
1
Teamber

Information Technology and Informatics
Fredrik Bajers Vej 5,
9100 Aalborg

AALBORG UNIVERSITY
STUDENT REPORT

Title:

Teamber

Project:

Development of a Database System for a Specific Application

Project period:

Spring semester 2020

Project group:

BaIT / INF bi406f20

Group members:

Daniel Berg Fabricius Laursen
Simon Eliasen
Thomas Kristian Margon Vinther
Jakob Rønnest Sørensen
Line Buch Christensen

Supervisor:

Van Ho Long

Pages: 90 pages
Finished: 28.05.2020

Abstract:

The following report describes the process of creating an IT system in order to optimize the team creation process, based on user-submitted data. An iterative and object-oriented approach was used to accomplish this. To gain the proper knowledge needed to create this system, research was conducted on the matter of team creation and personality types, and a system definition was created, to gain a proper understanding of the problem at hand. By using the object-oriented approach to identify classes, events and behaviors etc. the development process was aided. The system was developed primarily using C#, ASP.NET and Razor Pages, which allowed the application to be accessible through a web-browser. Visual design methodology was used to create a basic overview of the website's design. The system was tested throughout the development process, assuring that as many requirements as possible, would be met. In conclusion the system fulfills most of the requirements, but does require further development and testing in order to be deployed in an organisation.

Contents

Introduction 1

In recent years there has been a rising trend regarding self-improvement and optimisation of one's life, this is utilised by the individual but its application furthermore stretches to businesses as well (Sinclair, 2019). This is mainly drawn from the rising availability of data to track, manage and predict from and utilising it with techniques such as data science and machine learning. If we focus on the workplace, we can see methods such as personality tests and workflow optimisation techniques, e.g. scrum and agile methodology, which further seek to streamline the individuals' performance within a group. This is only a fraction of recent methodologies and technologies that seek to support human resources in businesses, both for management and hiring. It is evident that the majority of the focus within HR-technology first and foremost lies on the individuals' performance, only to later compare it to the groups performance (Saborio, 2017). This makes for beneficial results, but what would happen if we sought to take a group-first approach. This will be achieved by taking a more holistic approach to team creation in order to make more competent, happier and better performing teams. This leads us to the initial question:

How do you create optimal teams based on employee data?

Problem analysis 2

Based on the initial findings in our introduction, we wanted to research how information about employees could affect a team composition process positively. In our search we used keywords such as *team composition* and *team management*. From these results we found a general correlation between team composition and team performance. This will be expanded further upon below.

In the study *Importance of team roles composition to success of construction projects* they conclude that an appropriate team composition[1], is an essential attribute in order to select a suitable member for a team. Furthermore it is outlined that an appropriate team composition with appropriate team members makes for better team performance and feeling of success among team members (Oke et al., 2016, p. 150-151). In the study the team composition is based on team roles, the understanding of them, and how they are used, to improve the overall effectiveness of a team. They conclude that suitable roles for a concrete project is important. And that the team members personality type and characteristic, should match the role type to some degree, in order to secure a favourable team work (Oke et al., 2016, p. 142-150).

In the study *Team Role Stress: Relationships With Team Learning and Performance in Projects Teams* they focus on the term *Team Role Stress*, and it is consequences related to team performance and individual performance (Savelsbergh et al., 2012, p. 67). They define *Role Stress* as a strain resulting from ambiguity, conflict or overload in multiple task requirements or roles of employees (Savelsbergh et al., 2012, p. 68). If the team members personality type and competences does not match the role, a role conflict can occur, which results in stress for the individual that has experienced these discrepancies. If role stress occurs the study concludes that it will negatively affect the team performance as much as the individuals performance (Savelsbergh et al., 2012, p. 87-88).

The study *Group personality composition and work team effectiveness: Key factor in staffing the team-based organization?* uses other empirical studies which concluded a work teams effectiveness is related to the composition of personality types and characteristics of the individual members (Halfhill et al., 2003, p. 148). From these findings the study measures different team compositions based on the members personality types and the effectiveness of these compositions. The conclusion is a recommendation to *"Generate and maintain an individual-level database of employee profiles on work-related personality traits, individual performance measures, and other outcome variables."* (Halfhill et al., 2003, p. 163). This will make it possible and a valid way to use personality types and competences as selection for team members when teams are created.

[1] By appropriate team composition they mean the right team roles for a concrete project (Oke et al., 2016).

Based on the studies, a database with information about each employees competences and personality type would make it easier to compose an effective team. In a team composition it would be possible to match each individual with a team role that matches their competences and personality type, which will reduce stress on the individual and team as a whole. This will increase their performance and feeling of success as a team.
Besides investigating theories about team compositions, we also want to research if other similar IT-systems, uses both personality types and competences as matching criteria for team compositions. In our research we found that most systems used only personality types as a criteria, and did not include the individuals competences.

Because our research shows that both personality type and competences in the individual is important when composing a team, and that existing systems does not include both of these criteria, it would be a useful and relevant tool to create. More specifically we want to create a system for the manager, that stores employees' competences and personality types in a database, in order to make the team composition process more efficient, and at the same time create more effective and optimal teams.

System Definition 3

In this chapter the system definition will be established by use of the FACTOR criterion. Before this can be achieved more information needs to be gathered on personality types, and how we can use them practically in a system to support the process of composing teams. This will be examined and explained in the following sections.

3.1 Team composition

Before we can use personality types in our system, two obvious questions need to be answered: Which personality type theories exist? And how can we use theories on personality types and tests in practice, to support team composition?

In our research we found two personality type theories of particular interest. Respectively *The Five Factor model* and *Myers Briggs Personality indicator*. To better grasp and understand the limits of these personality type theories, we further researched empirical studies about these two test methods. What we found was that the two personality type theories to some extent were reliable and valid. As an example a study have criticised the *Myers Briggs Personality Indicator* for not corresponding sufficiently to its Jungian framework[1] (Robert R. McCrae, 1989, p.17). The same study also suggests combining the two personality type theories, and abandoning the Jungian framework, and reinterpreting the *Myers Briggs Personality indicator* with the *The Five Factor model* (Robert R. McCrae, 1989, p.37).

With a thorough study of *The Five Factor Model* and *Myers Briggs Personality Indicator* theory, the next step was to find a valid online personality test that can support the team composition process. After examining different online test sites, we chose the site `www.16personalities.com`. The site has a free online test which have been taken more than 200.000.000 times, and it is based on both the *Myers Briggs Personality Indicator* and *The Five Factor Model* (NERIS Analytics Limited, a). To better understand how they compose these theories into one test, we will briefly explain the theory behind the two tests below.

Myers-Briggs Type Indicator (MBTI)

Myers Briggs Personality Indicator is one of the most widely used psychological test and it builds on the psychologist Carl Jung's theories about how people perceive and process information (Fairfield, 2012). The MBTI consist of four different dimensions, which describes how people tend to behave (Fairfield, 2012).

[1] A framework for identifying physiological types, created by the psychologist Carl Jung (Robert R. McCrae, 1989, p.17).

The first dimension is extroversion/introversion, which describes whether or not people are energised by social interactions. Again it is a spectrum meaning that if a person is extroverted, it does not exclude that the person has introverted tendencies. However the most adequate tendency is marked with a letter, (I) for introversion, and (E) for extroversion. The second dimension is sensing (S) and intuition (N). The third dimension is thinking (T) and feeling (F). The fourth dimension is judging (J) and Perceiving (P) (Fairfield, 2012).

The result of the MBTI test is summarised with the combination of letters from each dimension such as INTJ. The combination of the four letters describes one personality type. In total it gives 16 different personality types. Each personality type describes how a person prefers or tend to behave in different situations (Fairfield, 2012).

The Five Factor Model

The Five Factor Model is not directly a personality type theory, but it describes important trait structures, and is widely used in organisations. The goal of the model is describing all important traits or characteristics of an individual in a structured way (Robert R. McCrae, 2008). This is done by grouping related traits into one factor or dimension. The factors are neuroticism (N), extraversion (E), agreeableness (A), conscientiousness (C) and openness (O) (Robert R. McCrae, 2008). These factors or dimensions are very similar to MBTI, but there are two key differences. Firstly it has one more dimension. Secondly each factor is presented by a scale (See figure 3.1 for an example). The final result would not be a composition of factors defining a personality type, but an overview of the individual personality traits illustrated by a scale for every factor.

16personalities

The 16personalities website combines the two methods, by firstly using MBTI to create an overall personality type such as INFJ. Then they have added an extra letter to accommodate five dimensions which is adopted from the *The Five Factor Model*. This gives a final personality type such as ISTJ-T. Furthermore they have adopted the scale from *The Five Factor Model* (NERIS Analytics Limited, a). As an example if a personality type contain the letter (E) for extrovert, it is shown on a scale from 0-100% how extroverted the person is (See figure 3.1). Contrary to the MBTI which distinguishes between either being extrovert or introvert.

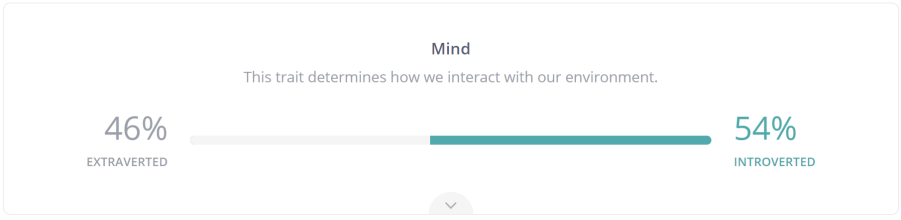

Figure 3.1. Example of a factor scale for extrovert/introvert (NERIS Analytics Limited, a)

Five aspects

The 16personalities are divided into five different personality aspects which are on a two sided continuum, see figure 3.1 (NERIS Analytics Limited, b).

- **Mind (I or E)** How we interact with our surroundings by either being introverted (I) or extroverted (E).
- **Energy (S or N)** How we see the world and process information by being observant (S) (practical, pragmatic and down-to-earth) or intuitive (N) (imaginative, open-minded and curious).
- **Nature (T or F)** How we make decisions and cope with emotions by thinking (T) (focus on objectivity and rationality over emotions) or feeling (F) (focus on social harmony and cooperation).
- **Tactics (J or P)** Our approach to work, planning and decision-making by judging (J) (decisive, thorough and organized) or prospecting (P) (improvising and flexible).
- **Identity (-A or -T)** How confident we are in our abilities and decisions by being assertive (-A) (refuse to worry too much and do not push too hard to achieving goals) or turbulent (-T) (wide range of emotions, success-driven, perfectionistic and eager to improve). This aspect will be added separately to the end of the given personality type (NERIS Analytics Limited, b).

Roles and Strategies

The five aspects will help define which type of personality the individual is. The type is divided into two layers: *Roles* and *Strategies*. The letters which the personality type is represented by comes from the five different aspects (NERIS Analytics Limited, b). An example could be *Analyst* with the letters N and T, which means they are intuitive and thinking types.

The difference between *Role* and *Strategy* will be described below. The *Roles* describes the goals, interests and preferred activities, and is divided into the following:

- **Analysts (_NT_)** This type embraces rationality and impartiality. They are good in STEM fields and in intellectual debates. They are independent, open-minded and strong-willed, and more interested in work than satisfying people. They are strategic thinkers but are having difficulties in social pursuits.
- **Diplomats (_NF_)** This type focus on empathy and cooperation. They are good in diplomacy and counselling, and good at cooperation and imagination, they do often have roles as harmonisers in workplaces. They are warm empathetic and influential people, but they have problems with cold rationality and difficult decisions.
- **Sentinels (_S_J)** This type is cooperative and practical. They create order, security and stability wherever they go. They are often hard working, traditional and good in logistical or administrative fields. They stick to their plans and do not shy away from difficult tasks, but can be inflexible and have a hard time accepting different points of view.
- **Explorers (_S_P)** The type is spontaneous and they have the ability to connect with their surroundings. They are practical and good in situations where there is need for quick reaction and thinking. They are good with tools and techniques (from physical tools to convincing people). They are irreplaceable in crises, crafts and sales, and can be pushed towards risky endeavors (NERIS Analytics Limited, b).

The *Strategy* helps define the preferred ways of doing things and achieving goals (NERIS Analytics Limited, b).

- **Confident Individualism (I___-A)** They prefer doing things alone and rely on their own skills. They know what they are good at and believes that personal responsibility and

trust in oneself are important values. They do not pay much attention to other peoples opinions.

- **People Mastery (E___-A)** They seek social contacts and have good communication skills. They are feeling at ease in social events or situations where they need to rely on or direct other people. Confident in their abilities and good at expressing their opinions and they wants to play an active roles in society and knows what affects other people, but are not concerned about what people think about them.
- **Constant Improvement (I___-T)** They are quiet and individualistic people, and tend to be perfectionists and success-driven. They spend a lot of time and effort to the result of their work are the best it can be. They are dedicated to their craft, but tend to worry too much about their performance.
- **Social Engagement (E___-T)** They are sociable, energetic and success-driven type, and tend to be restless, perfectionist individuals, experiencing both very positive and very negative emotions. They are curious and willing to work hard, which makes them high-achieving even and quite sensitive people. They find other people's opinions important, value social status and wants to succeed in everything they do (NERIS Analytics Limited, b).

An individual will be matched with both a role and strategy, which will define their personality type. There are 16 combinations of these. For example if a person gets INTP-T they would be a Analyst, which is a combination of the role *Analysts*, the strategy *Constant Improvement* and the aspect *Identity: Turbulent*.

Practical implementation

In this case we will use the personality type as a factor when composing a team. Each individual test result will be presented to the manager, so the said person can compose a team with the knowledge of the individuals competencies and personality type. This will make it possible to compose a more suitable and better team for the project. By knowing the personality type of the individuals it gives the manager a possibility to better communicate with the team and to handle and prevent problems within it.

3.2 The FACTOR criterion

The FACTOR criterion which describes the system definition will be described below based on the previous findings.

3.2.1 Theory

In the early stages of development, it is important to look at the core values and criteria of the system. Some challenges should be addressed at the early stages to prevent big problems in the later stages. Creating a system definition is a way to define these values and criteria clearly and concisely. Furthermore the system definition ensures that the stakeholders and developers agree on the overall system characteristics (Lars Mathiassen, 2000, p. 23). Therefore it is important that the system definition is written in a natural language that the stakeholders understand. One such way to create a system definition, is the FACTOR criterion.

The FACTOR criterion is a method for writing a system definition, using six elements: *functionality, application domain, conditions, technology, objects, responsibility.* The first element, *functionality* includes the core functionality of the system in regards to the application domain tasks. The second element is the *application domain*, that is the part of the organisation that administrates the problem domain. The third element is *conditions*, that are the contexts and environments that the system will be developed and used in. The fourth element is *technology. Technology* is the different kinds of technologies that are used during the system development and use. The fifth element is *objects*, which is the essential objects in the problem domain. The final element of the FACTOR analysis is *responsibility. Responsibility* is the clarification of what the system is responsible for and what the users are responsible for (Lars Mathiassen, 2000, p. 24).

3.2.2 Analysis

Functionality

The core functionality of the system is to suggest team members in order to optimise the team compositions. This will be based on the competences of the employees, and how well they match the competences for the team selected by the manager. Based on the match between them a employee competence score will be shown for the manager. A team competence synergy will also be shown, that is an average of the synergy scores of the team members. Furthermore the division of different personality types in the team, should be displayed on a graph or diagram. This is for helping the manager to understand the team dynamics, as mentioned in section 3.1.

The employees of the organisation should have a user profile which contains general information and competences. The employees competences is based on a questionnaire and their answers. The questionnaire is given based on the field in which the individual works, such as database development, social marketing, programming etc. These fields asserts the specific competences related to the employee.

Management should be able to access a user-database, where the users as well as their competences and teams, that they are assigned to is visible, and open for editing as well as creation and deletion of teams. Besides that, the system should allow the manager to access information about employee competences, create and send a questionnaire. The questionnaire questions are created based on the above-mentioned employee field. Furthermore it should be possible for management to add a new user to a team, where a rating of possible candidates for the team is shown based on the earlier mentioned *competence score.*

Application domain

Managers will use the system to create, manage and delete employees. They should also be able to manage all employees basic information, e.g. name and job title. All of this information is submitted when the employee is added to the system by the manager.

Managers should also be able to create, manage and delete teams and questionnaires.

Employees should be able to see their basic information in their profile. This includes teams that they are a part of and taking new personality test. The frequency of which employees should re-submit their self-assessed competence-level in questionnaires is determined by the manager.

Conditions

The system will be developed under extraordinary circumstances due to the COVID-19 pandemic. Concretely it gives limitations or changes to have the project group collaborate, as physical meetings are not possible. This means that all group and supervisor meetings will be done on an online platform (e.g. Skype, Microsoft Teams and Discord).

The users of the system will have very different levels of IT experience, depending on their education and experience.

Technology

The system will primary be developed using C#, ASP.NET Core, MSSQL, GitHub, Git, CSS/HTML, Javascript and additional libraries and frameworks for the interface. It will be accessible via a web-browser.

The system will ideally be used by medium to large sized companies, and we therefore expect that they have access to the technology required to access and use the software.

Objects

Employee, manager, team and questionnaire.

Responsibility

Managers will use the system to get curated suggestions for composing new teams based on team criteria set by the managers. The system does not differentiate the suggested members based on if they are already in a team.

The system will be able to suggest new team members based on each employee's competence match with the set team criteria. It will also show an overall team competence synergy, so it is possible for the manager to see how well the chosen team members match the set criteria.

The system will not take account of factors not defined by the system like the human factor.

Problem-Domain Analysis 4

In the following chapter the problem domain will be modelled, and build upon the knowledge gained from the FACTOR criterion. The overall purpose of this chapter is to understand the part of the systems context that is administrated, controlled or monitored by the system (Lars Mathiassen, 2000, p. 47). An important question to answer is, what does the system need to keep track of?

This chapter will thoroughly examine objects, classes, structures and behaviour.

4.1 Classes

4.1.1 Theory

In order to model the problem-domain, the first activity is to define the classes. In this activity the main classes and events are established, to ensure that the important phenomena are covered in the context of the project (Lars Mathiassen, 2000, p. 51). The result of the class activity is an event table, which contains the main classes and related events (See figure 4.1).

The first step in the class activity is to classify the main objects and events, by analysing user's work and thereby creating and selecting relevant abstractions for the objects and events. To understand this process, objects and events are used (Lars Mathiassen, 2000, p. 53).

Object
An object is an abstraction of a phenomenon in the problem domain and is generally defined as: *"An entity with identity, state, and behavior"* (Lars Mathiassen, 2000, p. 53). An example of an object could be a costumer.

Event
Events are used to describe the important incidences that the objects are involved in, and are generally defined as: *"An instantaneous incident involving one or more objects"* (Lars Mathiassen, 2000, p. 53). An example of an event could be reserved.

Class
The aim of the problem-domain analysis is to identify all relevant objects and events. This is accomplished using classes for practical and economical reasons (Lars Mathiassen, 2000, p. 55). As an example you can have the object *costumer* in a bank. If all objects and events should be described in this setting, it would be an unmanageable task with, a long list containing every costumer in the bank. Therefore the term *class* is introduced, which is a collection of shared objects. It is defined as: *"A description of a collection of objects sharing structure, behavioral pattern, and attributes"* (Lars Mathiassen, 2000, p. 55). An example of a class is the same as for an object, besides the fact that the class describes all similar objects.

Find Classes and Events

An important step and process in the class activity is to evaluate the possible classes and events, and only keep the suitable ones. A good practice is to operate experimentally. Meaning that you first identify potentially relevant classes and events in the problem domain (Lars Mathiassen, 2000, p. 56- 57). A method for generating potential events and classes could be to conduct an interview. It is important to mention that finding classes and events is a parallel activity. You can identify events through the classes and vice versa (Lars Mathiassen, 2000, p. 57). Subsequently you evaluate systematically and then select the classes and events that are relevant to the system at hand. In this process various evaluation criteria for both events and classes can be used as a guideline, to determine what classes and events are necessary (Lars Mathiassen, 2000, p. 57). To illustrate the class activity and its sub activities see figure 4.1.

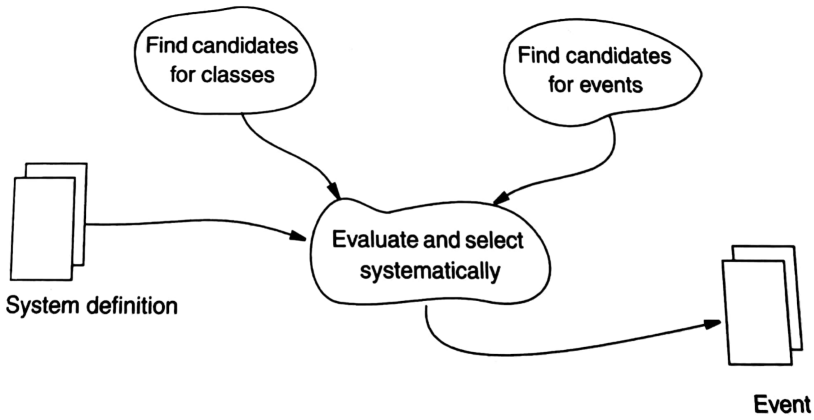

Figure 4.1. Illustration of subactivities in the class activity (Lars Mathiassen, 2000, p. 57)

When the above mentioned process is finished and relevant classes and events have been selected, the last step in the activity is to relate events to classes (Lars Mathiassen, 2000, p. 57). When doing this we ask the following general questions to determine if they are related (Lars Mathiassen, 2000, p. 66): *Which events is this class involved in? Which classes are involved in this event?.* Finally you end up with the result of the class activity: the event table (See figure 4.1).

