and Data Accessible* [Online]. Available at https://foundation.nodejs.org/wp-content/uploads/sites/50/2017/09/Node_CaseStudy_Nasa_FNL.pdf (Accessed 23 Aug 2019).

Holmes, S. (2015) *Getting MEAN With Mongo, Express, Angular, and Node*. Shelter Island, NY, Manning Publications [Online]. Available at https://learning-oreilly-com.libezproxy.open.ac.uk/library/view/getting-mean-with/9781617292033/kindle_split_000.html (Accessed 29 March 2019).

Nodejs.org (n.d) *About Node.js* [Online]. Available at https://nodejs.org/en/about/ (Accessed 17 August 2019).

Pinto, M., Pinto, J., & Prescott, J. (1993). 'Antecedents and Consequences of Project Team Cross-Functional Cooperation', *Management Science, 39*(10), 1281-1297. [Online]. Available at http://www.jstor.org.libezproxy.open.ac.uk/stable/2632967 (Accessed 23 August 2019).

Santa, R & Pun, W.K.D (2009) 'The impact of cross-functional teams on operational performance after the implementation of intelligent information systems' *2009 IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications.* Rende, Italy, 21-23 Sept. IEEE [Online]. Available at DOI: 10.1109/IDAACS.2009.5342921 (Accessed 24 August 2019)

Source Making (2019) *Software Architectural Antipatterns* [Online]. Available at https://sourcemaking.com/antipatterns/jumble (Accessed 22 Jun 2019).

v8.dev (n.d) *Documentation* [Online]. Available at https://v8.dev/docs (Accessed 20 Aug 2019).

Annexe B – Code in Detail

App.js

```
var createError = require('http-errors')
var express = require('express')
var path = require('path')
var cookieParser = require('cookie-parser')
var logger = require('morgan')
var helmet = require('helmet')
var compression = require('compression')

// import routes
var indexRouter = require('./routes/index')
var modelRouter = require('./routes/modelRouter')

var app = express()

// Set up the mongoose connection
var mongoose = require('mongoose')
var Url = 'mongodb+srv://jps462:H3l1c0pt3r123@cluster0-
cgcsu.mongodb.net/projectDB?retryWrites=true&w=majority'
var mongoDB = process.env.MONGODB_URI || Url;
mongoose.connect(mongoDB, { useNewUrlParser: true })
mongoose.Promise = global.Promise
var db = mongoose.connection
db.on('error', console.error.bind(console, 'MongoDB connection error:'))

// view engine setup
app.set('views', path.join(__dirname, 'views'))
app.set('view engine', 'pug')

app.use(logger('dev'))
app.use(express.json())
app.use(express.urlencoded({ extended: false }))
app.use(cookieParser())
app.use(express.static(path.join(__dirname, 'public')))
app.use(helmet())
app.use(compression()) // Compress all routes

// add the routes
app.use('/', indexRouter)
app.use('/modelRouter', modelRouter)

// catch 404 and forward to error handler
app.use(function (req, res, next) {
next(createError(404))
})

// error handler
app.use(function (err, req, res, next) {
// set locals, only providing error in development
res.locals.message = err.message
res.locals.error = req.app.get('env') === 'development' ? err : {}

// render the error page
res.status(err.status || 500)
res.render('error')
})

module.exports = app
// anything that is exported becomes pubic.
```

Controllers – databaseQueryController.js

```
// Here we import some middleware that allows for asynchronous operations
var async = require('async')

// These are the imports that are require for this app to function.
var User = require('../models/User')
var Objective = require('../models/Objective')
var KeyResult = require('../models/KeyResult')

exports.index = function (req, res) {
async.parallel({
user_count: function (callback) {
User.countDocuments({}, callback) // Pass an empty object as match condition to find all documents
of this collection
},
objective_count: function (callback) {
Objective.countDocuments({}, callback)
},
keyResult_count: function (callback) {
KeyResult.countDocuments({}, callback)
},
}, function (err, results) {
res.render('index', { title: 'OKR System', error: err, data: results })
})
}
```

Controllers – KeyResultsController.js

```
//Imports that the app requires to function
var KeyResult = require('../models/KeyResult')

// Display list of all KeyResults.
exports.keyresult_list = function (req, res) {
res.send('The keyresult_list function has not yet been implemented. This however proves that the
route works successfully.')
}

// Display KeyResult create form on GET.
exports.keyresult_create_get = function (req, res) {
res.send('The keyresult_create_get function has not yet been implemented. This however proves that
the route works successfully.')
}

// Handle KeyResult create on POST.
exports.keyresult_create_post = function (req, res) {
res.send('The keyresult_create_post function has not yet been implemented. This however proves that
the route works successfully.')
}
```

Controller – objectivesController.js

```
// Imports that the app requires to function
var Objective = require('../models/Objective')

// Display list of all Objectives.
exports.objectives_list = function (req, res, next) {
Objective.find({}, 'title description milestone assignedEmployees keyResults')
.populate('assignedEmployees')
.populate('keyResults')
.exec(function (err, listObjct) {
if (err) { return next(err) }
// Successful, so render
res.render('objectives_list', { title: 'Objectives List', objectives_list: listObjct })
})
}

// Display Objectives create form on GET.
exports.objectives_create_get = function (req, res) {
res.send('NOT IMPLEMENTED: Objectives create GET')
}

// Handle Objectives create on POST.
exports.objectives_create_post = function (req, res) {
res.send('NOT IMPLEMENTED: Objectives create POST')
}
```

Controller – userController.js

```
// These are the imports that are require for this app to function.
var User = require('../models/User')

const { body,validationResult } = require('express-validator')
const { sanitizeBody } = require('express-validator')

// Display a list of Users.
exports.user_list = function (req, res, next) {
User.find()
.populate('objectives')
.sort([['family_name', 'ascending']])
.exec(function (err, listUser) {
if (err) { return next(err) }
// Successful, so render
res.render('user_list', { title: 'User List', user_list: listUser }) //This will send the results of
this database query to the user_list.pug view
})

}

// Display User create form on GET.
exports.user_create_get = function (req, res, next) {
res.render('user_form', { title: 'Add a user' })
}

// Handle User create on POST.
exports.user_create_post = [

// Validate fields.
body('first_name').isLength({ min: 1 }).trim().withMessage('First name must be given.')
.isAlphanumeric().withMessage('First name has non-alphanumeric characters.'),
body('family_name').isLength({ min: 1 }).trim().withMessage('Family name must be given.')
.isAlphanumeric().withMessage('Family name has non-alphanumeric characters.'),

// Sanitize fields.
sanitizeBody('first_name').escape(),
sanitizeBody('family_name').escape(),

// Process request after validation and sanitization.
(req, res, next) => {

// Extract the validation errors from a request.
const errors = validationResult(req);
// Create User object with escaped and trimmed data
var user = new User(
{
first_name: req.body.first_name,
family_name: req.body.family_name,
}
)

if (!errors.isEmpty()) {
// There are errors. Render form again with sanitized values/errors messages.
res.render('user_form', { title: 'New User', user: user, errors: errors.array() })
return;
}
else {
// Data from form is valid.

// Save new user to database.
user.save(function (err) {
if (err) { return next(err) }
// Successful - redirect to home page.
res.redirect("/modelRouter")
})
}
}
];
```

Models – KeyResults.js

```
var mongoose = require('mongoose')

var Schema = mongoose.Schema

var keyresultschema = new Schema(
{
title: { type: String, required: true },
description: { type: String, required: true },
progress: { type: Number },
objectives: { type: Schema.Types.ObjectId, ref: 'Objective' }
}
)

// Virtual for KeyResults's URL
keyresultschema
.virtual('url')
.get(function () {
return '/modelRouter.js/KeyResult/' + this._id
})

// Export model
module.exports = mongoose.model('KeyResult', keyresultschema)
```

Models – Objectives.js

```
var mongoose = require('mongoose')
var moment = require('moment') // return formatted dates

var Schema = mongoose.Schema

var ObjectiveSchema = new Schema(
{
title: { type: String, required: true },
description: { type: String, required: true },
assignedEmployees: [{ type: Schema.ObjectId, ref: 'User' }],
milestone: { type: Date, required: true },
// objectiveProgress :{},
keyResults: [{ type: Schema.ObjectId, ref: 'KeyResult' }]
}
)

// Virtual for Objective's URL
ObjectiveSchema
.virtual('url')
.get(function () {
return '/modelRouter.js/objectives/' + this._id
})

ObjectiveSchema
.virtual('mileStone')
.get(function () {
return moment(this.milestone).format('DD-MM-YYYY');
})

// Export model
module.exports = mongoose.model('Objective', ObjectiveSchema)
```

Models – User.js

```
var mongoose = require('mongoose')
var Schema = mongoose.Schema
var UserSchema = new Schema(
{
first_name: { type: String, required: true, max: 100 },
family_name: { type: String, required: true, max: 100 },
objectives: [{ type: Schema.ObjectId, ref: 'Objective' }],
performance: { type: Number }
}
)

// Virtual for users' full name
UserSchema
.virtual('name')
.get(function () {
return this.family_name + ', ' + this.first_name
})

// Virtual for users' URL
UserSchema
.virtual('url')
.get(function () {
return '/modelRouter/user/' + this._id
})

// Export model
module.exports = mongoose.model('User', UserSchema)
```

Routes – index.js

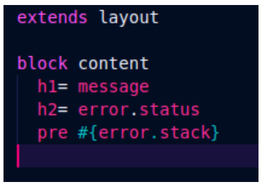
```
//node_module imports
var express = require('express')
var router = express.Router()

// GET home page. This calls the modelRouter.
router.get('/', function (req, res) {
res.redirect('/modelRouter')
})

module.exports = router
```

Routes – modelRouter.js

```
//node_modules imports
var express = require('express')
var router = express.Router()

// Required controller modules.
var userController = require('../controllers/userController')
var keyresultsController = require('../controllers/keyResultsController')
var objectivesController = require('../controllers/objectivesController')
var databaseQueryController = require('../controllers/databaseQueryController')


// GET the function that returns the number of entries within the database. Which are then populated
on the index page.
router.get('/', databaseQueryController.index)

/// USER ROUTES ///

// GET request for creating USER. NOTE This must come before route for id (i.e. display user).
router.get('/user/create', userController.user_create_get)

// POST request for creating USER.
router.post('/user/create', userController.user_create_post)

// GET request for list of all USERs.
router.get('/users', userController.user_list)

/// KeyResults ROUTES ///

// GET request for creating a KeyResults. NOTE This must come before route that displays KeyResults
(uses id).
router.get('/KeyResult/create', keyresultsController.keyresult_create_get)

// POST request for creating KeyResults.
router.post('/KeyResult/create', keyresultsController.keyresult_create_post)

// GET request for list of all KeyResults.
router.get('/KeyResults', keyresultsController.keyresult_list)

/// OBJECTIVES ROUTES ///

// GET request for creating an objective. NOTE This must come before route that displays objectives
(uses id).
router.get('/objectives/create', objectivesController.objectives_create_get)

// POST request for creating objectives.
router.post('/objectives/create', objectivesController.objectives_create_post)

// GET request for list of all objectives.
router.get('/objectives', objectivesController.objectives_list)

module.exports = router
```