4.1.2 Analysis

In our project the class activity was done as a mutual group activity based on the system definition in section 3.2.2. From the system definition we generated candidates for the classes and events. Besides that we investigated other sources for inspiration, such as systems in the same field. Through the candidates we meticulously discussed each candidate, to determinate its relevancy in our problem domain. This was conducted with our knowledge of the problem domain and the evaluation criteria for classes and events. It is also worth mentioning that the process was of an iterative nature, as many candidates were reevaluated. For instance, we came up with the following event candidate: *Team edited*. This event candidate encountered several events in the problem domain, for instance creating and deleting teams. The result of the discussion was that the event should be divided into more events, based on the evaluation

criteria that an event should be atomic, and not dividable (Lars Mathiassen, 2000, p. 65). The final event table is seen on figure 4.1.

Class description

To ensure that misunderstandings would not occur in the further development of the system, a description was added to each class. The description is mentioned below and describes the responsibility for each class.

Employee
The employee class stores and manages data about the employees of the business. With the main purpose of matching employees in teams, based on their attributes such as competences or personality type.

Manager
The manager class has the same attributes as the employee class, while also storing the teams and people they manage. It furthermore has access to administrate employees' data, the creation of teams and the management of these.

Team
The team class is a collection of employees. It has the responsibility for storing team members and furthermore the responsibility for calculating and storing the overall team competence synergy between the team members, which is described in 3.2.2.

Questionnaire
The questionnaire class generates a questionnaire based on the competences a manager has selected.

Event table

The result of all classes and events are summarised in the event table which is illustrated in table 4.1. In the event table the relation between our classes and events are marked with an X. For further clarification and understanding, the events that are harder to understand will be described below.

	Classes			
Events	*Employee*	*Manager*	*Teams*	*Questionnaire*
Employed	x	x		
Resigned	x	x		
Employee attribute edited	x	x		
Employee role changed to manager	x	x		
Team created		x	x	
Team deleted		x	x	
Team competence criteria edited		x	x	x
Added questionnaire to team		x	x	x
Removed questionnaire from team		x	x	x
Member added to team	x	x	x	
Member removed from team	x	x	x	
Questionnaire taken	x	x		x
Added competence to questionnaire		x		x
Removed competence from questionnaire		x		x
Questionnaire sent		x		x
Questionnaire created		x		x
Questionnaire deleted		x		x

Table 4.1. Event table

Employee attribute edited

Employee attributes in this instance include general information such as name and job title. This event involves the manager and employee class, since the manager has the authority to change some of the employees attributes, and the employee class is involved since this class' attributes have changed.

Questionnaire taken

When the employee has taken a questionnaire, the competence ratings in the questionnaire is automatically stored in the employee class. When the questionnaire is taken again then the new ratings will overwrite the old. This event is important to keep track of. If a manager creates a group with certain criteria, it is important that the system knows the updated employee competences so the result is reliable.

Added competence to questionnaire

When a manager wants to compose a questionnaire, competences are selected. This is because a questionnaire consists of competences such as PHP and CSS. When the competences are selected, the system automatically generates a standard question that contains the competence.

Added questionnaire to team

This event is initialised when a manager wants to create a team. To make it possible to change the competence criteria for a team, a manager needs to add a questionnaire. When the manager adds a questionnaire to a team, it is possible to set the criteria for the associated competences. As an example if the questionnaire consists of a competence called *C#*, the manager can set a priority from 1-5 (5: very important). This allows the manager to see the employees synergy match with the criteria and the team competence synergy, based on their answers of the questionnaire.

Team competence criteria edited
This event occurs when the manager changes the competence criteria for a specific team. It is connected to the *Added questionnaire to team* event. This event also involves the questionnaire class, as the questionnaire consists of the competences as explained earlier.

4.2 Structure

After having established the classes in the system, the structure will now be defined with the use of a class diagram.

4.2.1 Theory

The purpose of the class diagram is to describe structural relations between objects and classes in a problem domain (Lars Mathiassen, 2000, p. 71). The structural relations are different for the classes and objects which will be described below.

Class structures

The relations between the classes are abstract, static and general, and can consist of a generalisation. A generalisation is a general class that describes the properties that are common for a group of subclasses. Generalisations can be expressed linguistically using the formulation "is-a". The subclasses inherits all of the properties and behaviors that the super-class contains (Lars Mathiassen, 2000, p. 75). A class is abstract if it has no related objects. These classes are used for descriptive reasons and the class names are written in italic (Lars Mathiassen, 2000, p. 73).

Object structures

The relations between objects are - opposite to classes - specific and dynamic, and consists of either an aggregation or an association. An aggregation structure means that one object is a fundamental and defining part of the other (Lars Mathiassen, 2000, p. 77). Aggregations can be expressed linguistically using the formulation "has-a" (Lars Mathiassen, 2000, p. 78). Likewise the classes that are aggregated can be linguistically expressed as "part-of" or "owned-by" (see figure 4.2).

An association structure is much like the aggregation structure, but the difference is that the associated objects are not fundamental and defining for one another. Therefore the association structure is often used when an aggregation structure would imply a too strong relation (Lars Mathiassen, 2000, p. 79). The association structure can be linguistically expressed as "knows" or "associated with".

When creating the class diagram it is important to numerate the possible relations. If for example *object A* is an aggregation of zero or more *object B*'s, this is denoted by "0..*" (Lars Mathiassen, 2000, p. 78).

Creating class diagrams

When the class diagram is being created it is important to only model the necessary structural relations (Lars Mathiassen, 2000, p. 86). Likewise the use of generalisations and aggregations

should only be applied when they add anything relevant to the description. The names, concepts, and structural relations should reflect the users' understanding of the problem domain. But they do not necessarily have to be completely identical (Lars Mathiassen, 2000, p. 87). Though the best case scenario would be that the problem domain reflects a future user's understanding of the problem domain.

The graphical notation of the relation structures between classes is illustrated in figure 4.2.

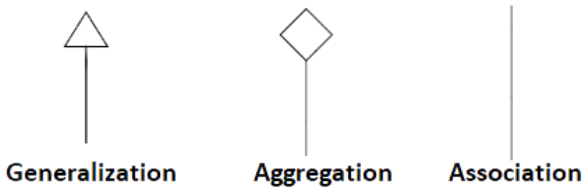

Figure 4.2. Graphical notations of structures

4.2.2 Class Diagram Analysis

In section 4.1.2 we described that our problem domain contains four classes: *Employee*, *Manager*, *Team*, and *Questionnaire*. The *employee* and *manager* classes could be described as roles for a new general class *user*. The validity of this choice is tested using the linguistic formulation for aggregations "has-a" and the linguistic formulation for generalisations, "is-a". So "a user has a role" and "a manager is a role". This confirms that these classes can be roles of a class called *user*. There are no objects related to *Role* so this is an abstract class, and the class name is therefore written in italic in the class diagram.

The primary job for the manager is to compose new teams. The relation between the *manager* class and the *team* class could therefore be described as an association. They have a meaningful relation but one object is not fundamental or defining of the other. This relation is linguistically expressed as "a manager is associated with a team". One team can be managed by one or more managers, while one manager manages zero or more teams. This is respectively denoted as "1..*" and "0..*" in the class diagram.

The *Team* class could be described as an aggregation of *employee*-objects, because the selected employees make up a specific team. Without these employees there would not be a team. The validity of this choice could be tested by using the linguistic formulation for aggregations, "has-a" or "is-part-of". So "a team has an employee" and "an employee is part of a team". One employee can be part of zero or more teams, while one team requires two or more employees.

The employees fill out questionnaires in order for the managers to collect data about the employees competences and personality types. For this reason the *employee* class is associated with the *questionnaire* class. There are multiple types of questionnaires based on the employees fields of work. For this reason one employee is associated with one or more questionnaires, and one questionnaire is associated with many employees.

The managers create the questionnaires, which is why the *manager* class are related to the *questionnaire* class. The managers are not a defining property of the questionnaire objects and vice versa. For this reason the relation is an association structure, and not an aggregation structure. One manager can be associated with zero or more questionnaires, while one

questionnaire is associated with at least one manager.

The managers manage the employees which is why they are related. Like the *manager* and *questionnaire* relation, the managers are not a defining property for the employees, which is why the relation is an association structure. One manager can manage zero or more employees, and one employee can have zero or more managers.

This results in the class diagram shown in figure 4.3.

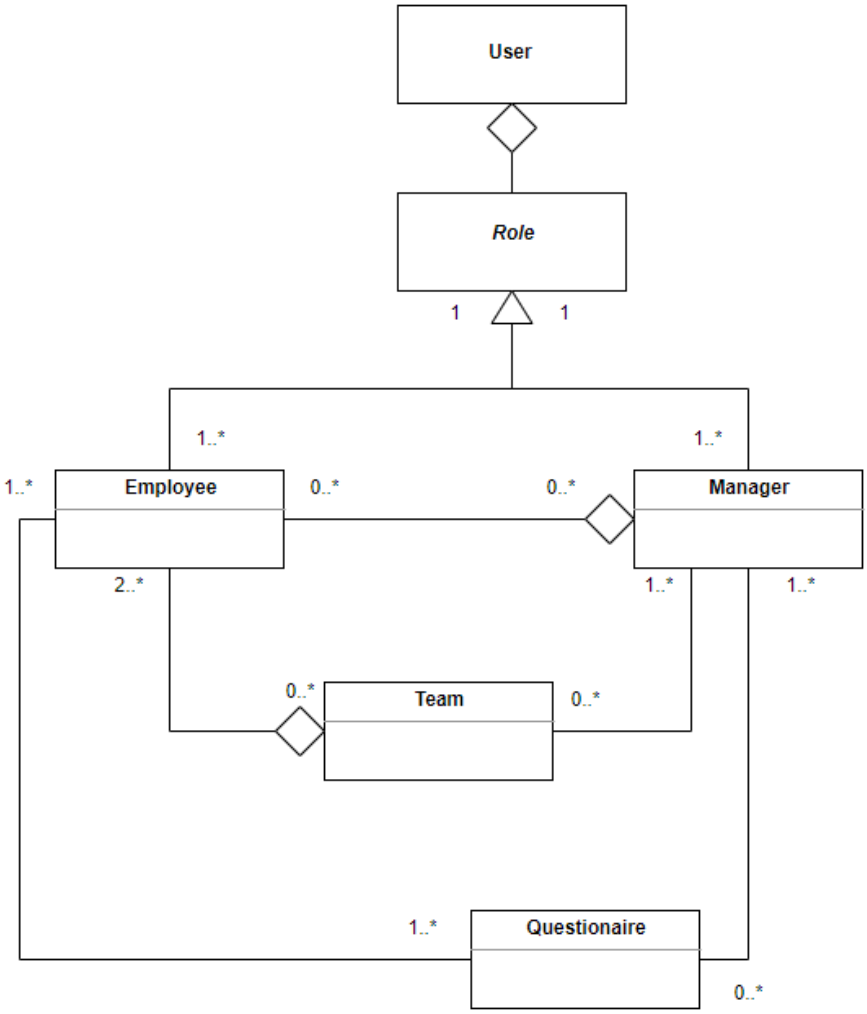

Figure 4.3. Class diagram for the problem domain

4.3 Behavior

4.3.1 Theory

The sole purpose of analyzing the behaviour of an application is *"to model the dynamics of the problem-domain"* (Lars Mathiassen, 2000, p. 91). When using the behaviour-activity the descriptions from the class-activity is utilised. This is further expanded upon with information about the behavioral patterns and attributes. As a result a statechart diagram can be created, as seen in figure 4.4, which showcases a class' behavioral pattern and attributes, which will be

described in depth below.

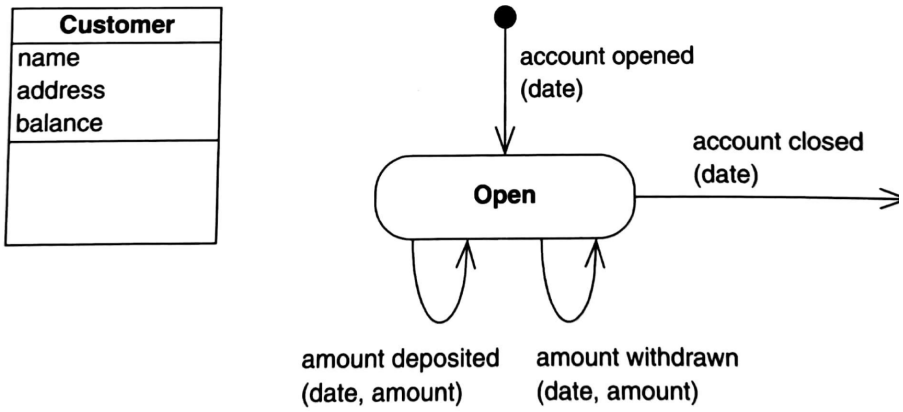

Figure 4.4. Example of a statechart diagram (Lars Mathiassen, 2000, p. 92)

In statechart diagrams the focus is on the behavior of objects, and their relative timing of execution. This is also called an event trace and is defined as *"a sequence of events involving a specific object"* (Lars Mathiassen, 2000, p. 92). An example of an event trace could be the life cycle of a bank account, like *opened account - money deposited - money withdrawn - account closed.*

In regards to behavior it does not focus on objects, but on the behavioral pattern of the objects in a class. Behavioral patterns can also be described as *"A description of possible event traces for all objects in a class"* (Lars Mathiassen, 2000, p. 92).

Behavioral patterns can be identified by studying all the possible outcomes of event traces, that can or will be performed by objects in the class. This gives you the class' behavioral pattern, that displays possible event.

Additionally common event traces in different objects help link behavioral patterns together to showcase how classes event traces interact with each other (Lars Mathiassen, 2000, p. 93).

Based on the behavioral patterns of classes it can be determined what attributes or data that needs to be transferred or translated by the event traces. An attribute is also defined as *"a descriptive property of a class or an event"* (Lars Mathiassen, 2000, p. 94). The attributes of a class can often be derived from a formerly developed class description. An example of attributes are shown in the left of figure 4.4.

Notation of behavioral patterns

To translate the possible events for behavioral patterns through time, similar notation to control structures can be used to describe the behavioral patterns in the problem domain. This can be done by using: *sequence*, *selection* and *iteration* which are defined as following (Lars Mathiassen, 2000, p. 95):

Sequence: Events in a set occur one by one

Selection: Exactly one out of a set of events occur

Iteration: an event occurs zero or more times

The notation can be utilized in writing as: *opened account + (money deposited | money withdrawn)* + account closed* which showcases the various notation techniques. This example showcases both selection, which is the "+" notation, selection marked by "|" and iteration "*". These control structures can likewise be translated graphically by using the notation seen in 4.5. This ultimately concludes to a fully developed statechart diagram as seen in figure 4.4.

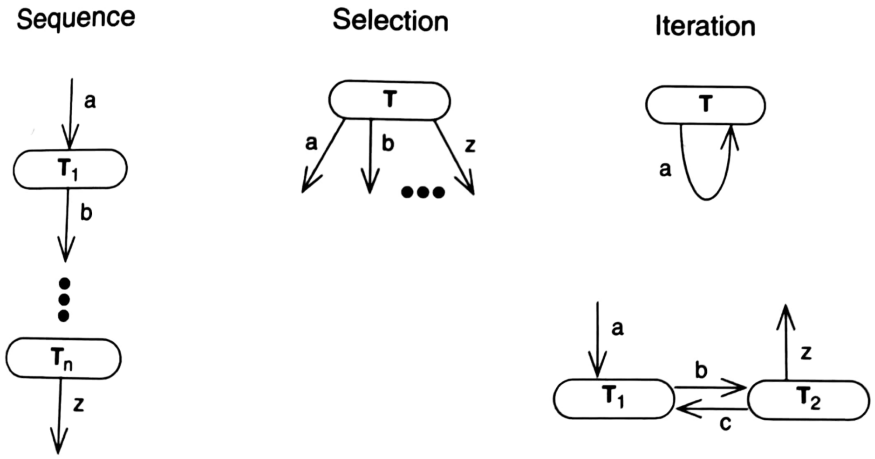

Figure 4.5. Statechart notation (Lars Mathiassen, 2000, p. 97)

4.3.2 Analysis

By using a statechart diagram it is possible to get a dynamic description of the problem domain, by looking at it over time based on the previously mentioned classes and events. In the following section the behavioral patterns of our classes will be presented, using statechart diagrams.

Employee class

The employee class is in charge of registering the data of the employees that later can be used for categorisation, identification, and matching employees in groups. These attributes are shown on the left of figure 4.6. The employee class is initialised once an employee is hired from where the account now is active. From an active class the employee can now utilise the following iterative behaviors.

- Employee added to a team.
- Employee removed from a team.
- Questionnaire taken.
- Added to questionnaire
- Removed from questionnaire

The life cycle of the object ultimately ends once the employee leaves the company.

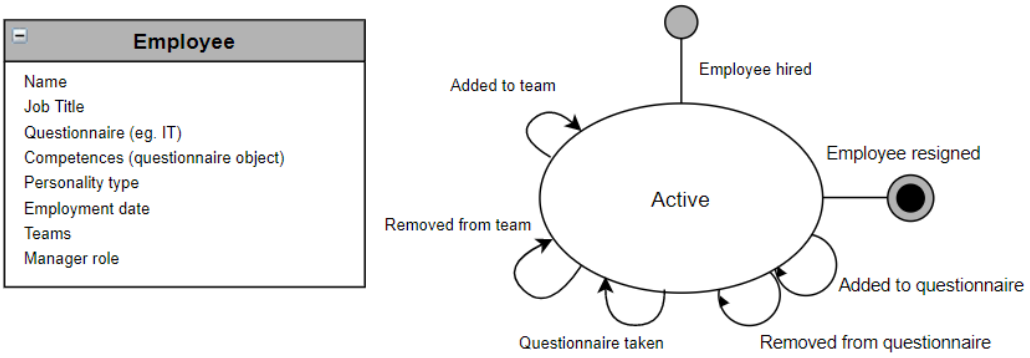

Figure 4.6. Statechart diagram of the employee class

Manager class

The manager class is one of the major classes as it orchestrates the majority of content creation on the platform. The class shares the same attributes and events as an employee but further has access to do a large number of updates in the system, such as editing employee or team information. The manager class is initiated, when a regular employee is assigned to be a manager. The events that the manager class behavior consists of, is shown in figure 4.7. These can be done iteratively by the manager. The manager object cycle ultimately ends once the managers position changes to being a regular employee or leaves the company.

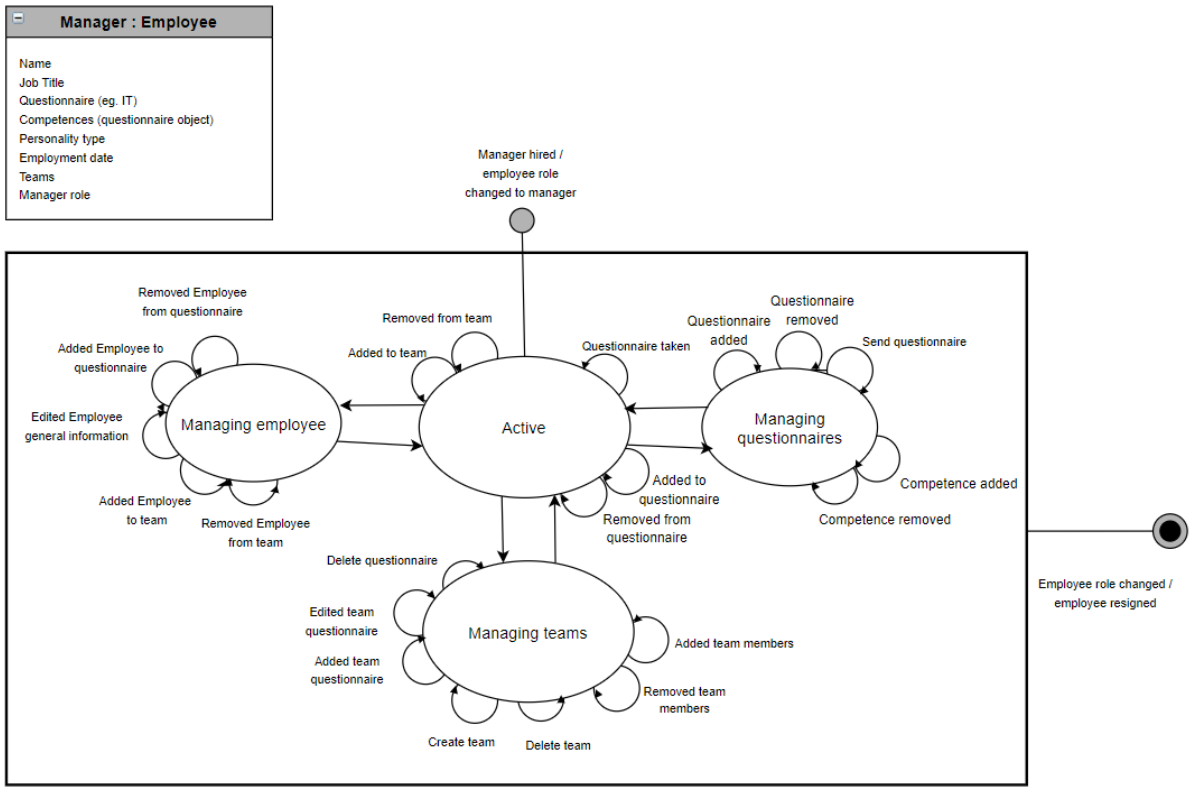

Figure 4.7. Statechart diagram of the manager class

Team class

The team class stores the employees included in a given team, the overall team competence synergy, as well as the criteria for employees to be considered an optimal fit for the team.