Views – error.pug

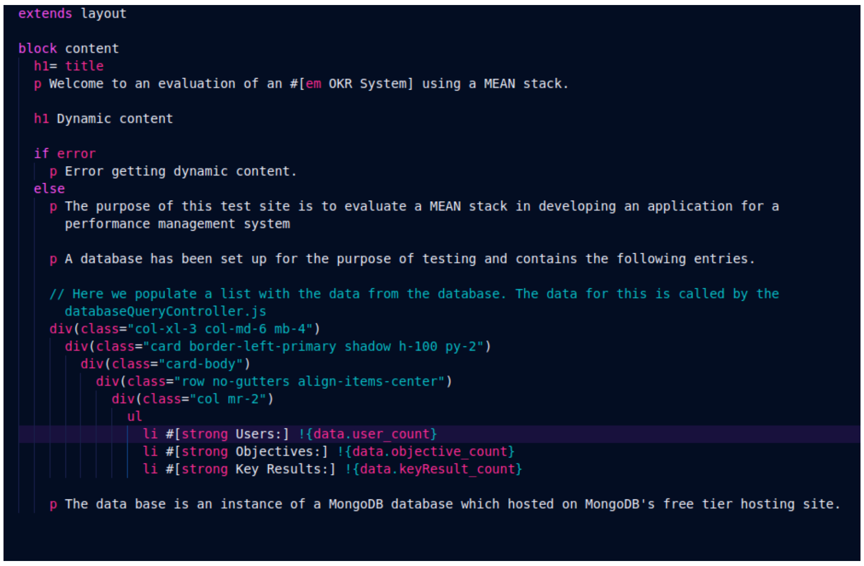
Here screen shots have been used to retain the indentation.



```
extends layout

block content
  h1= message
  h2= error.status
  pre #{error.stack}
```

Views – index.pug

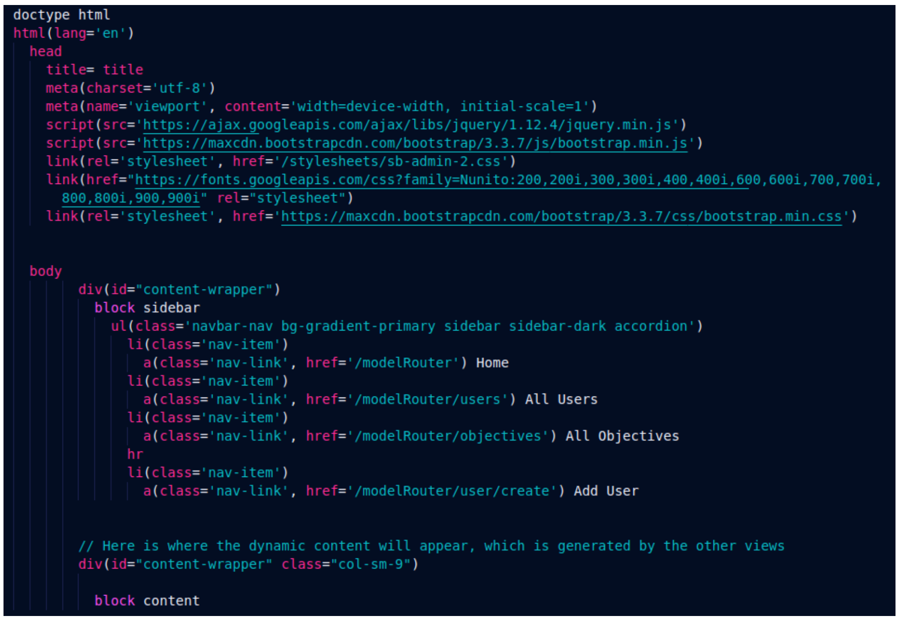
Here screen shots have been used to retain the indentation.



```
extends layout

block content
  h1= title
  p Welcome to an evaluation of an #[em OKR System] using a MEAN stack.

  h1 Dynamic content

  if error
    p Error getting dynamic content.
  else
    p The purpose of this test site is to evaluate a MEAN stack in developing an application for a
      performance management system

    p A database has been set up for the purpose of testing and contains the following entries.

    // Here we populate a list with the data from the database. The data for this is called by the
      databaseQueryController.js
    div(class="col-xl-3 col-md-6 mb-4")
      div(class="card border-left-primary shadow h-100 py-2")
        div(class="card-body")
          div(class="row no-gutters align-items-center")
            div(class="col mr-2")
              ul
                li #[strong Users:] !{data.user_count}
                li #[strong Objectives:] !{data.objective_count}
                li #[strong Key Results:] !{data.keyResult_count}

    p The data base is an instance of a MongoDB database which hosted on MongoDB's free tier hosting site.
```