The team class is initialised by a manager class, from where the team class is in a state of waiting for members. Different questionnaires can here be added which makes it possible to set the criteria for the team. When the criteria have been set it is possible to add or remove employees to the team. When the team composition is done the team enter the state *Active team* where team members can be added or removed, as well as the criteria for the team can be changed. During the different stages of the team, the team can always be deleted or canceled, as shown in figure 4.8.

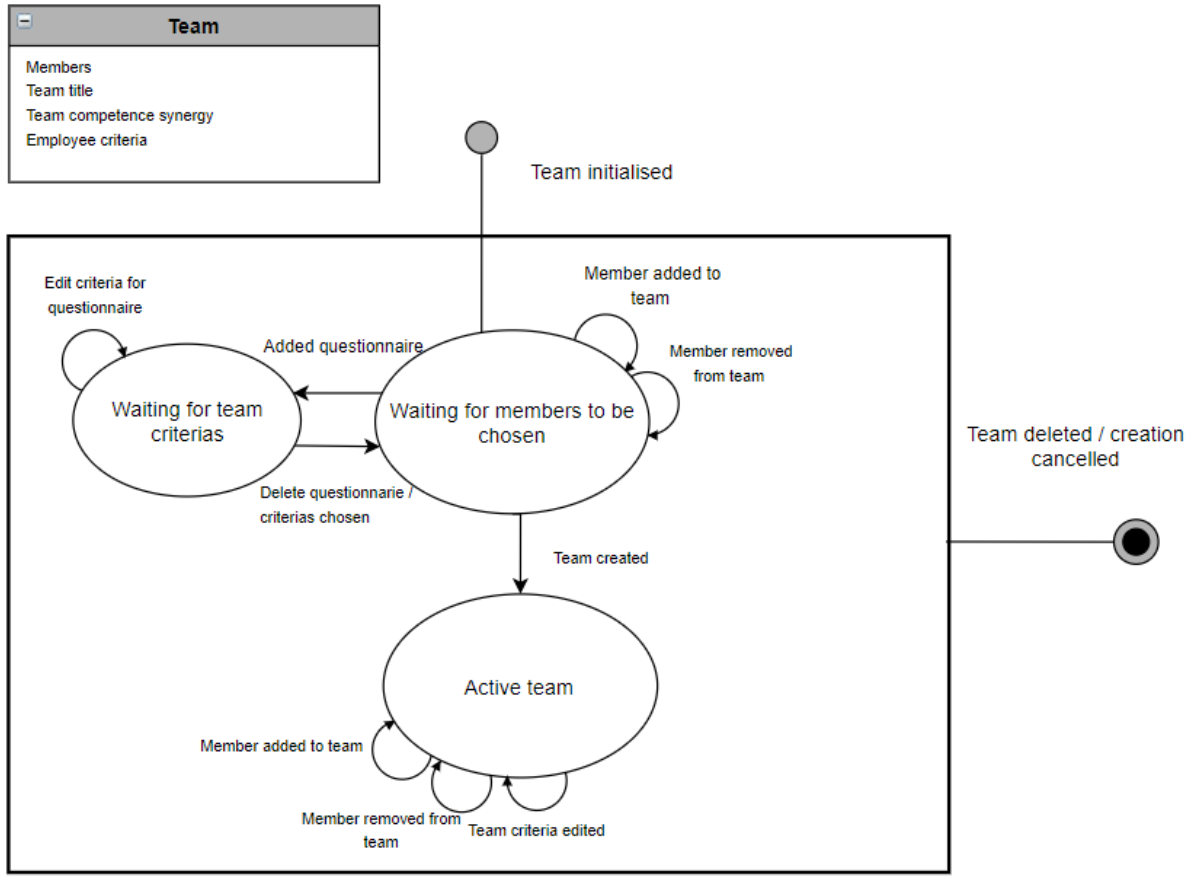

Figure 4.8. Statechart diagram of the team class

Questionnaire class

The questionnaire behavioral pattern is very dynamic, in the sense that its attributes are formed based on inputs from the manager, which generates the questionnaire. These attributes include:

- Questionnaire name
- Questionnaire cycle (How often should questionnaires be send to users)
- Competences

The questionnaire's life cycle is initialised upon the manager's request thereof. In the *Under creation* state competences can be added and removed. This results in the state of an active questionnaire, from which the questionnaires can be sent to the users. It also gives us the opportunity to go back later and change its attributes. At any point, the questionnaire creation can be cancelled or deleted which is defined as the determination point in figure 4.9.

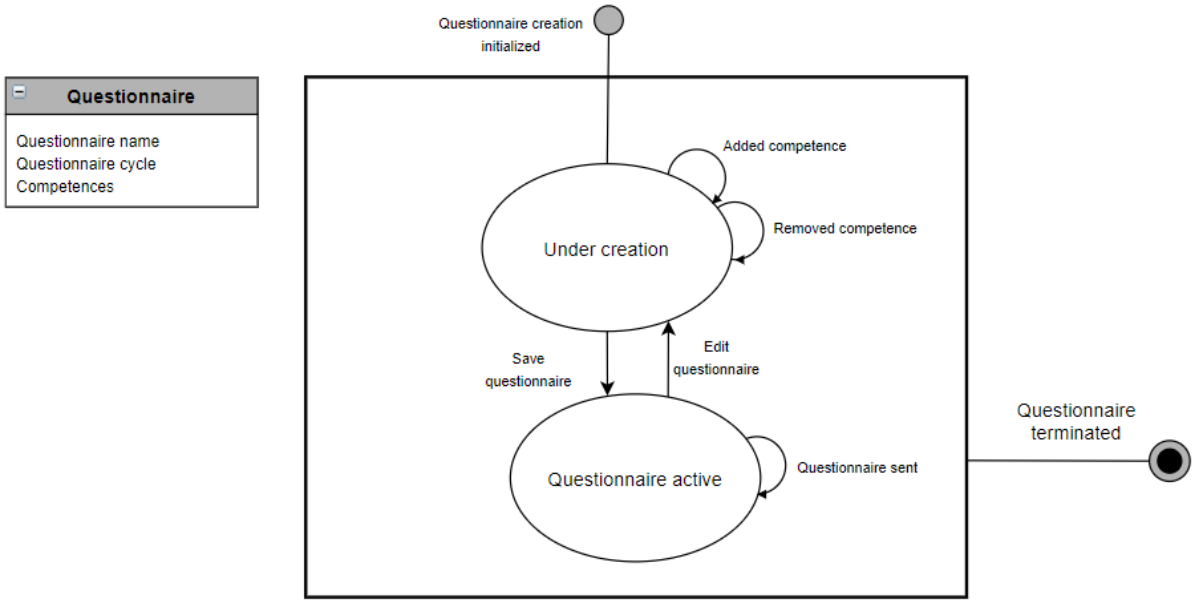

Figure 4.9. Statechart diagram of the questionnaire class

Application-Domain Analysis 5

In this chapter an application domain analysis be conducted to determine, how the system will be used by the user. With knowledge about the systems model from the problem-domain the requirements for the system can be defined. This will make it possible to understand how to best control our problem domain (Lars Mathiassen, 2000, p. 117).

The chapter will examine usage, functions and visual design of the system in the following sections.

5.1 Usage

5.1.1 Theory

The purpose of the usage activity is to understand and determine how the users interact with a system. This is done by examining the application domain. To maintain efficiency and value for the developers, relevant abstractions are needed to describe the usage. To achieve this the concepts *actors* and *use cases* are used (Lars Mathiassen, 2000, p. 123-124). The final result of the usage activity is a list of described actors and use cases. Furthermore an actor table is created, which shows the relation between these (See figure 5.3) (Lars Mathiassen, 2000, p. 123). The above mentioned concepts will be described below.

Actors

Actors are an abstraction of a systems users or other systems that interacts with the target system (Lars Mathiassen, 2000, p. 121). Users or systems with similar interaction patterns will be described as an actor. To identify actors you examine different work tasks in the application domain, and the people that are related to that task (Lars Mathiassen, 2000, p. 126). A way to do this is for example to study existing descriptions of the application domain, such as the system definition (Lars Mathiassen, 2000, p. 127). When the actors have been identified, an actor specification is used to describe them. The specification describes goals, characteristics and examples of the characteristics (Lars Mathiassen, 2000, p. 128) (See figure 5.1).

Use cases

Use cases are an abstraction of the actors' interaction with a system, and describes a limited use of different parts of the system (Lars Mathiassen, 2000, p. 122). It is defined as: *"A pattern for interaction between the system and the application domain"* (Lars Mathiassen, 2000, p. 122). Use cases can either be initiated by an actor or by the system itself. The goal is to determine the ways actors can interact with a system, in a few fitting use cases (Lars Mathiassen, 2000, p. 129). Identifying use cases can be a challenge. This can be accomplished by closely cooperating

with the future or current users of the system, and thereby gain insight in the application domain (Lars Mathiassen, 2000, p. 122). Another approach, as mentioned in the actor section, is to identify work tasks and actors from the system definition, and thereby identify use cases.

The next step after having identified the different use cases, is to describe the dynamic nature of them. This is either done by a textual description or a statechart diagram. A statechart diagram for a use case is very similar to the earlier described statecharts in the behavior theory 4.3.1. In this instance the different states shows how the user can interact with the system, and how the user can change the states (Lars Mathiassen, 2000, p. 129). The different ways the interaction can change state, are noted with a *sequence-*, *selection-* or an *iteration* notation, see figure 4.5. An example of a use case is shown in figure 5.2.

5.1.2 Analysis

Actors

We identified our actors by determining who the user of the system would be. By looking at our system definition and behavior analysis we found two key actors: *employees* and *managers*. Their usage in the system and characteristics are described to gain a better understanding of their interaction with the system, see table 5.1 and 5.2.

Employee	
Goal	An employee has access to a profile, where they should be able to take questionnaires with competences based on their field of work. Furthermore they can see their general information such as name, personality type etc.
Characteristics	The users have very diverse characteristics as this actor covers all of the employees in a company that are not managers.

Table 5.1. Actor *employee*

Manager	
Goal	A manager is an employee with manager rights in the system. Their main tasks are the composition of suitable teams and the management of these. They can create a questionnaire based on the employees field-of-work to gather data on their competences. They can register new employees and manage their general information.
Characteristics	The managers are primarily human resource employees and project managers. They can have different characteristics and IT-experience.
Examples	A human resource employee uses the system to manage many employees and teams. They normally use IT-systems to manage employees, so we presume they typically would be open-minded and curious using a system and its possibilities.

Table 5.2. Actor *manager*

Use cases

With the actors established, and our findings in the behavior analysis and system definition, we can now determine the most relevant use cases and how the employee and manager are involved in these.

To choose use cases we looked at the interaction between the actor and the system. By doing this we came up with the following use cases:

- Register employee
- Create questionnaire
- Take questionnaire
- Compose team

These use cases covers the most important parts of the system and are dependent on each other. It is necessary to register an employee, in order for the employee to take a questionnaire and be a part of a team. The employee can only take a questionnaire if the manager has assigned a questionnaire to the employee.

The actor table in figure 5.3 shows which actors are involved in each use case.

	Actors	
Use cases	*Employee*	*Manager*
Compose team		x
Register employee		x
Create questionnaire		x
Take questionnaire	x	

Table 5.3. Actor table

Each use case will be described with a statechart diagram to give a better understanding of how the actors interact with the system.

Login

In all the use cases the user have to login to the system. Although the login functionality is not a use case, we have chosen to describe it in a statechart in order to simplify the rest of the use cases.

The statechart describes how the login functionality works. We have two different user types, *employee* and *manager*, and the functions associated to these are different. If an employee logs in to the system the access given will only be for the said persons profile. If a manager logs in then a choice between logging in as a employee or manager will be given. If the user is logged in as a manager, then the team-, questionnaire-, and employee pages should be accessible.

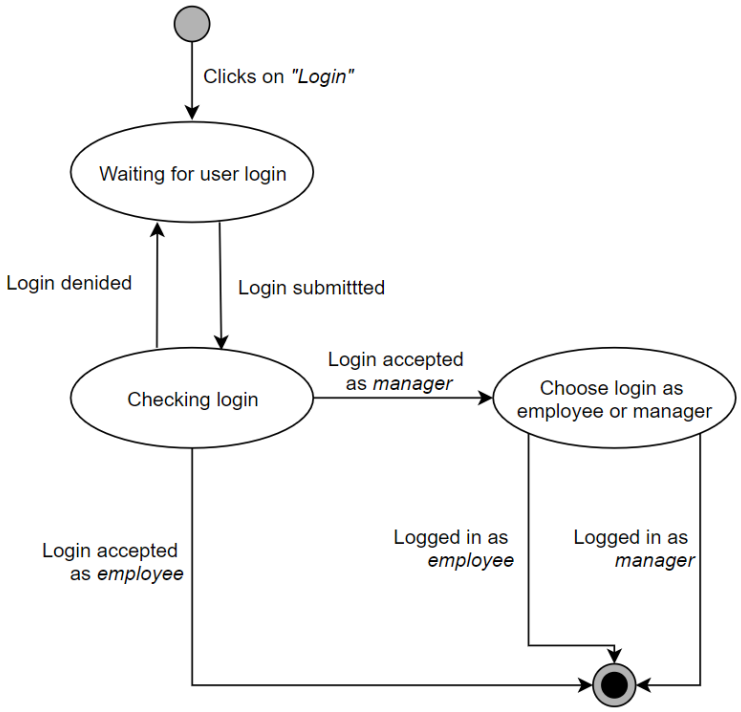

Figure 5.1. Statechart diagram for Login

Register employee

The statechart in figure 5.2 describes how the manager registers an employee in the system. In the state *Waiting for information to be filled out* the general information for an employee is filled. This includes information such as name, personality type and job title.

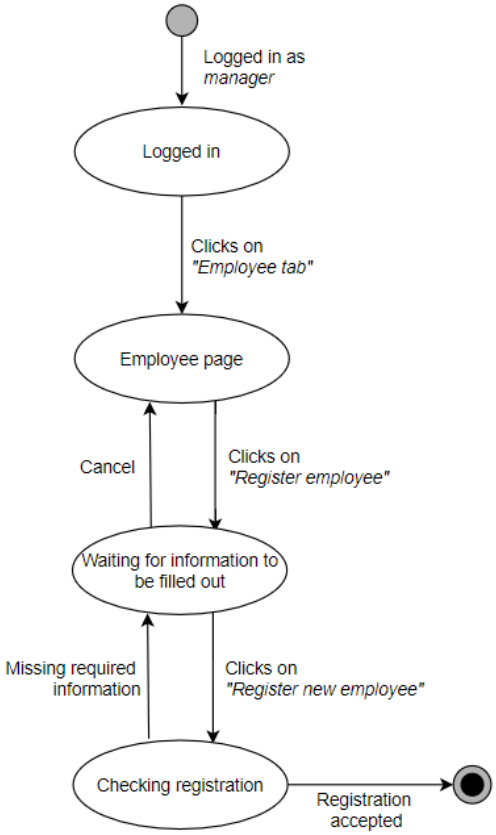

Figure 5.2. Statechart diagram for Register employee

Create questionnaire

This diagram shows how the manager creates a questionnaire. In the state *Defining questionnaire* the questionnaire is composed when the manager adds competences (See 5.3). As an example if a manager adds the competences *C#* and *JavaScript*, the system automatically generates two questions which contain the specific competences selected.

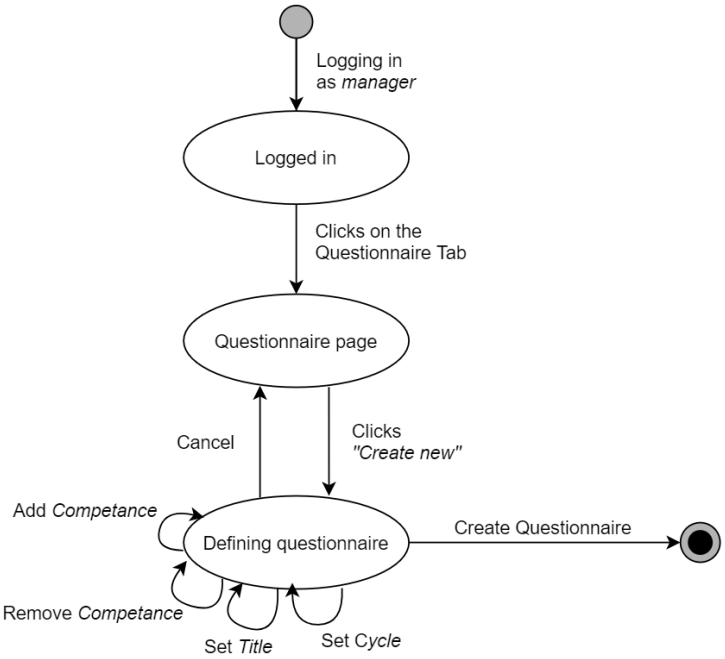

Figure 5.3. Statechart diagram for Create Questionnaire

Take questionnaire

This diagram describes how the employee answers a questionnaire given by the manager.

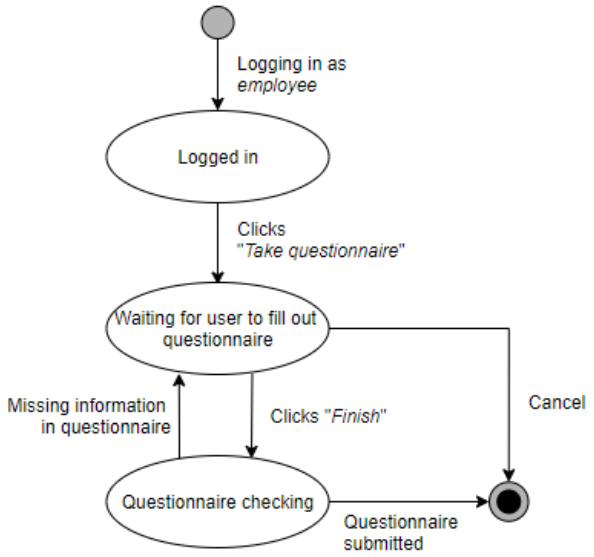

Figure 5.4. Statechart diagram for Take Questionnaire

Compose team

This statechart describes how the manager composes a team with team member suggestions based on the different employees information, see figure 5.5. In the general state *Waiting for team creation* members can be added and removed from a team. In the side process *Waiting for team criteria* a questionnaire can be added. By adding a questionnaire, the competences

which the questionnaire consist of, is automatically displayed for the manager. As an example if a questionnaire asks employees about their competences in *Photoshop* and *Premiere pro*, these two competences will be displayed for the manager, if the questionnaire is added. When the questionnaire is added, the manager can set the competence criteria from 1-5 (5: very important).

The whole process is iterative, and ends when the manager is satisfied with the team, and press create team. The team composition is heavily supported by the competence synergy scores and the display of the personality type distribution.

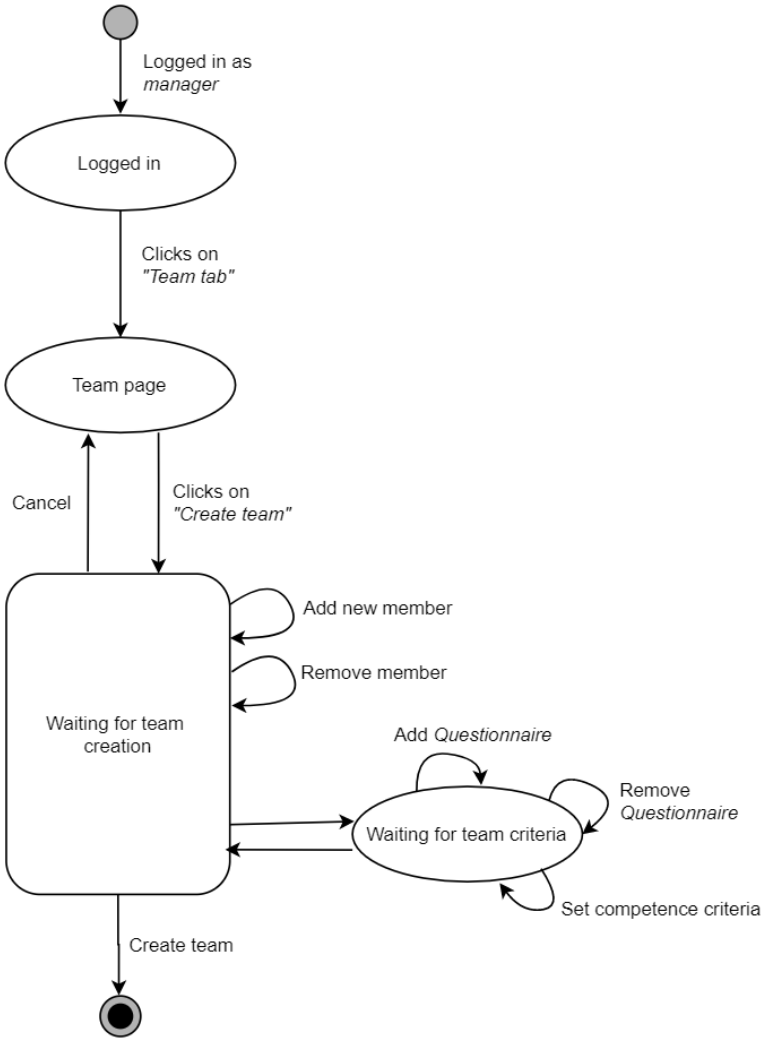

Figure 5.5. Statechart diagram for Compose Team

5.2 Functions

In the following section the functions of the system will be established. This is vital step for the later implementation of the system, as they lay a blueprint for later development and structuring of functions in the system.

5.2.1 Theory

Functions are facilities that makes the *Model* useful for the actors (Lars Mathiassen, 2000, p. 140). The intent of a system can for this reason be identified through the functions. Upon initiating a function, it is executed and provides a result in the *Model*, application domain, or problem domain. The functions must support the actors' use cases and should cover all parts of the *Model* (Lars Mathiassen, 2000, p. 143). The result of the function activity is a complete list of functions as well as detailed descriptions of complex functions.

There are four types of function: update-, read-, signal- and compute-functions. *Update functions* change the model's state and are initiated by a problem domain event. These functions often cover more than one event (Lars Mathiassen, 2000, p. 144). *Read functions* display relevant information in the *Model*. These are initiated in the application domain, when the user needs information to complete a work task. *Signal functions* are initiated in the *Model* and display information in the application domain or directly intervene in the problem domain (Lars Mathiassen, 2000, p. 140). These functions are initiated based on programmed rules. For example if a user should complete a task every 6 months, then a signal function in the system will be responsible for notifying the user every 6 months. *Compute functions* are comparable to the read functions. The difference is that a read function simply returns data, whereas a compute function handles the data before returning it (Lars Mathiassen, 2000, p. 145). These functions must be identified from the use cases.

It is important to notice that most system functions are mixtures of function types. The purpose of categorizing the functions is to increase the understanding of their character, and to assist the application-domain analysis (Lars Mathiassen, 2000, p. 141).

The complete list of functions is identified through the use cases, classes, and events. Update and read functions are typically identified in the classes, and events identify requirements for update functions (Lars Mathiassen, 2000, p. 143). Use cases can be used to identify all four types of functions.

When creating the list of system functions it is important to specify the complexity of each function. This can be done with a four-point scale: Simple, medium, complex, and very complex. Most of these functions should only be described briefly, whereas a detailed description can be used for special cases (Lars Mathiassen, 2000, p. 146). A detailed description can for example be constructed by pseudo code or by partitioning a function into multiple simpler functions in a complete list of functions.

5.2.2 Analysis

We identified the required functions by systematically analysing for each function type. When identifying the update functions we analysed each of the events mentioned in section 4.1.2. We made sure that all of the events were covered by a function. As mentioned, the update functions often cover more than one event, which also applied to our analysis. We identified the read functions by analysing the actors and the use cases. In this context we focused on what information the actor had to know in order to complete their use case. A similar approach was also used for analysing the compute functions. The difference is that for finding compute functions we also identified when the function had to process the data. Furthermore we also analyzed the problem domains' *Model* in order to identify the signal functions.

Table 5.4 shows the complete list of identified functions.

Function	Complexity	Function type
Register employee	Simple	Update
Delete employee	Simple	Update
Update employee information	Simple	Update
Create team	Simple	Update
Delete team	Simple	Update
Update team information	Medium	Update
Create questionnaire	Simple	Update
Delete questionnaire	Simple	Update
Update questionnaire information	Simple	Update
Send questionnaire reminder (e-mail)	Medium	Signal
Read competences	Simple	Read
Read employees	Simple	Read
Read teams	Simple	Read
Compose Questionnaire	Medium	Read
Validating login	Complex	Compute
Calculate team competence synergy	Very complex	Compute

Table 5.4. List of functions

Detailed function description

Most of these functions are intended to be understood simply by reading the function name. However, this is not the case for "validating login", “compose questionnaire" and “calculate competence synergy”. These functions will therefore be described in this section.

Validating login

The login function is described with pseudo code, which is seen in figure 5.6. This function makes it possible to control the manager and employees access to the system.

```
Requesting login
    given Username/email  and Password
    search objects in Employees and select those
        who    match Username and Password
               where the given Username is marked as Manager
    result if Username is marked as manager, login as manager or employee. If
    Username is not marked as manager login as employee.
```

Figure 5.6. Validating login pseudo code

Compose questionnaire

The function is best described by partitioning it into multiple simpler functions. This is shown in figure 5.7.

Compose questionnaire	
• Generate form-inputs based on the selected field competences. • Create form with generated questions.	*medium *simple

Figure 5.7. Compose questionnaire function description

Calculate team competence synergy

One of the synergies that needs to be calculated is the team competence synergy. This function calculates how well an employee's competences match a team's competence-criteria. This is fairly complicated and is elaborated in detail in figure 5.8.

Calculate competence-synergy

1. The competence criterias are given numeric values.
   - "Very important" numeric value is 5, and "Not important at all" numeric value is 0, and so on.

2. The sum of the numeric values of the competence criterias is calculated.
   - For example
     - C#:: Very important (numeric value: 5)
     - PHP: Not important for the team at all (numeric value 0)
     - MySql: Medium priority (numeric value: 3)
     - Max total score = 5 + 0 + 3 = 8

3. The employee competence scores are then gathered from the database.

4. Each of the employee competences is then compared to the respected team competence criteria.
   - For example:
     - Score = 0
     - An employee scores 3 at c#
       - Team criteria for c# was 5. This determines the max score for this competence. 3 is below 5, so 3 is added to the employee's score.
     - Employee scores 4 at MySql
       - Team criteria for MySql was 3. Therefore 3 is the max possible score for MySql and 4 is above, so 4 is rounded down to 3. 3 is then added to score.
     - Employee scores 4 at PHP
       - Team criteria for PHP was 0 which is below max score of 4, so 4 is rounded down to 0, and therefore nothing is added to score.
     - Score is now 3 + 3 + 0 = 6

5. The employee competence synergy is calculated in percent (score / max total score * 100).
   - Example:
     - Employee competence synergy = 6 / 8 * 100 = 75 %.

Figure 5.8. Calculate competence synergy function description

Visual Design 6

An important part of the application, is to define the visual design or interface, as it heavily influence the actors interaction with the system and their presumptions about it.

6.1 Theory

Based on the previous analysis work, it is possible to create a comprehensive image of the system, to create a representation of developers ideas, which can serve as a proficient tool, both for developers and involved actors in the design process. In this case we will be using David Benyons theory regarding envisionment, as described in his book *Designing interactive systems* (Benyon, 2014). This book describes the utilisation of envisionment-techniques which are methods that are used in different areas of the design process. The typical trend is going from low fidelity techniques such as sketches moving to higher fidelity techniques.

6.1.1 Sketches

Sketches seek to provide a quick visualisation of the ideas regarding the design of the application. The benefits of the sketch is that it can be iterated upon and no real loss is given, if you need to reject a sketch or start anew based on internal requests by designers or external parties such as collaborators. Sketches can also provide a powerful medium, for bridging the gap between designers and collaborators, which serves as being more comprehensive than a list of functionalities (Benyon, 2014, p. 168). Additionally, sketches can also be used as a tool for brainstorming for designers developing the application.

6.1.2 Navigation maps

Navigation serves as an essential part of systems. Based on this, a focus on intuitive navigation needs to be emphasised. A navigation map is often used in the beginning of the design process, where the collaborators' system requirements are translated into a logical navigation structure. The navigation map is often symbolised as a hierarchy tree, where you move from the homepage to subpages, although many different compositions of navigation maps exist. Each element in the navigation map represents a page in the application, where the lines connecting them describes the possible path that the user can navigate through the application (Benyon, 2014, p. 172).

6.1.3 Wireframes

As navigation maps focus on the structure of the application and navigation between pages, wireframes focus on the structure of each page, and provide a crude drawing of the elements that exists on each page. The advantage of wireframes is that they give a good understanding

of how the page will look like in the future, without spending too much time on picking text, images and focusing on details (Benyon, 2014, p. 173).

6.2 Analysis

Based on previous work, we will now tie the gathered data together and represent the system visually. This is both useful to give developers as well as involved actors with a frame of reference to discuss design choices of the system.

6.2.1 Market research

Before beginning the process of representing the actual system, an emphasis on gathering data on existing systems was made. To get a general idea of which aesthetics and functions, we wanted to provide, and what the existing market was providing, an example of the general direction of visual representation can be seen in figure 6.1.

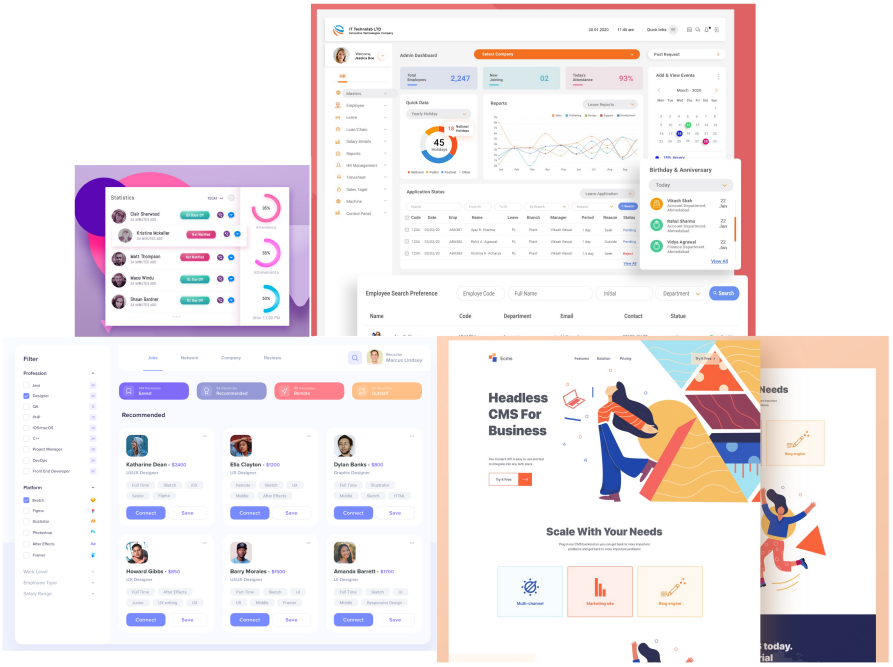

Figure 6.1. Visual representation of data gathering

From this market research it was evident that the general trend within the market was to utilise flat design. This is generally represented by the lack of shadows provided on elements such as buttons, containers and menus. Additionally the trend seemed to be the use of pastel colors in a simple color scheme often bringing a high contrast on items to a white background, with no use of gradients or similar effects. These trends inspired our own visual design choice.

6.2.2 Navigation map

To get an idea of the navigation between pages, a navigation map was created. This seeks to provide a fundamental structure where functions will be in the application. The navigation map can be seen in figure 6.2.

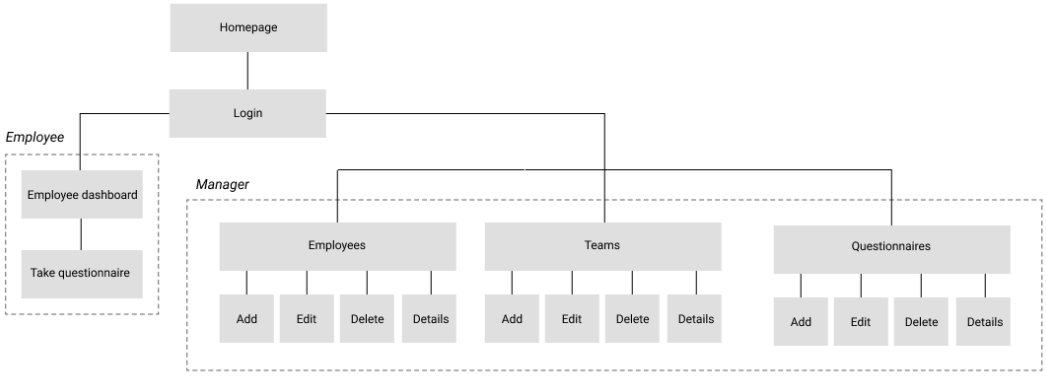

Figure 6.2. Navigation map

As aforementioned the navigation map shows the paths that a user can take on the page. It is important to explain the varied user navigation possibilities. This is caused by the differences in functions that are provided for employees and managers, where the employees are provided with relatively simple user navigation as seen on the left of the figure 6.2. Furthermore the manager section of the application, routes the user to the teams page upon logging in, as this is regarded as the main functionality of the system.

6.2.3 Wireframes

With an abstract idea of the systems structure provided with the developed navigation map, we can now seek to specify the systems content further. This can be done by placing our previously described functionality in section 5.2. in each of the pages shown in the navigation map.

Homepage

The homepage provides a general insight into the application and its features. The navigation bar is simple and only provides the showcase of a logo and a login button. This is visible on all pages of the application, see figure 6.3. A header is also shown that contains either a permanent picture with a text overlay, or a slideshow providing information that might be useful to companies considering using the application. Further down a section with images and text is provided, that will display the main features and services that the application provides.

Figure 6.3. Homepage

Login

When the user clicks *login* on the homepage, the user is redirected to a login-page which provides a simple form. The form navigates the user to either the employee or manager area of the system based on the credentials that is inserted, see figure 6.4. If a manager logs in to the site, a prompt is shown, asking if they want to login as an employee or a manager. This is caused by possible instances, where a manager might also use the system as an employee.

Figure 6.4. Page with login

Employee dashboard

This page is shown to employees upon login, which showcases a general overview of information that might be relevant to them, see figure 6.5. On the page a general overview of the teams

they are a part of can be seen. Furthermore they can see if there are pending questionnaires for them to fill out.

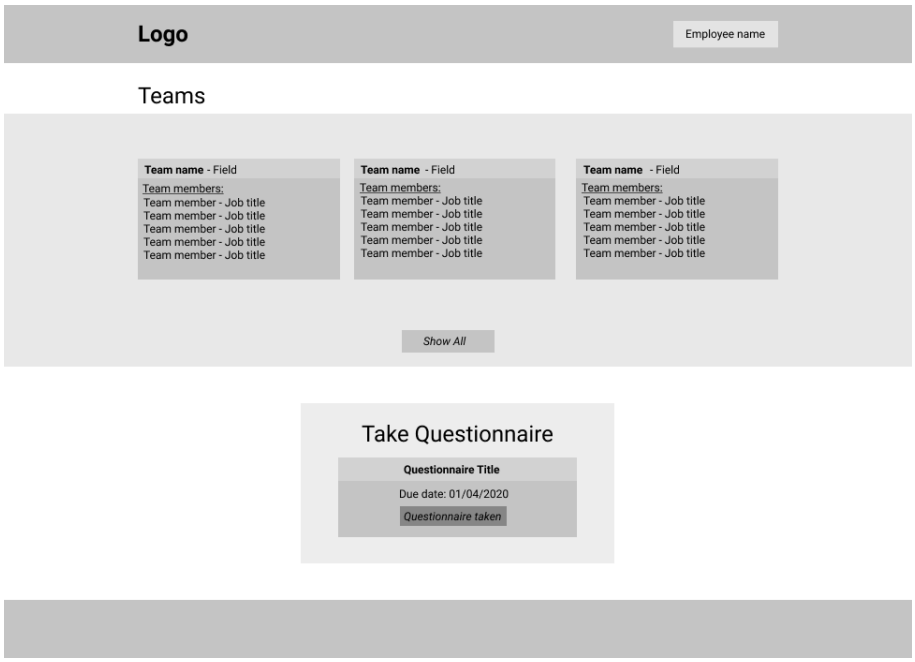

Figure 6.5. Employee dashboard

Team

Once logged in as a manager, the team page is shown, as this is deemed the page that provides the best overview of the applications functionality. This showcases the teams, their synergy score and number of team members, see figure 6.6. This page also provides shortcuts for adding and editing teams as well as a short path to the employee or questionnaire page.

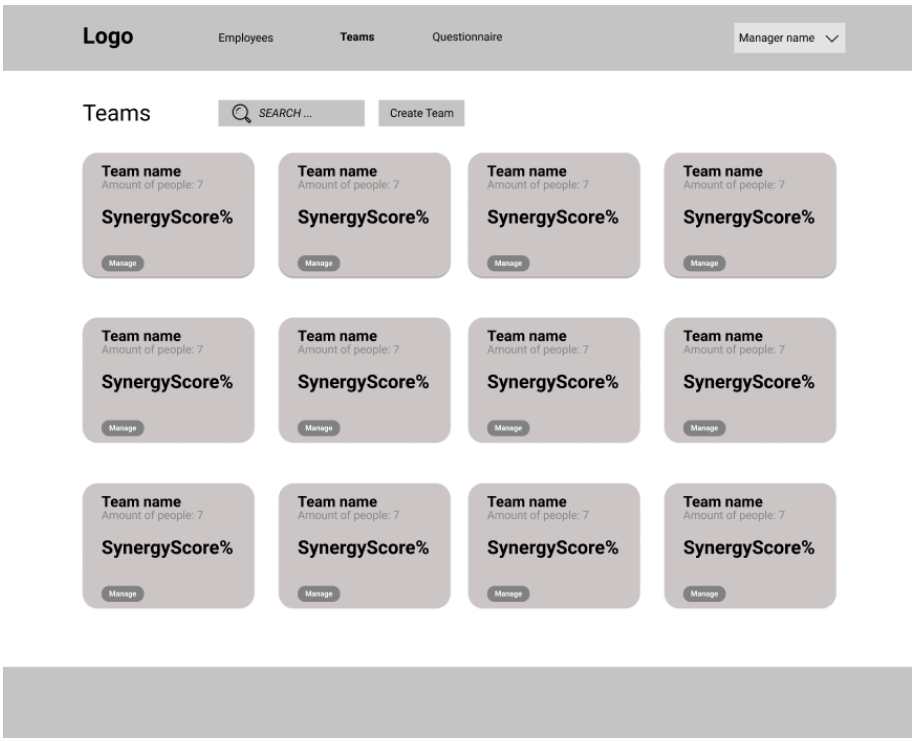

Figure 6.6. Team wireframe

Questionnaire

The questionnaire page in figure 6.7 showcases a preview of the relevant data of the issued questionnaires in a card view setup. This is similar to the team and employee page. Additionally a search feature and a shortcut to adding and editing questionnaires are provided.

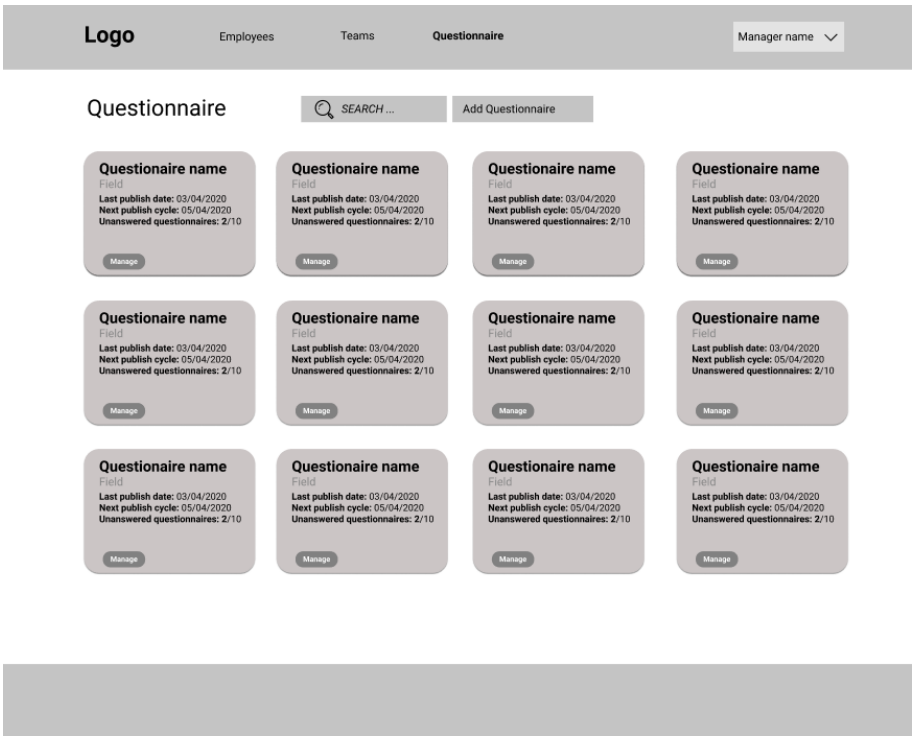

Figure 6.7. Questionnaire wireframe

Employees

The employee page in figure 6.8 gives an overview of all the employees that the manager is in charge of. The employees are listed with name and job title and additionally key information of their highest ranking skills, personality type, a search feature is also inserted just like in the team and questionnaire page.