Views – layout.pug

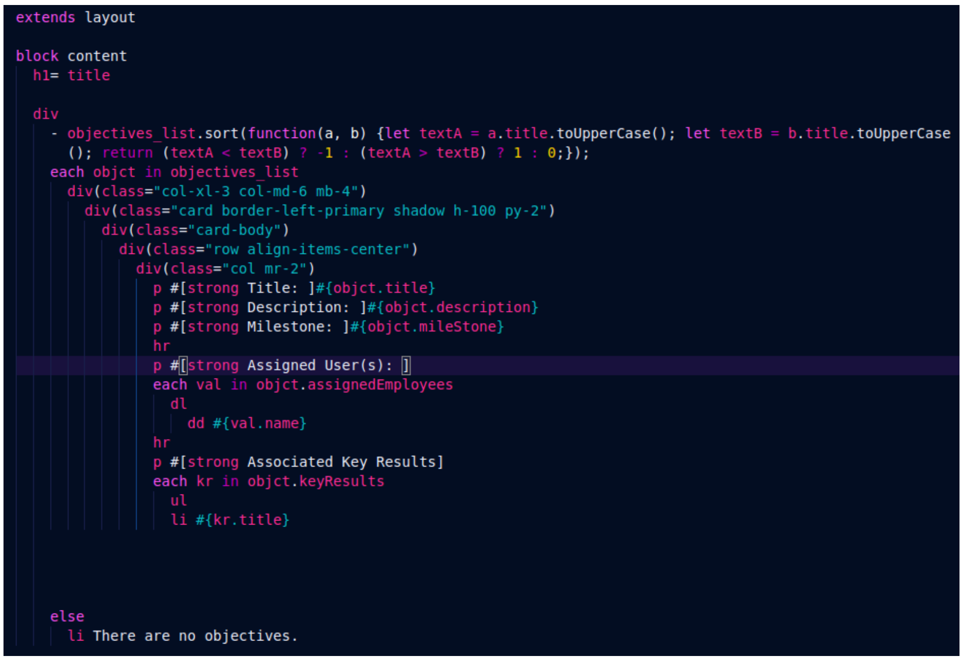
Here screen shots have been used to retain the indentation.



```
doctype html
html(lang='en')
  head
    title= title
    meta(charset='utf-8')
    meta(name='viewport', content='width=device-width, initial-scale=1')
    script(src='https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js')
    script(src='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js')
    link(rel='stylesheet', href='/stylesheets/sb-admin-2.css')
    link(href="https://fonts.googleapis.com/css?family=Nunito:200,200i,300,300i,400,400i,600,600i,700,700i,
      800,800i,900,900i" rel="stylesheet")
    link(rel='stylesheet', href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css')


  body
        div(id="content-wrapper")
          block sidebar
            ul(class='navbar-nav bg-gradient-primary sidebar sidebar-dark accordion')
              li(class='nav-item')
                a(class='nav-link', href='/modelRouter') Home
              li(class='nav-item')
                a(class='nav-link', href='/modelRouter/users') All Users
              li(class='nav-item')
                a(class='nav-link', href='/modelRouter/objectives') All Objectives
              hr
              li(class='nav-item')
                a(class='nav-link', href='/modelRouter/user/create') Add User


        // Here is where the dynamic content will appear, which is generated by the other views
        div(id="content-wrapper" class="col-sm-9")

          block content
```

Views – objectives_list.pug

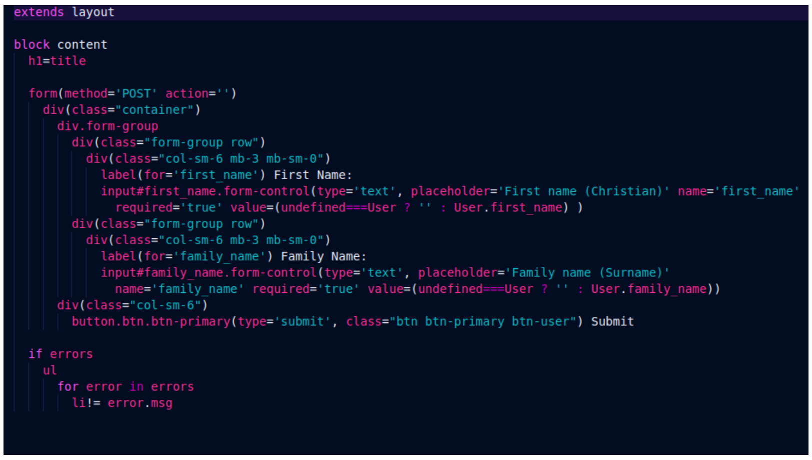
Here screen shots have been used to retain the indentation.



```
extends layout

block content
  h1= title

  div
    - objectives_list.sort(function(a, b) {let textA = a.title.toUpperCase(); let textB = b.title.toUpperCase
      (); return (textA < textB) ? -1 : (textA > textB) ? 1 : 0;});
    each objct in objectives_list
      div(class="col-xl-3 col-md-6 mb-4")
        div(class="card border-left-primary shadow h-100 py-2")
          div(class="card-body")
            div(class="row align-items-center")
              div(class="col mr-2")
                p #[strong Title: ]#{objct.title}
                p #[strong Description: ]#{objct.description}
                p #[strong Milestone: ]#{objct.mileStone}
                hr
                p #[strong Assigned User(s): ]
                each val in objct.assignedEmployees
                  dl
                    dd #{val.name}
                hr
                p #[strong Associated Key Results]
                each kr in objct.keyResults
                  ul
                  li #{kr.title}




    else
      li There are no objectives.
```