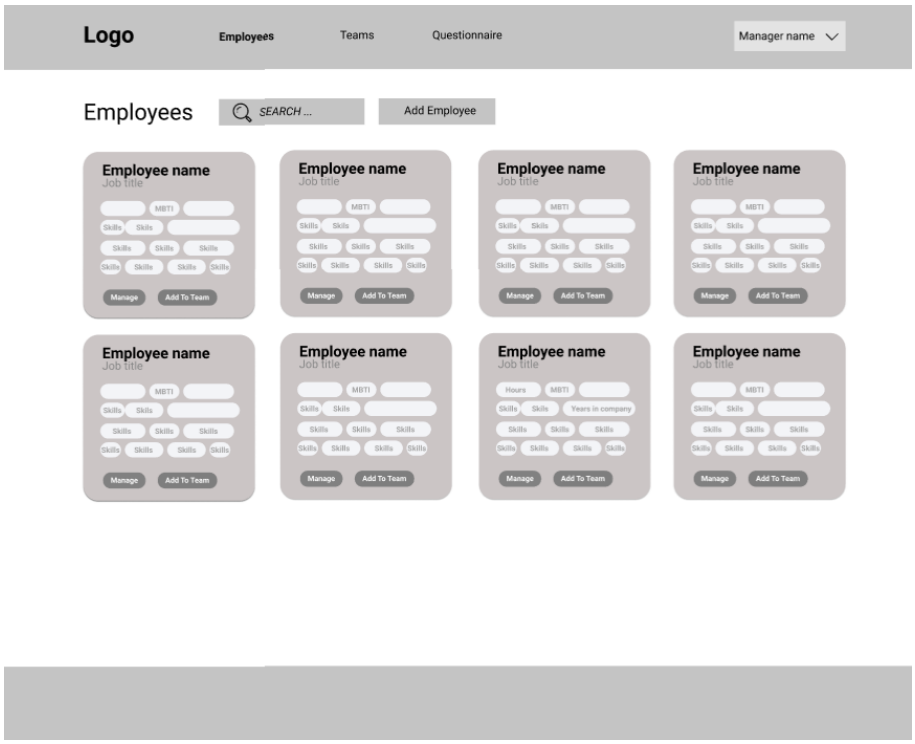

Figure 6.8. Employee page

Take questionnaire

The take questionnaire page is used by the employees when they press the "*Take questionnaire*" button on the employee dashboard, see figure 6.9. It presents a series of questions defined by the manager. The possible answers will change depending on the question, ranging from radio buttons to a drop-down menu.

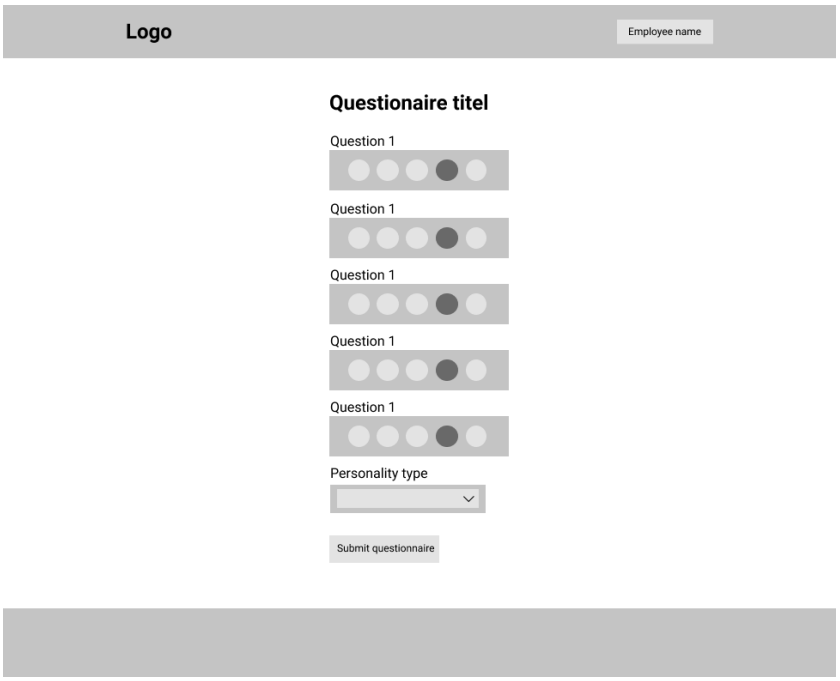

Figure 6.9. Take questionnaire page

Create teams

On the create teams page as seen in figure 6.10, the criteria for calculating the team competence synergy is selected. Based on the selection on the left hand column, the system will suggest members in order from best to worst fit. The possible options include what personality types and what competences should be prioritized in the team. The manager is able to add and remove members in the right column and view the current user composition. The total team score is also shown in this column and an overview of the personality types added to the team is shown at the top of the right column. Once the team criteria and members have been selected, the team can also be created from this page.

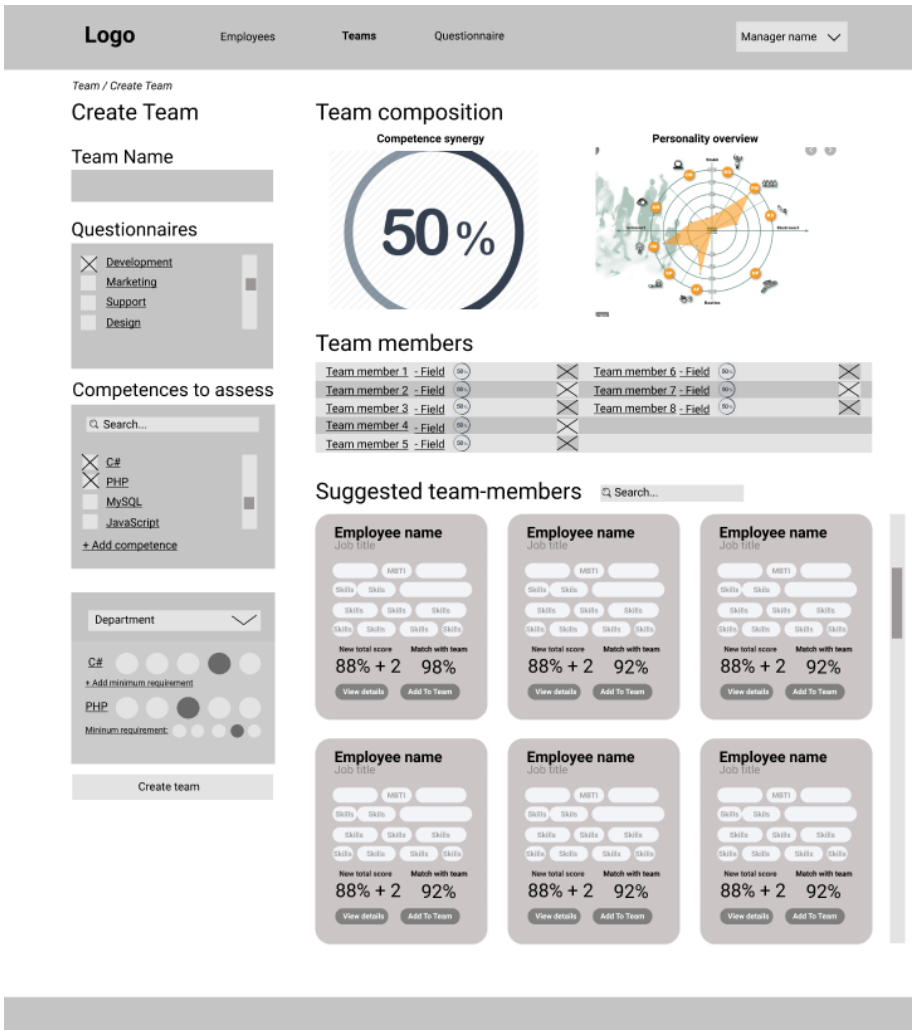

Figure 6.10. Create team wireframe

Add or edit questionnaire

The add and edit questionnaire page seen in figure 6.11, uses the same structure, but has different functionality. The add page is for adding questionnaires, and the edit page is for editing existing questionnaires. Using the form, the manager is able to select what competences the questionnaire should access.

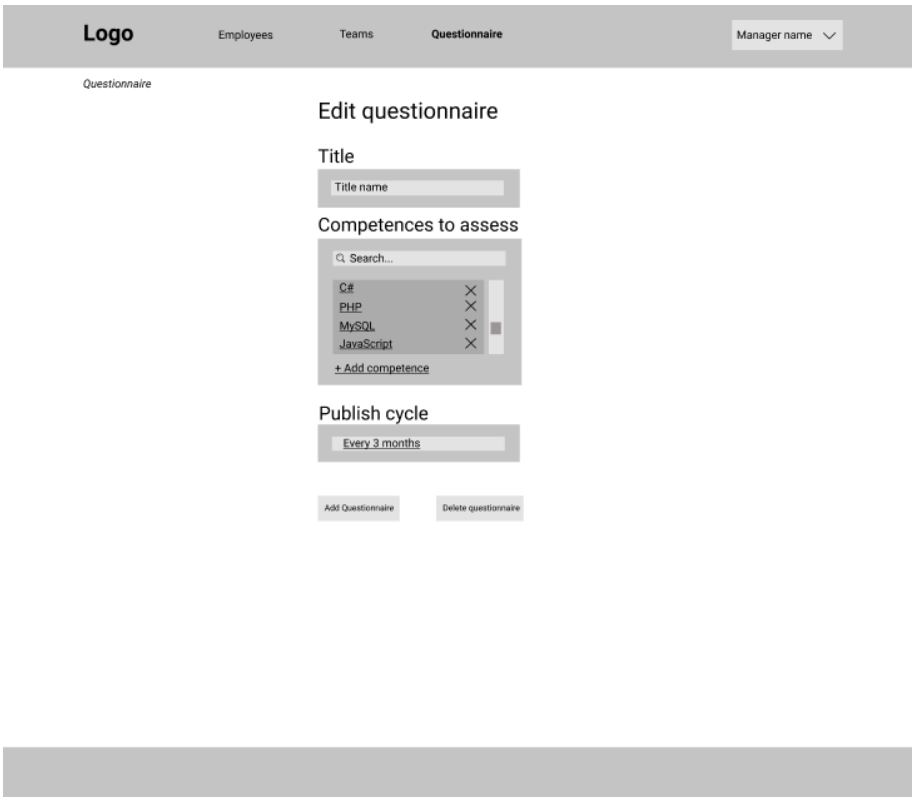

Figure 6.11. Add or edit questionnaire wireframe

Add or edit employee

Similar to the add and edit questionnaire page, the page structure is shared, which is seen in figure 6.12. The functionality is also slightly different. The add employee page is for adding employees, and the edit employee page is for editing already existing employees. For an employee to be included in a team they need to be registered in the system from this page beforehand. Using the form, the manager is able to fill in basic information about the employee to the system. Some of this information will be used to calculate their fit in specific teams whilst other information is used for identification of the employee. The submitted information can be edited using the edit employee page.

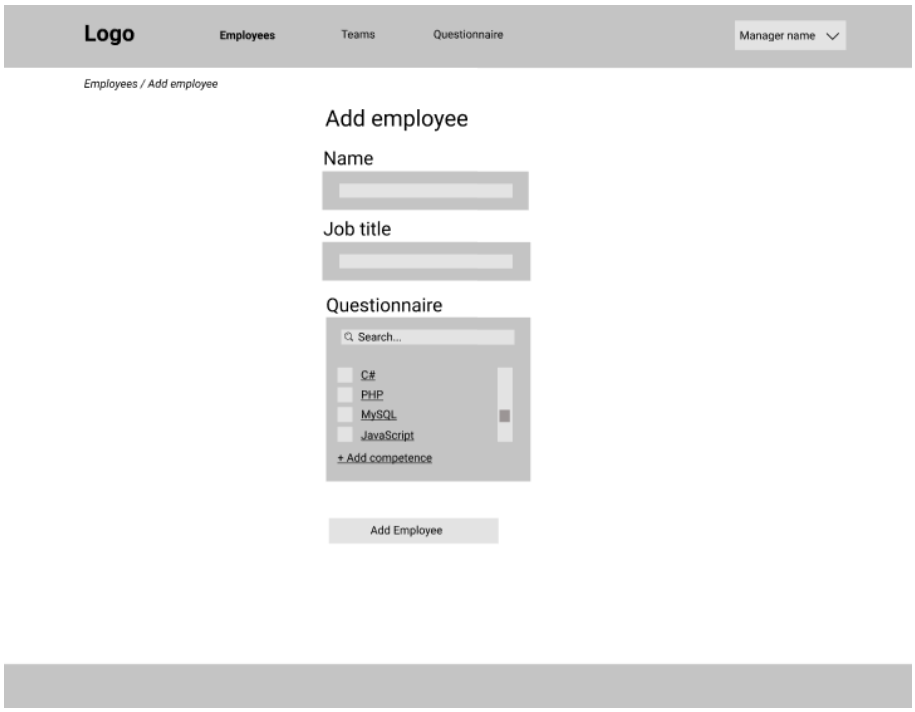

Figure 6.12. Add or edit employee wireframe

Architectural Design 7

The architectural design of a system describes the method of structuring individual components from design criteria to make up the system as a whole. The purpose of the architecture is also to support development activities (Lars Mathiassen, 2000, p. 127). In this chapter our focus will be on describing the primary criteria for the application as well as analysing what components the system will consist of, to support future development-activities.

7.1 Criteria

7.1.1 Theory

The purpose of the criteria activity is to set design priorities and to arrive at a good design (Lars Mathiassen, 2000, p. 179). This is achieved through design criteria which are defined as: *"A preferred property of an architecture"* (Lars Mathiassen, 2000, p. 180). Many different design criteria exist, and in the context of software the criteria categories, seen in figure 7.1, are recommended to use. A criterion could be *usable*, which measure the systems ease of use (Lars Mathiassen, 2000, p. 180).

It would be easy to assume that a good design simply is achieved by highly prioritising each criterion. But this is not an option in almost any system development case, because of surrounding conditions. Conditions can be technical, organisational or human limits (Lars Mathiassen, 2000, p. 180). An example of an organisational limit can be financial limitations such as a budget, which can eliminate the option of highly prioritising all criteria. The final result of the activity is a table which contain all criteria, and a prioritisation of each criterion (see figure 7.1).

7.1.2 Analysis

The design criteria used in our project are the same as the suggested criteria in the book *Object oriented analysis and design* (Lars Mathiassen, 2000). Each criteria has been prioritised based on the conditions and the purpose of our IT-system. The result is shown in figure 7.1.

	Very important	Important	Less important	Irrelevant	Trivially fulfilled
Usable		(x)	x		
Secure			x		
Effective				x	
Correct		x			
Reliable		x			
Maintainable			x		
Testable			x		
Flexible			x		
Understandable		x			
Reusable				x	
Movable				x	
Integratiable				x	

Table 7.1. Prioritisation of criteria

To further elaborate upon our prioritisation, the criteria will be explained below.

Usable
It is less important that the system is easy to use, because the focus of our project is on the development of functionality. However if the system is later being implemented in an organisation, an actual test of the systems usability should be conducted.

Efficient
Our focus in the development will not be on the economical utilisation of our system's technical facilities, as the focus is on the application's core functionalities. which is why this criterion is irrelevant.

Secure
It is less important that the system is secure, as the scope of this project is firstly to develop a system that can support the process of composing teams. If the system is going to be adopted by an organisation later on, security must be a priority because of sensitive employee data that the system handles.

Correct
This criterion is important as the primary focus is on functionality of the system and the fulfillment of the functional requirements.

Reliable
It is important that the functions are reliable and trustworthy. Many functions are interconnected, therefore if one function does not work correctly then it will have a negative impact on the rest of the application.

Maintainable
The cost of identifying and fixing bugs in the system is not a high priority. However the focus is still on the readability of the code. Therefore the criterion is set to less important.

Testable
It is less important that the system is testable, because the focus is not on the cost of ensuring

that the deployed systems works as intended. However if the system is implemented in an organization, it will be of higher priority.

Flexible
Is not that important that the system is easy to modify later on. But some focus will be on the readability of the code and its structure, which is why this criterion is not completely irrelevant.

Comprehensible
The system needs to be easy to understand. It is important that the design of the system is understandable, so confusion does not arise. Therefore a lot of focus has been on explaining design choices, and the structure of the system (e.g. class diagram). Furthermore it is also important in regards to future stakeholders, and therefore the criteria is set to important.

Reusable
It is not a focus that the system's parts can be used in other related systems, and re-usability is therefore an irrelevant criteria.

Portable
The focus is not on how much it costs to move the system to another platform. The intent of our system, is that it can easily be used on most modern internet browsers, and the criterion is therefore irrelevant to the system.

Interoperable
This criterion is irrelevant, as the focus is not on the cost of integrating the system to other systems.

7.2 Components

7.2.1 Theory

A component architecture improves the understanding of the system structure. It is composed by interconnected components, which is a collection of program parts that has well-defined responsibilities (Lars Mathiassen, 2000, p. 191-194). Many of these parts will be classes identified earlier in the process. The different components should address different concerns and by dividing them the system will be more comprehensive and flexible (Lars Mathiassen, 2000, p. 191-194).

The architecture pattern can be presented in different ways. One way is the layered pattern, where each layer has a well defined purpose and interface with the layers above and below it. Each component describes its responsibility and the connection to other layers. A connection downwards describes which operation there can be accessed in the layer below. The upward interface describes which operation the upper layer has access to (Lars Mathiassen, 2000, p. 195-197). If the layer contains more than one component it can be split into parts. Parts in the same layer does not have any essential interaction with each other. Each part can relate to different components in each layer. This will make the system more flexible and possible to modify parts individually (Lars Mathiassen, 2000, p. 195-197).

The layered pattern can be used to generate basic systems. It can be used in the generic architecture pattern, which contains interfaces, functions and model components. Here the

model component is the lowest layer, connected to the function layer in the middle and on the top of the interface layer (Lars Mathiassen, 2000, p. 198). The relation between these can be seen in figure 7.1.

7.2.2 Analysis

In the architecture design for our system the layered and generic architecture is used. In figure 7.1 the overall architecture design is shown. The design includes a User Interface component, a Function component, a Model component and a Technical platform component. Each of these will be described in the following sections.

Most of the connections are downwards. For instance the User Interface component has access to operations in the Function component. Because of the upward arrow between these, the Function component can access operations in the User Interface component too. The layers are dependent on each other, so when changes are made in one component it can lead to changes in another.

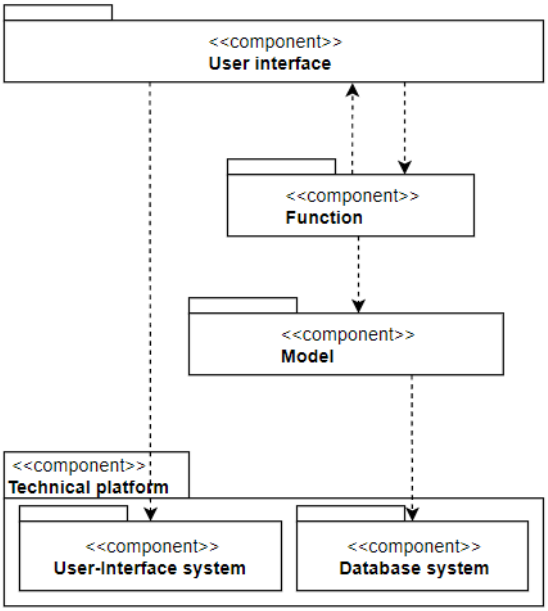

Figure 7.1. Component design

User Interface component

The component at the top is the User Interface, see figure 7.1. This component contains the interface of the system in which the user has to interact with. It displays information to the user and holds client-side functionality, see figure 7.2. The component is related to the Function component and the User-Interface System component.

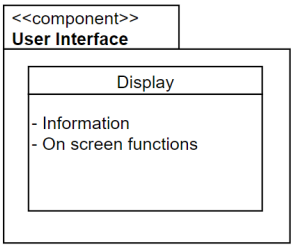

Figure 7.2. User Interface component

Function component

The Function component contains all the functionality in the system, see figure 7.3. The different functions affect different data, for example the employee functions affect the employee data. Here it is possible to create, change and delete information regarding employees. The function component is connected downwards to the Model component and upwards to the User Interface component.

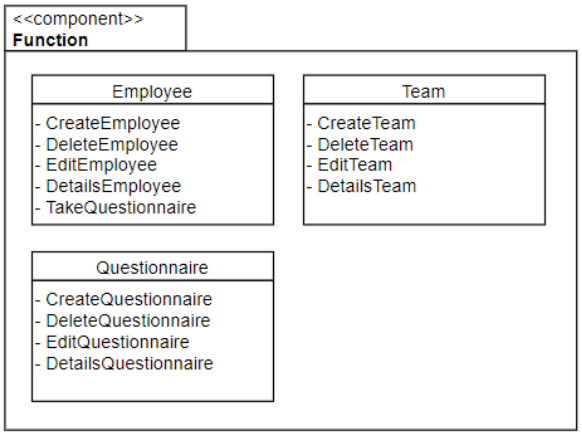

Figure 7.3. Function component

Model component

The Model component contains all the information that the system will keep track of. In our iterative work process we concluded that the employees and teams should no longer be assigned to a single manager, but rather all of the managers. After this change, the employee class and the manager class is very similar. The only difference being the manager role. The information is therefore combined to one table instead of two. The different models and their associated properties are shown in figure 7.5. How the structure and design of the models are, can be seen in figure 8.9 in section 8.2. The Model component is connected downwards to the Database system component.

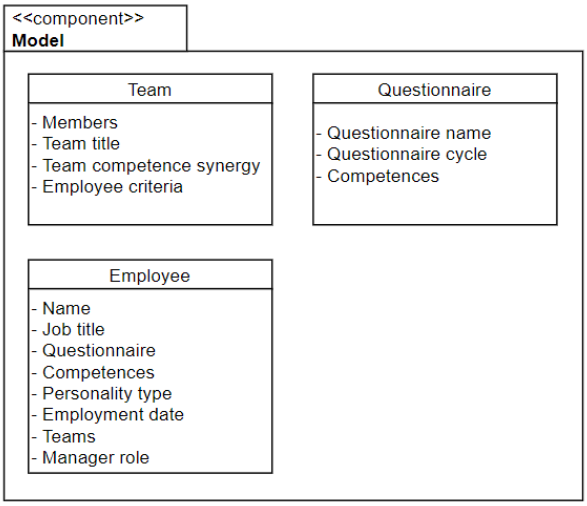

Figure 7.4. Model component

Technical platform component

The technical platform contains two parts: User-Interface system and Database system. The User-Interface system holds the used libraries and frameworks to aid the visualisation of our design, which will be further explained in chapter 8.