Views – user_form.pug

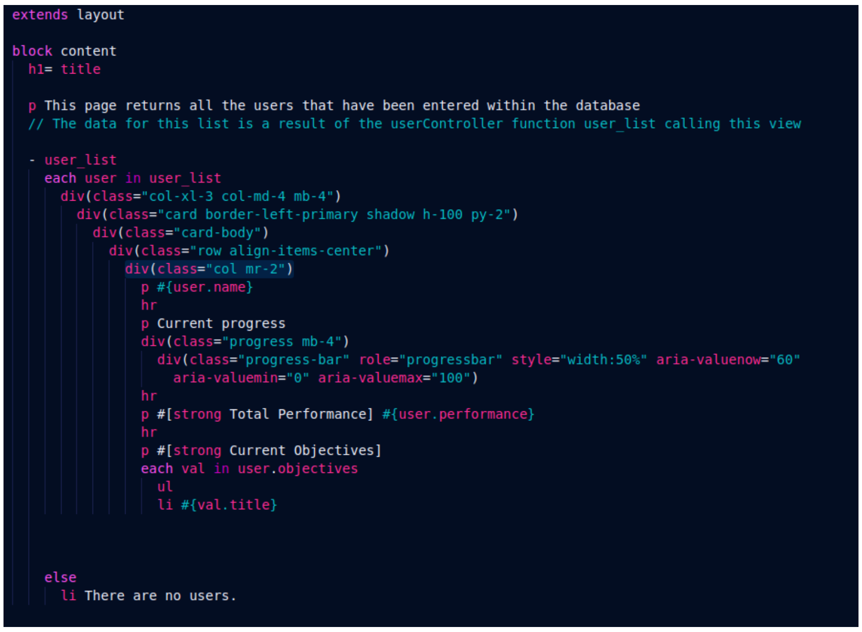
Here screen shots have been used to retain the indentation.



```
extends layout

block content
  h1=title

  form(method='POST' action='')
    div(class="container")
      div.form-group
        div(class="form-group row")
          div(class="col-sm-6 mb-3 mb-sm-0")
            label(for='first_name') First Name:
            input#first_name.form-control(type='text', placeholder='First name (Christian)' name='first_name'
              required='true' value=(undefined===User ? '' : User.first_name) )
        div(class="form-group row")
          div(class="col-sm-6 mb-3 mb-sm-0")
            label(for='family_name') Family Name:
            input#family_name.form-control(type='text', placeholder='Family name (Surname)'
              name='family_name' required='true' value=(undefined===User ? '' : User.family_name))
      div(class="col-sm-6")
        button.btn.btn-primary(type='submit', class="btn btn-primary btn-user") Submit

  if errors
    ul
      for error in errors
        li!= error.msg
```

Views – user_list.pug

Here screen shots have been used to retain the indentation.



```
extends layout

block content
  h1= title

  p This page returns all the users that have been entered within the database
  // The data for this list is a result of the userController function user_list calling this view

  - user_list
    each user in user_list
      div(class="col-xl-3 col-md-4 mb-4")
        div(class="card border-left-primary shadow h-100 py-2")
          div(class="card-body")
            div(class="row align-items-center")
              div(class="col mr-2")
                p #{user.name}
                hr
                p Current progress
                div(class="progress mb-4")
                  div(class="progress-bar" role="progressbar" style="width:50%" aria-valuenow="60"
                    aria-valuemin="0" aria-valuemax="100")
                hr
                p #[strong Total Performance] #{user.performance}
                hr
                p #[strong Current Objectives]
                each val in user.objectives
                  ul
                  li #{val.title}



    else
      li There are no users.
```

Annexe C – The Gantt Plan

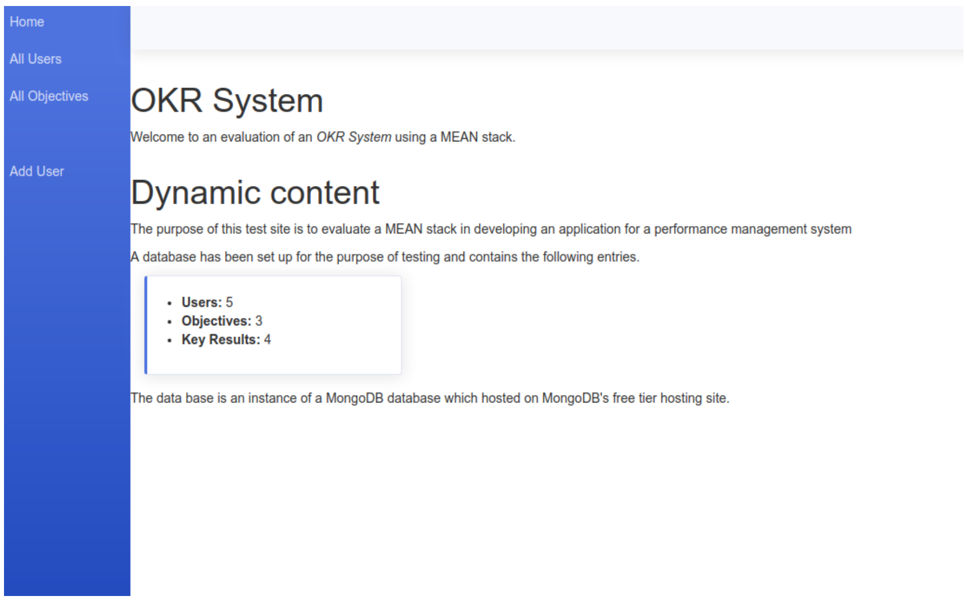
Here is the Gantt plan in tabular format.

WBS	Name	Start	Finish	Work
1	Domain Modelling - Aim	Feb 22	Mar 27	23d 6h
1.1	Planning - Phase	Feb 22	Feb 26	4d 3h
1.1.1	Choose Project Title - Task	Feb 22	Feb 23	1d
1.1.2	Populate Schedule - Task	Feb 22	Feb 24	1d 4h
1.1.2.1	ID Phases	Feb 22	Feb 22	4h
1.1.2.2	ID Tasks	Feb 23	Feb 23	4h
1.1.2.3	ID Mile Stones	Feb 24	Feb 24	4h
1.1.3	ID Life cycle - Task	Feb 22	Feb 22	5h
1.1.3.1	Compare and contrast Alts	Feb 22	Feb 22	5h
1.1.4	Identify resources - Task	Feb 22	Feb 22	5h
1.1.5	ID Risks - Task	Feb 26	Feb 26	5h
1.2	Research - Phase	Feb 22	Mar 11	11d 7h
1.2.1	literature search - Task	Feb 22	Feb 23	1d 2h
1.2.2	analyse literature - Task	Feb 24	Feb 24	5h
1.2.3	compile list of literature for possible use in project - Task	Feb 24	Feb 25	5h
1.2.4	write down relevant points from list - Task	Feb 25	Mar 7	6d 2h
1.2.5	complete review of literature - Task	Mar 7	Mar 11	3d 1h
1.3	Concept Development - Phase	Mar 11	Mar 20	5d 5h
1.3.1	Problem Statement - Task	Mar 11	Mar 14	1d 7h
1.3.2	Business Processes - Task	Mar 14	Mar 17	1d 7h
1.3.3	Domain Structural Model -Task	Mar 17	Mar 20	1d 7h
1.4	Evaluation - Phase	Mar 20	Mar 23	1d 7h
1.4.1	Review project to date - Task	Mar 20	Mar 21	5h
1.4.2	Adjust plan as required - Task	Mar 21	Mar 23	1d 2h
1.5	Milestone 1	Mar 27	Mar 27	
2	Requirements - Aim	Mar 27	May 12	30d 5h
2.1	Planning - Phase	Mar 27	Mar 27	5h
2.1.1	Readjust the schedule - Task	Mar 27	Mar 27	5h
2.2	Research - Phase	Mar 27	Apr 10	9d 3h
2.2.1	Literature search - Task	Mar 27	Mar 28	1d 2h
2.2.2	Analyse literature - Task	Mar 29	Mar 29	5h
2.2.3	Compile list of literature - Task	Mar 30	Mar 30	5h
2.2.4	Write down relevant points - Task	Mar 31	Apr 9	6d 2h
2.2.5	Complete the review - Task	Apr 9	Apr 10	5h
2.3	Concept Development - Phase	Apr 10	May 8	18d 6h
2.3.1	Use case Model - Task	Apr 10	Apr 19	6d 2h
2.3.2	Elaborated Use Cases - Task	Apr 19	Apr 28	6d 2h
2.3.3	Requirements F/NF - Task	Apr 29	May 8	6d 2h
2.4	Evaluation - Phase	May 8	May 11	1d 7h
2.4.1	Review project to date - Task	May 8	May 9	5h
2.4.2	Adjust plan as required - Task	May 9	May 11	1d 2h
2.5	Milestone 2	May 12	May 12	
3	Analysis - Aim	May 12	Jul 1	31d 2h
3.1	Planning - Phase	May 12	May 12	5h
3.1.1	Readjust the schedule - Task	May 12	May 12	5h
3.2	Research - Phase	May 12	May 26	9d 3h
3.2.1	Literature search - Task	May 12	May 13	1d 2h
3.2.2	Analyse literature - Task	May 13	May 14	5h
3.2.3	Compile list of literature - Task	May 14	May 15	5h
3.2.4	Write down relevant points - Task	May 15	May 25	6d 2h
3.2.5	Complete the review - Task	May 25	May 26	5h
3.3	Concept Development - Phase	May 26	Jun 23	19d 3h
3.3.1	System operations - Task	May 26	Jun 9	9d 3h
3.3.2	Analysis structural model - Task	Jun 9	Jun 23	9d 3h
3.3.3	Deploy a MEAN Stack	Jun 1	Jun 1	5h
3.4	Evaluation - Phase	Jun 23	Jun 25	1d 7h