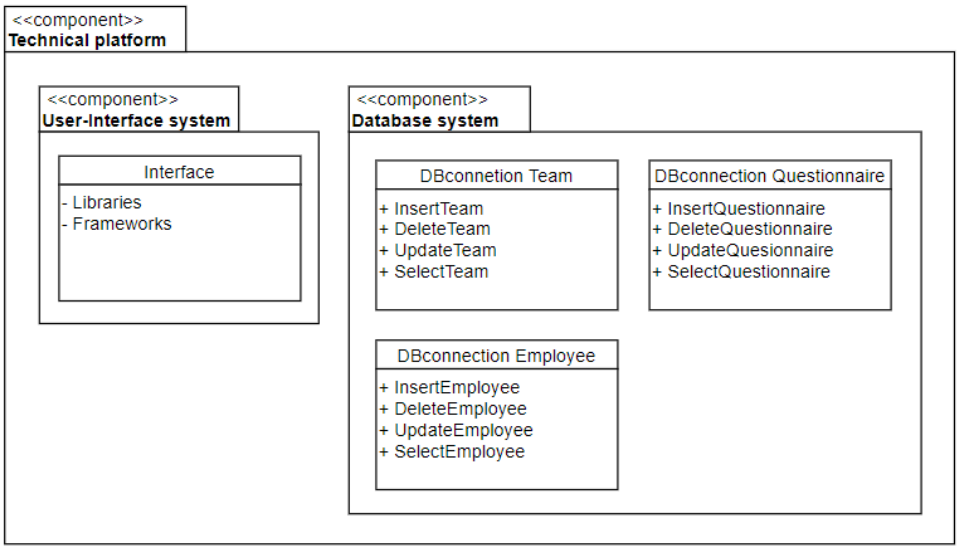

Figure 7.5. Technical platform component

This component design follow the design convention in ASP.NET Core Razor Pages. Our use and design of this will be elaborated in chapter 8.

Implementation 8

Building upon our previous discoveries in the object oriented analysis and design, this chapter will look to expand upon the implementation of the system. This will include the use of different technologies as well as how they work together to create the final system. The following sections will also include the system architecture, further expanding upon the MVVM[1] and MVC[2] architecture, database design, data models. Finally a walk-through of some of the core functionality will be explained in section 8.3.

The system will utilise object oriented programming, as it is a natural next step after the object oriented analysis and design-phase. The primary programming language in the application is C# as all authors of this paper has some degree of familiarity with this language. As the system is a web application, we will furthermore make use of the ASP.NET framework, that serves as an extension of C#, specifically aimed towards building web applications.

Below is an overview of the used technologies, libraries and frameworks, that have supported the development of the application in relation to the front-end. Back-end technologies and frameworks will be described in the database design section 8.2.

HTML is used to mark up the structure of the content on the web page.

CSS is used for styling HTML elements.

Javascript is used to provide dynamic content that updates on the client side based on user interactions.

Bootstrap is a front-end framework that in this applications context provides basic styling and responsiveness to the web application and speeds up the development process.

jQuery is a Javascript framework that makes for a simplified event-handling, document traversal and manipulation of data. This is primarily used in the context of validating user input in the application.

Chart.js is a Javascript library that is used to provide more advanced charts, with the developer only required to provide the correct input to the library. This library is used to aid in providing the radar chart as seen in figure 8.1.

[1] Model - View - ViewModel
[2] Model - View - Controller

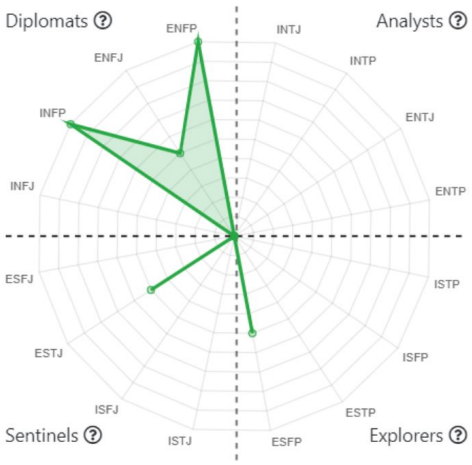

Figure 8.1. Using Chart.js to create a radar chart, to display the team personality distribution

Tagify.js is a Javascript library that supports the use of tags as user input. This both allows for dynamic creation, editing, and deletion of the input. The library is used to remove some unnecessary complexity for the developers during the development of the application (see figure 8.2).

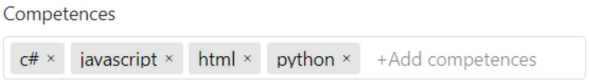

Figure 8.2. Using Tagify.js to get customised input for competences in questionnaires

RadalIndicator.js is a Javascript library that serves as a lightweight indicator, which in the application is used to show the total team synergy. This is shown in figure 8.3.

Figure 8.3. Using Radalindicator.js to display team synergy

8.1 Architectural design in practice

A MVC structure is primarily used with the ASP.NET framework. This enables a faster development process and makes parallel development by more developers achievable, primarily because of the separation of logic in the application. This structure also enables groupings of related views[3], and furthermore enables multiple views for a single *Model*. This makes for a more robust application, with a high degree of code reusability and modularity, so multiple developers can work on the application simultaneously.

[3] A *View* is the code that displays content to the user

In later years Microsoft has developed an alternative way to implement a MVC-like structure called Razor Pages. Razor Pages works as an abstraction over the regular MVC-core approach. This makes for a page-focused development process, that bundles the programming logic and *View* together as seen in figure 8.4. This is defined as a page. This is done with the purpose of making it easier to navigate the code, and reduce unnecessary complexity for smaller web-applications (Microsoft, 2017). The implementation of the architectural pattern builds upon the components defined in section 7.2.

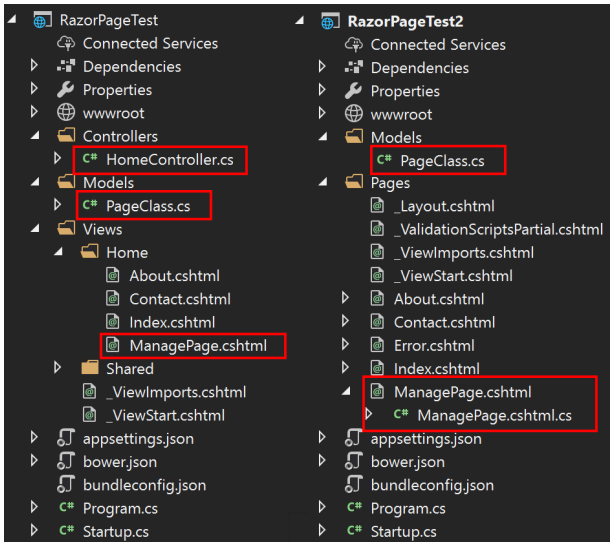

Figure 8.4. Difference between the file structure of ASP.NET MVC (1) and Razor Pages (2)

Razor Pages uses a MVVM architectural pattern, where instead of using controllers *ViewModels* is used that serves as an intermediary to enable two way data-binding[4] between the view and model. This greatly decreases the navigational and complexity issues, that can be associated with the MVC architecture. This theoretically enables faster development with a greater overview of the application (Microsoft, 2017). In figure 8.5 a general overview of the MVVM structure can be seen.

Figure 8.5. The MVVM architectural pattern

In the following sections the basic setup of the MVVM architectural pattern and its correlation to the Razor Page structure will be explained.

[4]Data related changes in the *Model*, is automatically sent to the *View* and vice versa.

8.1.1 Application structure

Now we will look at the structure of the application through a simplified version of the code with core elements that represents the code structure throughout the application. Furthermore we will also explain each areas relation to the MVVM architectural pattern.

Razor Page initialization

Razor Pages are enabled within startup.cs with the *ConfigureServices* method (see listing 8.1), which also sets a database context for the *PageModel*. This will be expanded further in section 8.1.1.

```
1  public void ConfigureServices(IServiceCollection services)
2        {
3            services.AddRazorPages();
4
5            services.AddDbContext<TeamberContext>(options =>
6                  options.UseSqlServer(Configuration.GetConnectionString("TeamberContext")));
7        }
```

Listing 8.1. Example of startup.cs with *ConfigureServices* method.

The code seen in listing 8.1 enables the use of razor specific syntax and methods, and with that the previously mentioned MVVM architecture, so that we can use built-in directives such as *@page* and *@model*.

The routing on the page is configured using the *Configure* method. This sets up the correct routes between pages. The method is stored in the same file, and can be seen in the following code (see listing 8.2).

```
1  public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
2        {
3            app.UseEndpoints(endpoints =>
4            {
5                endpoints.MapRazorPages();
6            });
7        }
```

Listing 8.2. Example of *Configure* method.

Page

The code in listing 8.3 displays the *View* of the application that serves as the entry point to the software for the user. The *@page* razor directive enables the page to handle requests, which enables razor-specific syntax for displaying data from the *ViewModel*. In short this means that the page can handle requests without a *Controller*. The *@model* razor directive serves as a reference to the *PageModel* for the page, shown in section 8.1.1.

```
1  @page
2  @model IndexModel
3  @{
4      ViewData["Title"] = "Create";
5  }
6  <body>
7  <div class="form-group">
8  <form method="post">
9            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
10           <div class="form-group">
11               <label asp-for="Employee.LastName" class="control-label"></label>
```

```
12              <input asp-for="Employee.LastName" class="form-control" />
13              <span asp-validation-for="Employee.LastName" class="text-danger"></span>
14          </div>
15      </div>
16  </div>
17
18  <div class="form-group">
19                  <input type="submit" value="Create" class="btn btn-primary" />
20              </div>
21  </form>
22  </div>
```

Listing 8.3. Example of @page razor from file-destination Pages.cshtml

PageModel

The *PageModel* shares the same name as the page by convention extended with .cs as it's file extension. It acts as a *ViewModel* in the MVVM architecture, and implements the Page Controller pattern seen in figure 8.6. This can be described as *"An object that handles a request for a specific page or action on a Web site"* (Fowler, 2002).

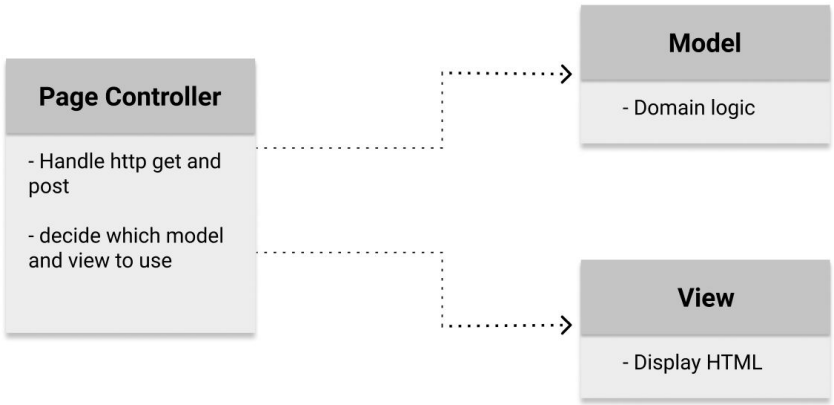

Figure 8.6. The pagecontroller pattern that is used by the *PageModel*

This is done through handlers such as seen in listing 8.4. This showcases the handling of http get and post requests between the server and client-side. This is done with the method *OnGet*, shown on line 8-14, that gets data from the *Data Model*, more specifically the employee and employees team (*EmpTeam*). The *OnPost* method on line 16-38 handles user input from the form on the page, and sends it to the *Data Model*. A more detailed description of the *OnGet* and *OnPost* methods are provided in section 8.3.2.

```
1  namespace Teamber.Pages
2  {
3     public class IndexModel : PageModel
4     {
5     [BindProperty]
6     public Employee Employee { get; set; }
7
8         public void OnGet()
9         {
10        var employee = new Employee();
11        employee.EmpTeams = new List<EmpTeam>();
12        PopulateAssignedTeamData(_context, employee);
```

```
13        return Page();
14        }
15
16        public void OnPost()
17        {
18        var newEmployee = new Employee();
19                newEmployee.EmpTeams = new List<EmpTeam>();
20                foreach (var team in selectedTeams)
21                {
22                    var teamToAdd = new EmpTeam
23                    {
24                        TeamID = int.Parse(team)
25                    };
26                    newEmployee.EmpTeams.Add(teamToAdd);
27                }
28             await TryUpdateModelAsync<Employee>(
29                newEmployee,
30                "Employee",
31                i => i.FirstName);
32
33                _context.Employees.Add(newEmployee);
34                await _context.SaveChangesAsync();
35                return RedirectToPage("./Index");
36
37            PopulateAssignedTeamData(_context, newEmployee);
38            return Page();
39        }
40    }
41 }
```

Listing 8.4. Simplified example of *@pagemodel* razor from file-destination Pages/Employees/Index.cshtml.cs

Data Model

The code in listing 8.5 showcases an example of a *Data Model* utilising the Entity framework (EF)[5], which uses Object relational mapping (ORM)[6]. This lets us manipulate data from within an object oriented language. This is beneficial when working with smaller applications with limited configuration. The *Data Model* lets us define the content and furthermore uses EF-specific data annotations and built in functionality, that automatically assigns ID (line 5) as primary keys. Furthermore it sets required fields as seen on line 7 as well as display names and string length, which is displayed at line 8-9. We can define navigational properties as seen on line 12, which sets up relations in the database as shown in the code in listing 8.5 using ICollection[7]. This is the default interface used with the Entity framework that lets you create, edit, read and delete items, which results in the *entity* table 8.7.

```
1  namespace Teamber.Models
2  {
3    public class Employee
4    {
5      public int ID {get; set;}
6
7      [Required]
8      [StringLength(50)]
9      [Display(Name = 'Name')]
```

[5]EF is a Object Relational Mapper (ORM) that simplifies mapping between objects in the software to the tables and columns of a relational database.

[6]ORM automates the transfer of data stored in relational databases tables into objects in the software

[7]An interface contains definitions for a group of related functionalities

```
10      public string FirstName {get; set;}
11
12      public ICollection<EmpTeam> EmpTeams { get; set; }
13    }
14  }
```

Listing 8.5. Example of *Data Model* from file-destination Models/Employee.cs

The application utilises a code first approach, where instead of creating the tables in the database first. Instead, *Data Models* are created, that later on will be converted to tables in our database, using a context class which is shown in section 8.1.2.

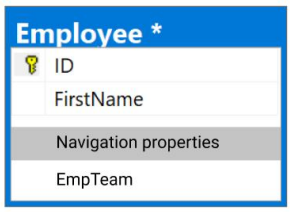

Figure 8.7. An *entity* table created based on the *Data Model*

8.1.2 Domain Context

The context class is derived from the DbContext class that is native to EF. This class works as a connection between our page models and the database, and lets us perform CRUD[8]-operations, without writing queries ourselves. Instead we use ORM to manipulate our data, by using the code first approach. In the context class we use DbSet[9] as seen in listing 8.6 lines 10-11 to define our tables, with the content defined in our *Data Model*. From here we can set the prerequisites for our database creation by writing *add-migration MigrationName* in the package manager console in visual studio.

```
1  using Teamber.Models;
2  using Microsoft.EntityFrameworkCore;
3
4  namespace Teamber.Data
5  {
6      public class TeamberContext : DbContext
7      {
8          public TeamberContext(DbContextOptions<TeamberContext> options) : base(options) {}
9
10         public DbSet<EmpTeam> EmpTeams { get; set; }
11         public DbSet<Employee> Employees { get; set; }
12
13         protected override void OnModelCreating(ModelBuilder modelBuilder)
14         {
15             modelBuilder.Entity<Team>().ToTable("Team");
16             modelBuilder.Entity<EmpTeam>().ToTable("EmpTeam");
17         }
18     }
19 }
```

Listing 8.6. Example of DbContext from file-destination Models.cs

[8] CRUD means Create - Read - Update - Delete
[9] The DbSet class represents an entity set that can be used for create, read, update, and delete operations

8.2 Database design

Utilising the techniques discussed in section 8.1.1 we can implement the desired database structure as shown below in figure 8.8. In this section we will go through the majority of the database diagram, describing occurrences of *many-to-many* and *one-to-many* relations in the database, as well as implementing *normalisation* in order to achieve atomic data.

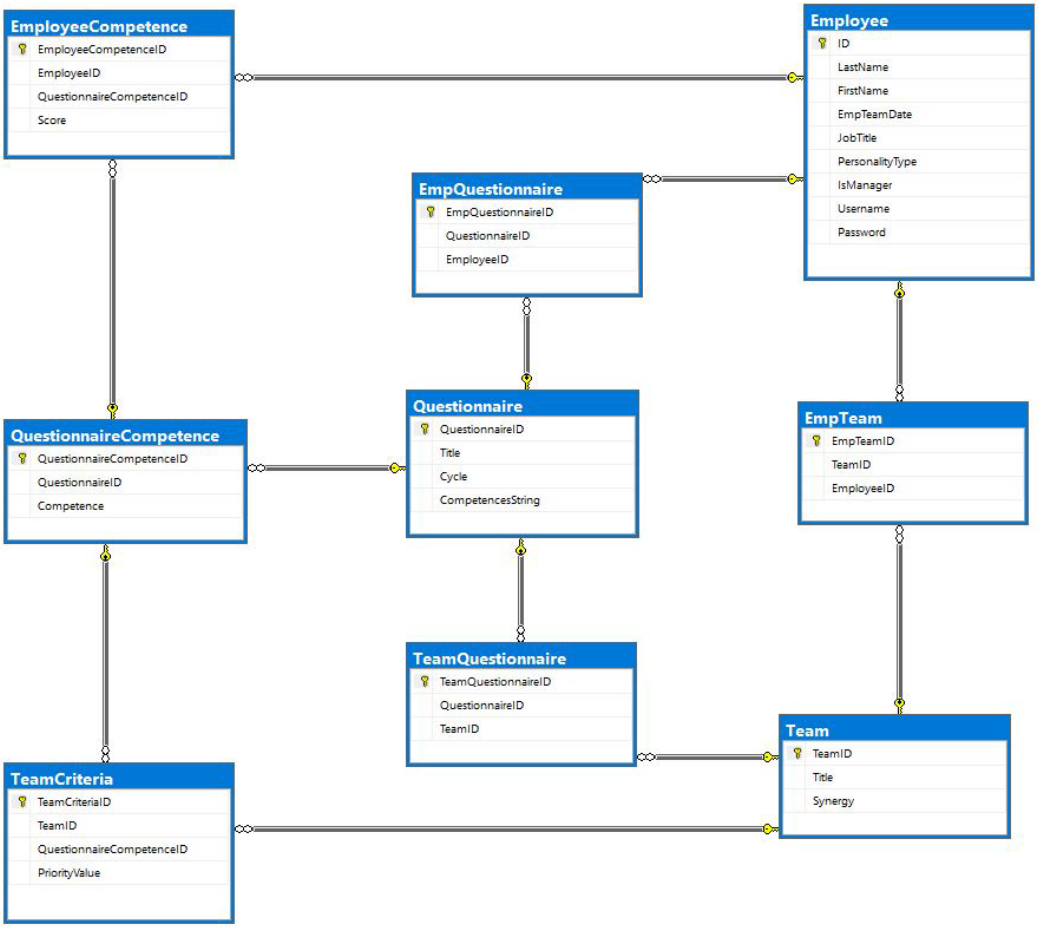

Figure 8.8. Database diagram of the application

The applications main focus is the *Employee*, *Questionnaire* and *Team* tables, and these have a many-to-many relationship with each other; For instance, one employee can be a member of multiple teams, and one team can have multiple team members.

We have normalised the data by introducing tables to store these relationships as seen in figure 8.8, which also makes the data atomic. These many-to-many tables are called *EmpTeam*, *EmpQuestionnaire*, and *TeamQuestionnaire*. This division leads to smaller table schemas, that increases the maintainability of each table. An example of a many-to-many table is the *EmpTeam* that has the attributes: *teamID*, *employeeID*, and *EmpTeamID*, where the *teamID* and *employeeID* function as foreign keys.

When an employee takes a questionnaire the data is stored in the *EmployeeCompetence* table, which has a many-to-many relationship between the employee and the associated questionnaire

competences. Likewise, the teamcriterias are stored in the *TeamCriteria* table.

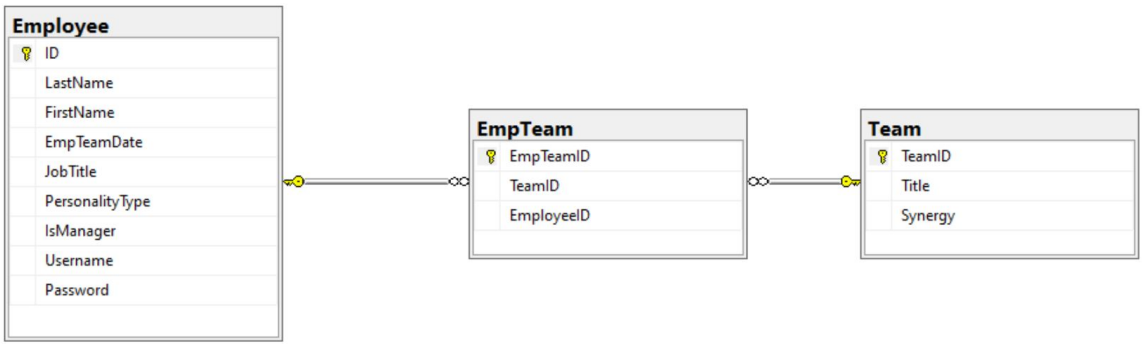

Figure 8.9. Many to many relation between employees and teams

In the database design a one-to-many relationship is used with the *QuestionnaireCompetence* table and is related to the *Questionnaire* table, see figure 8.10. Here a questionnaire can have multiple competences, but a questionnaires competence, can not have multiple questionnaires assigned.