3.4.1	Review project to date - Task	Jun 23	Jun 24	5h
3.4.2	Adjust plan as required - Task	Jun 23	Jun 25	1d 2h
3.5	Milestone 3	Jul 1	Jul 1	
4	Design - Aim	Jun 1	Aug 1	40d 5h
4.1	Planning - Phase	Jul 1	Jul 1	5h
4.1.1	Final adjustments - Task	Jul 1	Jul 1	5h
4.2	Design - Phase	Jun 1	Jul 29	37d 4h
4.2.1	Behaviour Model	Jul 1	Jul 15	9d 3h
4.2.2	Design structural model	Jul 15	Jul 29	9d 3h
4.2.3	Populate app logic for evaluation	Jun 1	Jun 29	18d 6h
4.3	Evaluation - Phase	Jul 29	Aug 1	2d 4h
4.3.1	Review end product - Task	Jul 29	Aug 1	1d 7h
4.3.2	Deploy MEAN stack	Jul 29	Jul 30	5h
4.4	Milestone 4	Aug 1	Aug 1	
5	Report writing consolidation - Phase	Aug 5	Sep 2	18d 6h
6	Submit TMA01	Feb 22	Mar 9	10d
7	Submit TMA 02	Apr 2	Apr 23	14d
8	Submit TMA 03	Jun 25	Jul 6	7d 4h
9	Submit EMA	Sep 2	Sep 16	9d 5h

Annexe D – Prototype Screen Shots

The live application's landing page is displayed within image 1. This page queries the database and returns the number of entries within it.



OKR System

Welcome to an evaluation of an *OKR System* using a MEAN stack.

Dynamic content

The purpose of this test site is to evaluate a MEAN stack in developing an application for a performance management system

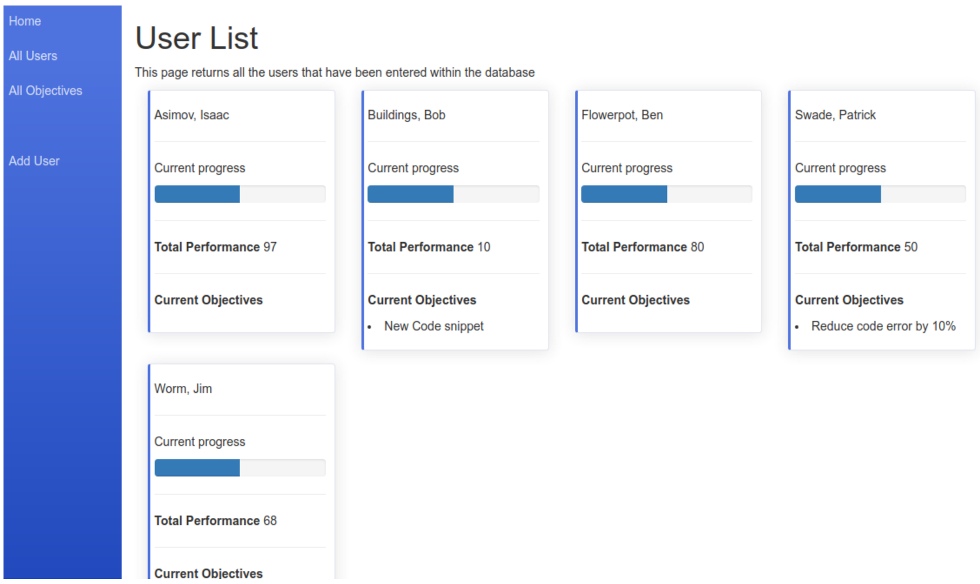
A database has been set up for the purpose of testing and contains the following entries.

- **Users:** 5
- **Objectives:** 3
- **Key Results:** 4

The data base is an instance of a MongoDB database which hosted on MongoDB's free tier hosting site.

Image 1

The all users page returns the details of all users and their current objectives. The total performance and progress for this prototype has been hard-wired into the User database. Future iterations will have this calculated by logic within the controller as defined by the Volere template (Annexe A)



User List

This page returns all the users that have been entered within the database

Asimov, Isaac
Current progress
Total Performance 97
Current Objectives

Buildings, Bob
Current progress
Total Performance 10
Current Objectives
- New Code snippet

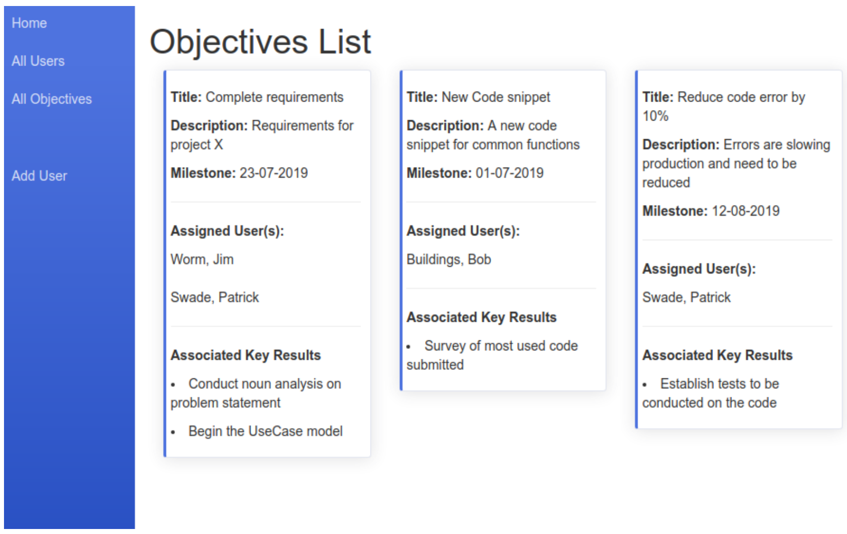
Flowerpot, Ben
Current progress
Total Performance 80
Current Objectives

Swade, Patrick
Current progress
Total Performance 50
Current Objectives
- Reduce code error by 10%

Worm, Jim
Current progress
Total Performance 68
Current Objectives

Image 2

Image 3 illustrates the all objectives view. As alluded to within the report this function was chosen as it is a prime example of how Express conducts inter-module communication.



Objectives List

Title: Complete requirements
Description: Requirements for project X
Milestone: 23-07-2019

Assigned User(s):
Worm, Jim
Swade, Patrick

Associated Key Results
- Conduct noun analysis on problem statement
- Begin the UseCase model

Title: New Code snippet
Description: A new code snippet for common functions
Milestone: 01-07-2019

Assigned User(s):
Buildings, Bob

Associated Key Results
- Survey of most used code submitted

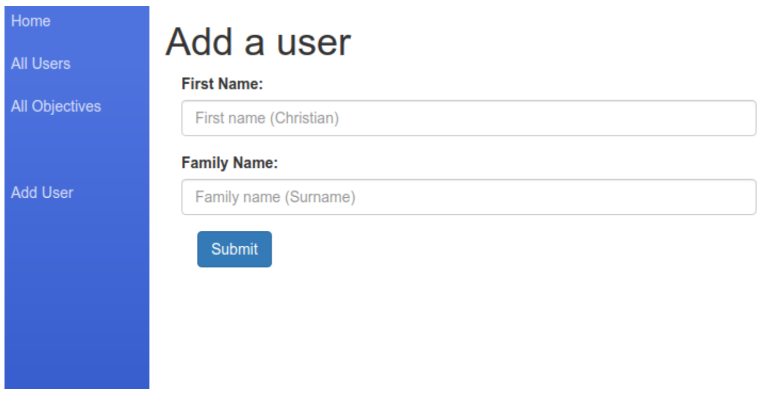
Title: Reduce code error by 10%
Description: Errors are slowing production and need to be reduced
Milestone: 12-08-2019

Assigned User(s):
Swade, Patrick

Associated Key Results
- Establish tests to be conducted on the code

Image 3

Image 4 illustrates the ass user function. This function demonstrates how data can be added to the database for subsequent queries.



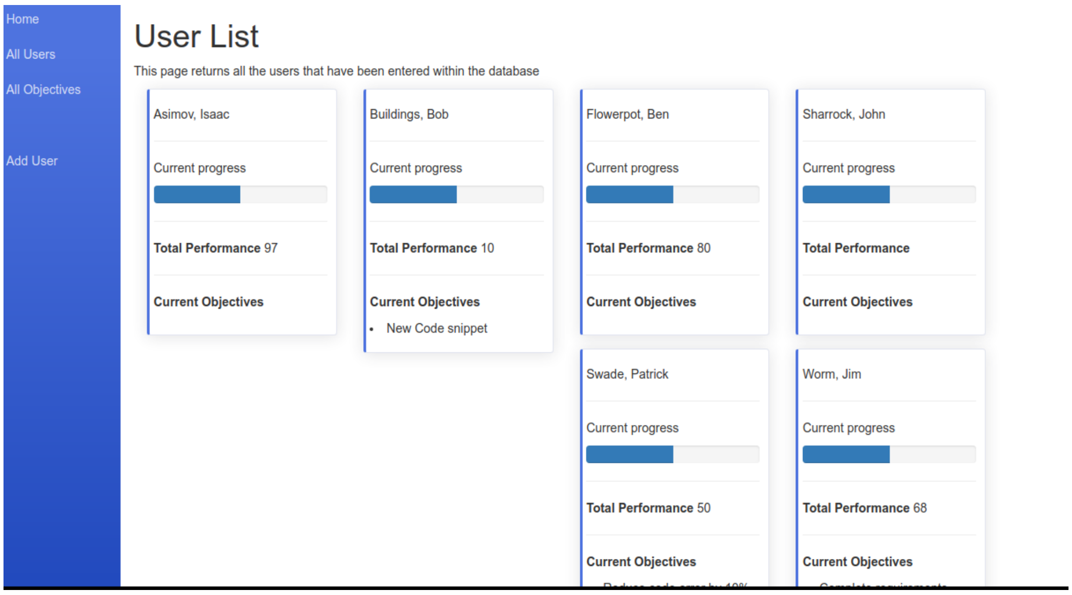
Add a user

First Name: First name (Christian)

Family Name: Family name (Surname)

Image 4

Image 5 illustrates the user list view after another person has been added to the database.



Home

All Users

All Objectives

Add User

User List

This page returns all the users that have been entered within the database

Asimov, Isaac

Current progress

Total Performance 97

Current Objectives

Buildings, Bob

Current progress

Total Performance 10

Current Objectives

- New Code snippet

Flowerpot, Ben

Current progress

Total Performance 80

Current Objectives

Sharrock, John

Current progress

Total Performance

Current Objectives

Swade, Patrick

Current progress

Total Performance 50

Current Objectives

- Reduce code error by 10%

Worm, Jim

Current progress

Total Performance 68

Current Objectives

- Complete requirements

Image 5