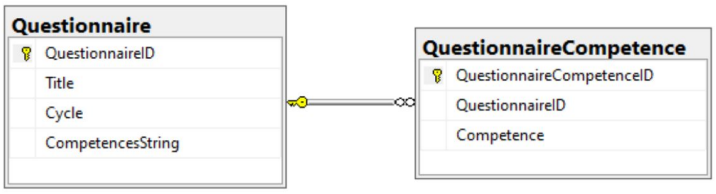

Figure 8.10. One to to many relation between questionnaire and questionnaire competences

8.3 Core functionality

After having laid the foundation for the application in the previous section, we will now describe how the core functionality is implemented in the application.

8.3.1 Login functionality

An essential part of the system is the login functionality. Firstly, this functionality makes sure that only the intended users of the system can get access, and secondly we need to make sure that the system can distinguish between a regular employee and a manager, as described in section 5.2. Only the most essential code of the login system will be explained. In the listing 8.7 the *OnPost* method for the *PageModel Login* is displayed.

```
1  public IActionResult OnPost()
2        {
3
4            // New list which contain the employee that matches username in database.
5            var usernameExist = new List<Employee>(
6                _context.Employees.Where(c => c.Username == Username));
7
8
9            // The exception makes sure the system dont crash, if no employee was found above.
10           try
11           {
12               // Gets the password in db from the entered username.
```

```
13              LoginPassword = usernameExist[0].Password;
14          }
15          catch
16          {
17              ErrorMessage = "Username or Password invalid";
18              return Page();
19          }
20
21          // The if statement checks if the entered password in page, matches the password of the found
                username in db.
22          if (LoginPassword == Password)
23          {
24
25              // checks if the user thats logs in is manager.
26              if (usernameExist[0].IsManager == true)
27              {
28                  HttpContext.Session.SetString("username", Username);
29                  return RedirectToPage("/Login/ManagerLogin");
30
31              }
32              // If the user is not manager, the user is logged in as regular employee.
33              else
34              {
35                  HttpContext.Session.SetString("username", Username);
36                  return RedirectToPage("/Employees/Dashboard");
37              }
38          }
39          else
40          // If entered password does not match password in db, the same page is returned with
                errormessage.
41          {
42              ErrorMessage= "Username or Password invalid";
43              return Page();
44          }
45      }
46  }
```

Listing 8.7. Login *PageModel* from file-destination Pages/Login/Login.cshtml.cs

The login *PageModel* retrieves the entered username and password from the form at the login page, seen in figure 8.11. Line 5-6 in listing 8.7 is the first step in validating the login shown. A list *usernameExist* is declared, which contains the employee object, where the submitted username matches a username in the database. Next is an exception declared from line 10-19. This exception makes sure that the system do not crash if no employee was found. If the list *usernameExist* does contain an employee, then the employee's password is saved in the property *LoginPassword*.

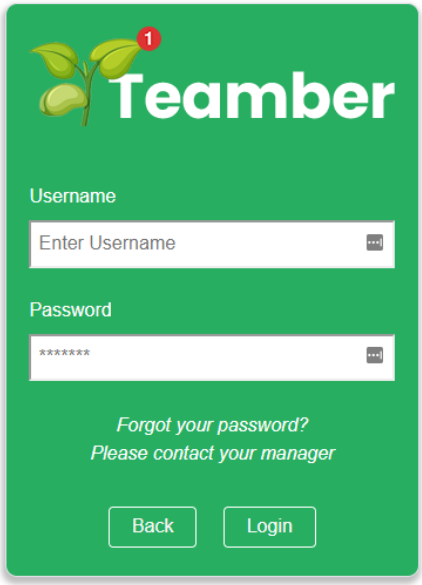

Figure 8.11. Teamber login form

After this an if statement is defined from line 22-44. The nested if statements checks if the entered password matches the password of the found employee in the database. If this is not true, then the user will be directed to the same login page with an error message. Otherwise if it is true, then a nested if statement from line 26-37 checks if the employee is a manger or a regular employee. In both cases a session variable will be declared, which saves the username belonging to the employee. If it is a regular employee, then the user is logged in. But if it is a manager, then the user will be directed to the page *ManagerLogin*. This page allow the user to select if it wants to login as manager or employee. The essential code on the *PageModel* *ManagerLogin* is seen in the listing 8.8 below.

```
1  // checks if user have selected manager or employee in form at pagemodel.
2        public IActionResult OnPost(string submitchoice)
3        {
4
5            if (submitchoice == "LoginAsManager")
6            {
7                // If user have selected manager, a addtionel manager sessions is started.
8                // The sessions contains the string "ManagerLogin".
9                HttpContext.Session.SetString("Manager", Managerlogin);
10               return RedirectToPage("/Teams/index");
11           }
12           else
13           {
14               return RedirectToPage("/Employees/Dashboard");
15           }
16
17       }
```

Listing 8.8. ManagerLogin *PageModel* from file-destination Pages/Login/ManagerLogin.cshtml.cs

The code in listing 8.8 checks if the user has selected to login in as a manager or a regular employee. If employee is selected, then the user will be logged in as an employee. But if manager is selected another session variable is declared, which contains the string *managerlogin* (see line 9 in listing 8.8). This session variable is ultimately what the system will use to distinguish

between a regular employee and a manager. On each page of the system an if statement checks if the session variables *username* and *manager* is not empty. If they are empty, access is denied, and the user will be redirected to the login page. On the pages for employees, access is only granted if session *username* is not empty, and *manager* is empty.

8.3.2 Edit employee

As mentioned in section 8.2, there are a lot of many-to-many tables in the system. When updating teams and employees, the respective objects in the *Model* have to interact with the aforementioned tables. The method for updating employees, teams, and questionnaires are very similar. To demonstrate these methods, the *edit employee* method will be used as an example. The webpage is shown in figure 8.12.

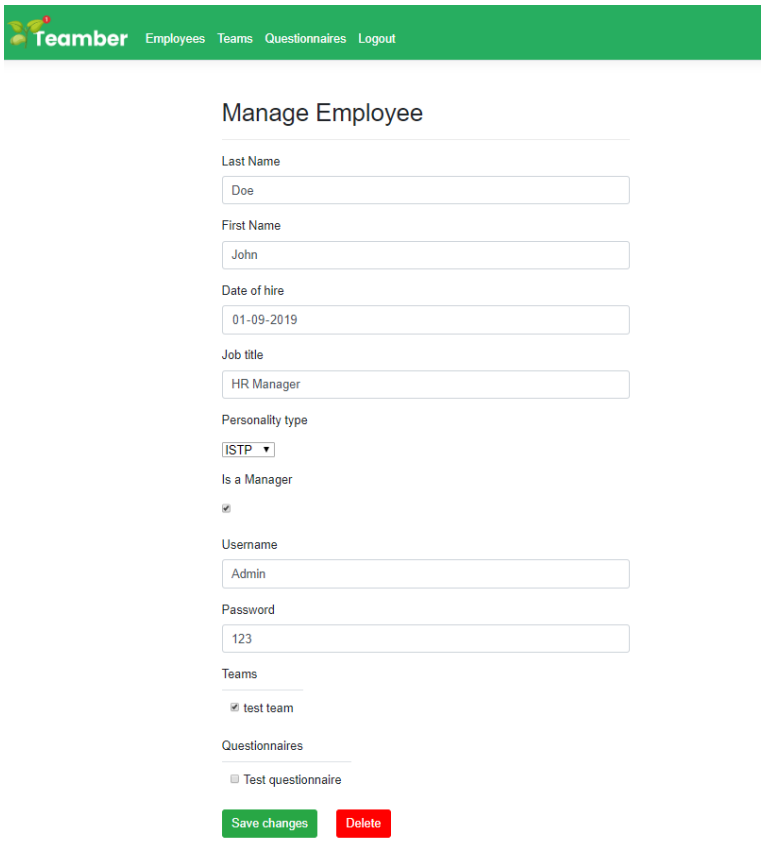

Figure 8.12. The manage employee page

The two major methods to update the employee, is the *OnGetAsync* method and the *OnPostAsync* method. The *OnGetAsync* method is used for displaying the information on the webpage and the *OnPostAsync* method is used to update the data in the system. The two methods will be explained in the next sections.

OnGetAsync method

The complete method is shown in listing 8.9.

```
1 public async Task<IActionResult> OnGetAsync(int? id)
2       {
3           Login = HttpContext.Session.GetString("username");
```

```
4           Manager = HttpContext.Session.GetString("Manager");
5
6           if (id == null)
7           {
8               return NotFound();
9           }
10
11          Employee = await _context.Employees
12              .Include(i => i.EmpTeams).ThenInclude(i => i.Team)
13              .AsNoTracking()
14              .Include(k => k.EmpQuestionnaires).ThenInclude(k => k.Questionnaire)
15              .FirstOrDefaultAsync(m => m.ID == id);
16
17          if (Employee == null)
18          {
19              return NotFound();
20          }
21
22          PopulateAssignedTeamData(_context, Employee);
23          PopulateAssignedQuestionnaireData(_context, Employee);
24          return Page();
25      }
```

Listing 8.9. *OnGetAsync* method in Edit Employee from file-destination Pages/Employees/Edit.cshtml.cs

The first part from line 3-4 in listing 8.9, gets the session data as described in section 8.3.1. The *Employee* property is then updated with the information that is needed to update the employee. This is seen on lines 11 to 15. First the many-to-many table *EmpTeams* is included because this table contains the team-members for each team. The table primarily consists of two foreign keys; a *TeamID* and a *EmployeeID*. After the *include* method the *ThenInclude* method is called. Because we want to include the team data - not just the *TeamID*. This is also done for the questionnaires, as the questionnaire data is also needed. Lastly the *FirstOrDefaultAsync* method is called, because we only want the information of the current employee.

Lambda expressions are used in these methods to shorten the code and increase the readability. An example of a lambda expression is seen in the first include statement on line 11-12:

$_{c}ontext.Employees.Include(i => i.EmpTeams).ThenInclude(i => i.Team)$ The first part, $.Include(i => i.EmpTeams)$, can be expressed verbally as “for the employee, include an iCollection of said employees related Empteam objects”. So “i” means the employee, and "EmpTeam" is the associated many-to-many object, that stores which teams the employee is a member of. In the ThenInclude function the “i” means the EmpTeam object. So for the EmpTeam object we want to include the team, which is expressed as (i => i.Team).

Lastly on lines 22 and 23 two methods are called that updates the list of the employees associated teams and questionnaires. This data is used to check the correct checkboxes as seen in figure 8.13. This enables that the user can see the correct data.

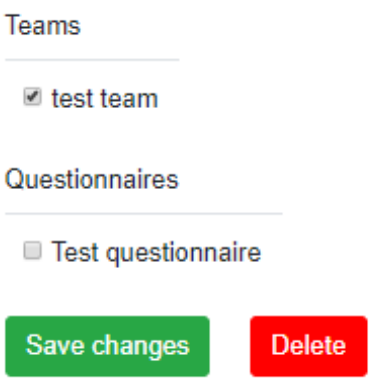

Figure 8.13. Sample checkboxes for the Manage Employee page

The two methods are very similar so only the PopulateAssignedTeamData function is described below.

PopulateAssignedTeamData method

This method populates a list of objects that are called *AssignedTeamData* which can be seen in listing 8.10.

```
namespace Teamber.Models.TeamberViewModels
{
    public class AssignedTeamData
    {
        public int TeamID { get; set; }
        public string Title { get; set; }
        public bool Assigned { get; set; }
    }
}
```

Listing 8.10. AssignedTeamData from Models/TeamberViewModels/AssignedTeamData.cs

The only data in this class is a teamID, a title for the team, and a boolean that is true when the employee is assigned to the team. When this boolean is true the checkbox is checked as seen in figure 8.13 for the *test team*.

The *PopulateAssignedTeamData* method is shown in listing 8.11.

```
public void PopulateAssignedTeamData(TeamberContext context,
                                     Employee employee)
      {
          var allTeams = context.Teams;
          var employeeTeams = new HashSet<int>(
              employee.EmpTeams.Select(c => c.TeamID));
          AssignedTeamDataList = new List<AssignedTeamData>();
          foreach (var team in allTeams)
          {
              AssignedTeamDataList.Add(new AssignedTeamData
              {
                  TeamID = team.TeamID,
                  Title = team.Title,
                  Assigned = employeeTeams.Contains(team.TeamID)
              });
          }
      }
```

Listing 8.11. PopulateAssignedTeamData from file-destination Pages/Employees/EmployeeTeamsPageModel.cs

The two parameters are a database context object and an employee object. The employee object reference includes the related team and questionnaire data as described in the beginning of this section. The overall approach we used for this method was to compare each of the teams in the database and each of the employees associated teams. If the team is related to the employee then the assigned boolean is set to true, as seen on line 14; the employeeTeams variable is a list of the employees associated teams ID's and the team variable contains a team.

The *AssignedTeamDataList* then contains all of the teams, but has the *Assigned* property set accordingly for the current employee. So that the *Assigned* boolean is set to true when the employee is a member of the team.

OnPostAsync method

The *OnPostAsync* method is used to update the current employee and is shown in listing 8.12.

```
1  public async Task<IActionResult> OnPostAsync(int? id, string[] selectedTeams, string[]
       selectedQuestionnaires)
2          {
3              if (id == null)
4              {
5                  return NotFound();
6              }
7
8              var employeeToUpdate = await _context.Employees
9                  .Include(i => i.EmpTeams)
10                     .ThenInclude(i => i.Team)
11                 .Include(k => k.EmpQuestionnaires)
12                     .ThenInclude(k => k.Questionnaire)
13                 .FirstOrDefaultAsync(s => s.ID == id);
14
15             if (employeeToUpdate == null)
16             {
17                 return NotFound();
18             }
19
20             await TryUpdateModelAsync<Employee>(
21                 employeeToUpdate,
22                 "Employee",
23                 i => i.FirstMidName, i => i.LastName,
24                 i => i.EmpTeamDate, i => i.JobTitle,
25                 i => i.PersonalityType, i => i.IsManager,
26                 i => i.Username, i => i.Password);
27             {
28                 UpdateEmployeeTeams(_context, selectedTeams, employeeToUpdate);
29                 UpdateEmployeeQuestionnaires(_context, selectedQuestionnaires, employeeToUpdate);
30                 await _context.SaveChangesAsync();
31                 return RedirectToPage("./Index");
32             }
33         }
```

Listing 8.12. OnPostAsync function from file-destination Pages/Employees/Edit.cshtml.cs

The content of the method is very similar to the *OnGetAsync* method. One of the major differences is that the *TryUpdateModelAsync* method is called (line 20-26 in listing 8.12). This method updates the correct row in the *Employee* table. The data should be updated for all of the columns in the table except the ID column. This is shown in the lambda expressions, e.g. $i => i.FirstMidName$ on line 23.

After this the *UpdateEmployeeTeams* method and the *UpdateEmployeeQuestionnaires* method

are called on lines 28-29. This is done in separate methods as the data that needs to be updated is not stored in the *Employee* table but instead in the *EmpTeam* and *EmpQuestionnaire* tables. Both methods are very similar and therefore only the first will be described below.

UpdateEmployeeTeams method

The method is shown in listing 8.13.

```
1  public void UpdateEmployeeTeams(TeamberContext context,
2            string[] selectedTeams, Employee employeeToUpdate)
3        {
4            if (selectedTeams == null)
5            {
6                employeeToUpdate.EmpTeams = new List<EmpTeam>();
7                return;
8            }
9
10           var selectedTeamsHS = new HashSet<string>(selectedTeams);
11           var employeeTeams = new HashSet<int>
12               (employeeToUpdate.EmpTeams.Select(c => c.Team.TeamID));
13           foreach (var team in context.Teams)
14           {
15               if (selectedTeamsHS.Contains(team.TeamID.ToString()))
16               {
17                   if (!employeeTeams.Contains(team.TeamID))
18                   {
19                       employeeToUpdate.EmpTeams.Add(
20                           new EmpTeam
21                           {
22                               EmployeeID = employeeToUpdate.ID,
23                               TeamID = team.TeamID
24                           });
25                   }
26               }
27               else
28               {
29                   if (employeeTeams.Contains(team.TeamID))
30                   {
31                       EmpTeam teamToRemove
32                           = employeeToUpdate
33                               .EmpTeams
34                               .SingleOrDefault(i => i.TeamID == team.TeamID);
35                       context.Remove(teamToRemove);
36                   }
37               }
38           }
39       }
```

Listing 8.13. UpdateEmployeeTeams from Pages/Employees/EmployeeTeamsPageModel.cs

This method is very similar to the *PopulateAssignedteamsData* method described in section 8.3.2. One of the notable differences is that this method takes three parameters whereas the other function only took two. The extra parameter is the *string[] selectedTeams* shown in line 2. This parameter is used to contain the team ID's of the checked teams at the edit employee page as illustrated in figure 8.14.

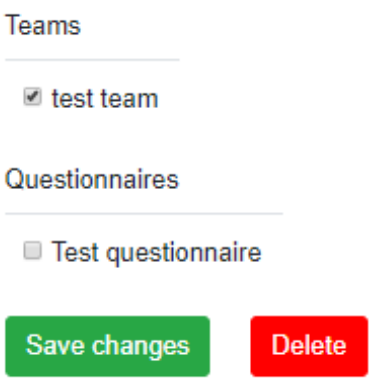

Figure 8.14. Sample checkboxes for the Manage Employee page

The primary approach for the method is to compare all of the teams in the database with the selected teams. If the team is also in the checked array, then the employee belongs to the team. If the employee was not in the team prior to the edit, then a new row is added to the *EmpTeams* table containing this data, as seen from lines 17 to 25 in listing 8.13.

If the team is not in the selected teams array, then we check if the employee currently belongs to the team. If that is the case, then the employee should be removed from the team, as the team has been unchecked on the edit employee page. This is seen on line 29 - 36.

8.3.3 Take Questionnaire

The take questionnaire page lets an employee select their personality type, as well as rating their own competences, that corresponds to the competences that are in the questionnaires, which are assigned to the employee by the manager (see figure 8.15).

The personality type is retrieved by taking a personality test on the external website `www.16personalities.com`, that later gives managers an overview of the distribution of personality types on the team page, (see figure 9.1).

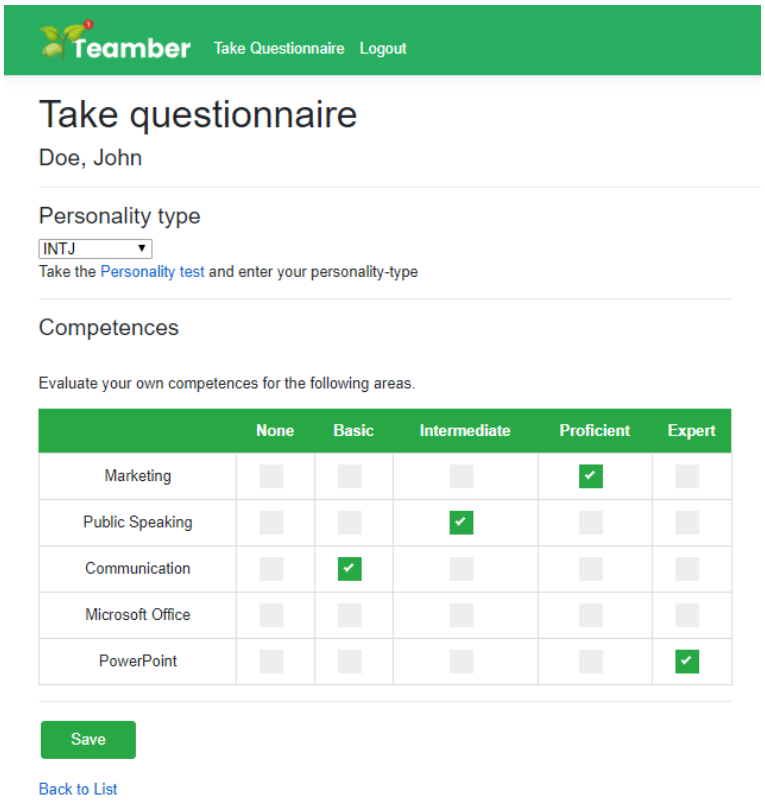

Figure 8.15. Take questionnaire page

The following code excerpts will be taken from *TakeQuestionnaire.cshtml* and *TakeQuestionnaire.cshtml.cs*, to give an insight into the functionalities associated with employees taking questionnaires, in the application.

The first thing seen on the page is the page title and the users name, as displayed on listing 8.14. The users name is retrieved directly from the Employee *Data Model* using two-way data binding, and displays the name of the user who is currently logged in. This is maintained by the applications login functionality, as explained further in section 8.3.1. The title on the page is inserted as hard-coded text.

```
1 <h1>Take questionnaire</h1>
2 <h4>@Html.DisplayFor(model => model.Employee.FullName)</h4>
```

Listing 8.14. Title and Username from file-destination Pages/Employees/TakeQuestionnaire.cshtml

The form on the page is a regular HTML-form, which is displayed in listing 8.15. The form incorporates custom validation for checkbox inputs. The custom checkbox validation is executed on the forms *onsubmit* attribute as seen on line 1. The functionality of the custom validation will be expanded further upon near listing 8.18. The form furthermore uses built in Razor Page validation, as seen on line 2 for the input and it also has a hidden input field that stores the Employees ID on line 3. This gives us a reference to the employee taking the questionnaire, and thereby knows what specific users values to update.

```
1 <form method="post" onsubmit=" return ValidateCheckboxes()">
2         <div asp-validation-summary="ModelOnly" class="text-danger"></div>
```

```
3           <input type="hidden" asp-for="Employee.ID" />
```

Listing 8.15. HTML-form from file-destination Pages/Employees/TakeQuestionnaire.cshtml

Within the form on the page, a *form-group* that can be seen on line 2 in listing 8.16. This code excerpt provides the user with a drop-down select option, as seen on line 3-13. This showcases the options of personality types, which is hard-coded as an array, that is shown on line 5. Initially *Select a type* is shown to the user, as seen on line 4. The options showcased to the user are displayed based on the foreach loop on line 7-11, that iterates for each of the hard-coded personality types. Once the user clicks on a personality type from the options, this value is sent to the database upon submitting the form. This works as the *PageModel* searches for the value attribute as seen on line 9, that defines what personality type to add to the user. If the user does not select a personality type, the form will not be submitted, as the entire input field is set as *required* which is displayed on line 3.

```
1   <h4>Personality type</h4>
2           <div class="form-group">
3               <select name="Employee.PersonalityType" required>
4                   <option value="">Select a type</option>
5                   @{ string[] personalityTypes = { "ISTJ", "INFJ", "INTJ", "ENFJ", "ISTP", "ESFJ",
                        "INFP", "ESFP", "ENFP", "ESTP", "ESTJ", "ENTJ", "INTP", "ISFJ", "ENTP", "ISFP" };
6                   int cnt1 = 0;
7                   foreach (var personality in personalityTypes)
8                   {
9                   <option value="@(Html.Raw(personality))"
                        name="Employee.PersonalityType">@(Html.Raw(personality))</option>
10                  cnt1++;
11                  }
12              }
13          </select>
14          </div>
```

Listing 8.16. Personality types from file-destination Pages/Employees/TakeQuestionnaire.cshtml

With the previously hidden input described on line 3 in listing 8.15 which gets the user ID in our *PageModel*, we can assign the selected personality. This is stored as a *value* in the previous listing to the *Employee.PersonalityType* attribute in the *Data Model*. This is because we referenced it in the *name* attribute on line 9. In listing 8.17 seen below, we take the aforementioned reference's to the page and send them to the *PageModel* as seen in listing 8.17 on line 1-4. The *TryUpdateModelAsync* method awaits user inputs in the HTML-form, and then on submission, sends the data to the *Data Model*.

```
1   await TryUpdateModelAsync<Employee>(
2               employeeToUpdate,
3               "Employee",
4               i => i.PersonalityType);
5
6   await _context.SaveChangesAsync();
7   return RedirectToPage("./Dashboard");
```

Listing 8.17. OnPostAsync Update from file-destination Pages/Employees/TakeQuestionnaire.cshtml.cs

Competences

The competence questionnaire which is shown in listing 8.18 consists of a *foreach loop* that iterates over the competences that are assigned to the user (line 18-44), based on the questionnaires assigned to them by a manager. The rating of competences are seen on line

9-13, which in the system are stored as the numbers 1-5, when submitted and processed in the system.

Traditionally HTML radio buttons would be used for selecting one value in a range. In our application we use checkboxes, as the entire *form-group* which the ratings exist in, is considered one context when using loops. This means that it would only be possible to check one radio button in the entire loop that we iterate over, if we were to use radio buttons.

Based on this, we use checkboxes with custom functionality, utilising jQuery, as seen on line 38-40. This provides us with the same functionality as radio buttons.

```
1
2  <h4>Competences</h4>
3            <br />
4            <div class="form-group">
5                <table id="competenceRating">
6                    <!-- Rating values-->
7                    <tr>
8                        <th></th>
9                        <th>None</th>
10                       <th>Basic</th>
11                       <th>Intermediate</th>
12                       <th>Proficient</th>
13                       <th>Expert</th>
14                   </tr>
15
16                   <!-- Itterator that gives each checkbox group a unique class, so the javascript can
                          refference it and only make one checkbox clickable in each group-->
17                   @{ int cnt = 1;
18                       foreach (var competence in Model.AssignedEmployeeCompetenceDataList)
19                       {
20
21                       // Hidden refference to assign elements correctly
22                       <input type="hidden" name="selectedCompetences"
                              value="@competence.QuestionnaireCompetenceID" />
23                       <tr>
24
25                           <!--Component name-->
26                           <td>@competence.Criteria</td>
27                           <!-- five checkboxes for each component -->
28                           @for (int i = 1; i < 6; i++)
29                           {
30                           <td>
31                               <label class="containercheck">
32                                   <input type="checkbox" class="option@(Html.Raw(cnt))"
                                          value="@(Html.Raw(i))" name="selectedCompetencesValue" required>
33                                   <span class="checkmark"></span>
34                               </label>
35                           </td>
36
37                           <!--Only make it possible to check one checkbox in each row -->
38                           <script type="text/javascript"> $('.option@(Html.Raw(cnt))').on('change',
                                  function () {
39                                                $('.option@(Html.Raw(cnt))').not(this).prop('checked',
                                                    false);
40                                            });
41                           </script>
42                           }
43                       </tr>
44                       cnt++;
45                       }
46                   }
47               </table>
```

```
48          </div>
```

Listing 8.18. Checkboxes from file-destination Pages/Employees/TakeQuestionnaire.cshtml.cs

The code displayed in listing 8.18 gives us a hidden input on line 22-23 that stores an ID for each competence in our *Model*, where the competences are assigned to a questionnaire that the employee is also assigned too. This data is stored in *Model.AssignedEmployeeCompetenceDataList*. This gives us a reference to what competence to assign the competence rating to later. The for loop on line 28-40 creates five checkboxes with a value from 1-5 assigned to them. This corresponds to the rating scale displayed in the beginning of the file at line 7-14.

As previously stated we modified the functionality to make the checkboxes behave as radio buttons. Listing 8.19 showcases an excerpt from line 38-41 from the previous listing, that handles this functionality.

```
1   <!--Only make it possible to check one checkbox in each row -->
2 <script type="text/javascript"> $('.option@(Html.Raw(cnt))').on('change', function () {
3       $('.option@(Html.Raw(cnt))').not(this).prop('checked', false);
4 });
5 </script>
```

Listing 8.19. Checkbox change from file-destination Pages/Employees/TakeQuestionnaire.cshtml

The script above iterates over all checkboxes on the page, when something in the form is changed by the user. This is described by the *.on('change')* notation before the function is defined on line 2.

Each competence-row has a unique class name defined by the *cnt* attribute, that increments each time that the initial foreach loop has run, which starts on line 18 on listing 8.18. The nested for loop then creates five checkboxes for each component as seen on line 28.

When a checkbox value is set to *checked*, the script seen on listing 8.19 on line 3 runs. This script translates to: set the property *checked* to *false* for all checkboxes in the current class (*cnt*), that is not the currently clicked element (*.not(this)*)

Finally we have the *ValidateCheckBoxes()* function as seen on listing 8.20 that runs when the form is submitted. This script validates if all competences have been given a rating by the user.

```
1   <form method="post" onsubmit=" return ValidateCheckboxes()">
```

Listing 8.20. Form submit from file-destination Pages/Employees/TakeQuestionnaire.cshtml.cs

The script shown in listing 8.21 loops over all inputs shown on line 7. While running the loop the *checkBoxCount* variable initialised on line 5, counts how many checkboxes are checked. This is done by incrementing the variables values by one, if an element on the page is a checkbox and its property is set to *checked*. This translates into the if-statement seen on line 8. The HTML-form will submit its data as long as the *checkboxCount* is not less than the amount of competences assigned to to the user in the *Model*, as described in the if statement on line 12-14. If the condition is true, it will return false and display an error message, as shown on line 13-14.

```
1   <!--Because of the unorthodox checkbox setup, we need a dedicated Validation method for the
        checkboxes-->
2           <script>
3               function ValidateCheckboxes() {
4                   var inputElems = document.getElementsByTagName("input"),
5                       checkBoxCount = 0;
```

```
6
7                 for (var i = 0; i < inputElems.length; i++) {
8                     if (inputElems[i].type == "checkbox" && inputElems[i].checked == true) {
9                         checkBoxCount++;
10                    }
11                }
12                if (checkBoxCount < @Model.AssignedEmployeeCompetenceDataList.Count()) {
13                    alert("Looks like you forgot to fill out some parts of the form");
14                    return false;
15                }
16            }
17        </script>
```

Listing 8.21. Validation of checkboxes from file-destination Pages/Employees/TakeQuestionnaire.cshtml

If both a personality type and a rating is selected for each competence, the data is now registered in the system.

8.3.4 Calculating the competence synergy

The primary function of the website is the algorithm for calculating the competence synergy between the employee-competences and the team-criteria. This function is conceptually described in section 5.2, whereas this section will describe the actual implementation of it. The function is used on the create and manage team pages, and is seen in figure 8.16.

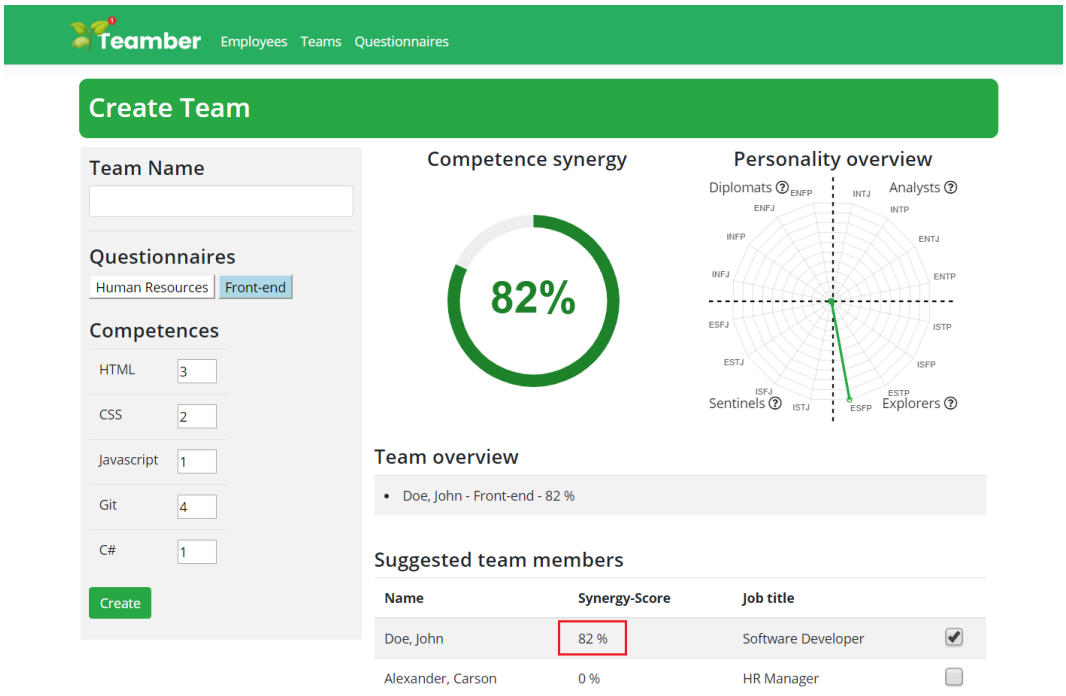

Figure 8.16. The create team page with the employee synergy highlighted

The function is implemented using Javascript, as the data needs to be updated on the client side dynamically based on the user's input. The whole function is seen in listing 8.22.

```
1 function calculateEmployeeSynergy(employee, questionnaire, max_score) {
2
```

```
3         var memberInChosenQuestionnaire = false;
4
5         try {
6             var memberQuestionnaireTitle = document.getElementById("" + employee +
                  "-memberQuestionnaire").innerText;
7             var currentQuestionnaire = document.getElementById(activeQuestionnaireButton).innerText;
8
9             if (memberQuestionnaireTitle == currentQuestionnaire) {
10                memberInChosenQuestionnaire = true;
11            }
12        }
13        catch (err) {
14
15        }
16
17        if (document.getElementById("" + employee + "-teamMember") == null || memberInChosenQuestionnaire)
               {
18            var totalScore = 0;
19            var target = QuestionnaireCriterias[questionnaire]; //target contains the criterias for the
                   active questionnaire in a dictionary
20
21            for (var competence in target) {
22                if (target.hasOwnProperty(competence)) {
23                    var competenceInt = parseInt(competence);
24                    var tempScore = EmpCompetences[employee][competenceInt]; //EmpCompetences contains
                           dictionaries of employees and their competences
25                    var competenceScore = parseInt(tempScore);
26
27                    if (competenceScore > target[competenceInt]) {
28                        competenceScore = parseInt(target[competenceInt]);
29                    }
30
31                    totalScore += competenceScore;
32                }
33            }
34
35            var tempSynergy = totalScore / max_score * 100;
36            var synergy = Math.round(tempSynergy);
37
38            if (Number.isInteger(synergy)) {
39                document.getElementById("" + employee).innerHTML = synergy;
40            }
41            else {
42                document.getElementById("" + employee).innerHTML = 0;
43            }
44        }
45        else if (memberInChosenQuestionnaire == false) {
46
47        }
48    }
```

Listing 8.22. Calculating competence synergy function

The function takes three parameters: an employee ID, a questionnaire ID, and a max_score as seen on line 1. The questionnaire ID is the selected questionnaire that determines which criteria the employee synergy should be calculated from. The employee ID is a single employee. In order to calculate all of the employee synergies, this function runs for each employee. The max_score is the max total synergy score as mentioned in section 5.2.

The first part of the function checks whether the employee is associated with the selected questionnaire, and is seen in listing 8.23.

```
1         var memberInChosenQuestionnaire = false;
```

```
2
3       try {
4           var memberQuestionnaireTitle = document.getElementById("" + employee +
                "-memberQuestionnaire").innerText;
5           var currentQuestionnaire = document.getElementById(activeQuestionnaireButton).innerText;
6
7           if (memberQuestionnaireTitle == currentQuestionnaire) {
8               memberInChosenQuestionnaire = true;
9           }
10      }
11      catch (err) {
12
13      }
```

Listing 8.23. Function which checks assosiation with selected questionnaire. Line 3-15 in listing 8.22

If the current questionnaire is called *front end developers*, then the front end developer employees will get a synergy score calculated. The rest of the employees, which are not included in the front end developers questionnaire, will get a zero score (if they are not already in the team). The code in listing 8.23 checks if an employee is added to the team as for example a front-end developer. If that is true then the employee's synergy should be calculated, see line 7-8. The reason being that the team members, that are not associated with the current questionnaire, should keep their current synergy score - it should not be recalculated.

Figure 8.17 shows what this would look like on the actual website with the associated questionnaire highlighted.

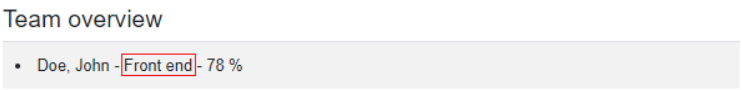

Figure 8.17. Illustration of calculated employee synergy

The next part is a big if statement with the condition seen in listing 8.24 line 1. The first part checks if the current employee is not a member of the team. But if the employee is part of the team, then it checks to see if the employee is added for the corresponding questionnaire. If any of this is true, then the next code in listing 8.24 is executed.

The next part of the code is the actual algorithm and is seen in figure 8.24. This part is already described in pseudo-code in section 5.2.

```
1       if (document.getElementById("" + employee + "-teamMember") == null || memberInChosenQuestionnaire)
            {
2           var totalScore = 0;
3           var target = QuestionnaireCriterias[questionnaire]; //target contains the criterias for the
                active questionnaire in a dictionary
4
5           for (var competence in target) {
6               if (target.hasOwnProperty(competence)) {
7                   var competenceInt = parseInt(competence);
8                   var tempScore = EmpCompetences[employee][competenceInt]; //EmpCompetences contains
                        dictionaries of employees and their competences
9                   var competenceScore = parseInt(tempScore);
10
11                  if (competenceScore > target[competenceInt]) {
12                      competenceScore = parseInt(target[competenceInt]);
13                  }
14
```

```
15                  totalScore += competenceScore;
16              }
17          }
18
19          var tempSynergy = totalScore / max_score * 100;
20          var synergy = Math.round(tempSynergy);
```

Listing 8.24. The part that calculates the competence synergy. Line 17-36 in listing 8.22

First, the current criteria are stored as a dictionary in the target-variable (listing 8.24 line 3). Then we loop through each of these criteria and get the employees score for this criteria (listing 8.24 line 5-17). For example, if C# competences are highly prioritised in the current team, it may have a numeric value of 5. Then we see what the employee scores on C# and add that score to totalScore (listing 8.24 line 15). This is done for all of the competence criteria for the questionnaire. In the second if statement we check whether the employees competences are higher than the competence criteria (listing 8.24 line 11-12). If this is true, then we change the employees competence score to the criteria value.

After the for loop we have a totalScore for the employee. However the synergy needs to be shown in percent. This can be seen in listing 8.24 line 19 and 20.

The result of this is either a number from 0 to 100 or Not-A-Number (NaN). When the function is run initially, there are no criteria given, and therefore the max score is 0. This gives a divide-by-zero error and results in Not-A-Number. When the result is not a number, the synergy score displays a zero. The code for this is shown in listing 8.25.

```
1       if (Number.isInteger(synergy)) {
2           document.getElementById("" + employee).innerHTML = synergy;
3       }
4       else {
5           document.getElementById("" + employee).innerHTML = 0;
6       }
7   }
8   else if (memberInChosenQuestionnaire == false) {
9
10  }
```

Listing 8.25. The part of the method that inserts the synergy on the page. Line 38-47 in the overall listing 8.22

The innerHTML[10] that is changed is shown in figure 8.18 and contains the competence score.

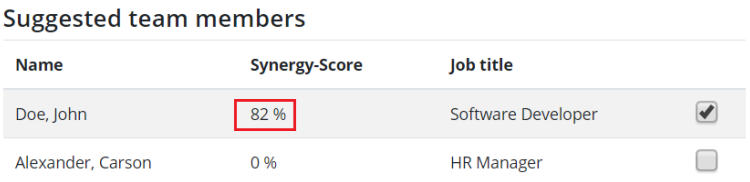

Figure 8.18. Illustration of employees competence score

Related functions

The prior described function calculates the competence synergy for a single employee. This function is run through another function called *calculateSynergies*. The primary goal of this

[10]Example: the innerHTML of Hello is Hello.

function is to call the prior function for each of the employees. And lastly sort the employees by their current synergy score so that the employees with the highest synergy score is shown first.

Another core function is called *calculateTeamSynergy*. This function simply calculates the average synergy of the chosen team members. For example if a team has 3 members: Person A, B and C. And their respectfully synergy score is 80 %, 60 % and 90 %, then the team synergy score is (80 + 60 + 90) / 3 = 77 %.

System evaluation 9

In this chapter we will look into the requirements and needs that the application must fulfill based on the work done in the problem- and application domain. We will utilise the FACTOR criterion (see section 3.2.2), to determine if the system reflects the requirements gathered. We will also look at the classes from the problem-domain in section 4 and functions listed in the application domain in section 5. Furthermore we will see if the design criteria listed in the architectural design-phase in section 7 is fulfilled and corresponds to the finished system.

9.1 Testing methodology

The testing of the system will be based on Glenn Myers definition hereof: "*Testing is the process of executing a program or system with the intent of finding errors*" (Rick D. Craig, 2002). Furthermore we will take an iterative approach to the testing efforts. This is to catch issues earlier on, in order to prevent a snowball effect, where previous errors, could result in more errors, or needing to implement workarounds that could later cause further errors and less code-maintainability in the system.

As the current state of the software is an alpha release[1], as it is not yet aimed towards hitting production, our testing-efforts will be aimed towards internal testing of the functionality of the application. Which in future work should be followed up by some external testing, utilising black-box techniques[2] or white-box techniques[3].

The application will be tested utilising Jakob Nielsens approach to heuristic evaluation. This is defined as: *"A systematic inspection of a user interface design for usability."* (Nielsen, 1993, p. 155). This is done by having an expert evaluator test a system, based on a predefined list of heuristics, which marks areas of the application to test. In this case we are utilising summative evaluation[4], as this allows testing of our previously defined system requirements, as a measurement test, where our requirements will work as heuristics (Nielsen, 1993, p. 170).

The success of the system test will be determined by the fulfillment of the requirements listed in the FACTOR criterion, functions described, and our design criteria. This results in the table shown in 9.1.

[1] A release where the features which are developed are partially complete.

[2] Testing method that asses the overall experience of a finished product, not including internal structures or workings (Farmeena Khan, 2012)

[3] Testing method that tests internal structures or workings, typically by using unit- or integration testing (Farmeena Khan, 2012)

[4] An evaluation that is done to asses the overall experience of a finished product

9.2 Evaluation results

Table 9.1 shows the requirements and if they are fulfilled in the system.

ID	Requirement	Fulfilled
User		
1	**Employee**	
1.1	Take questionnaire	✓
1.2	Read teams	✓
1.3	Read general employee information	✓
2	**Manager**	
2.1	Read employees	✓
2.2	Register employee	✓
2.3	Read employee information	✓
2.4	Update employee information	✓
2.5	Delete employee	✓
2.6	Read teams	✓
2.7	Create team	✓
2.8	Read team information	✓
2.9	Update team information	✓
2.10	Delete team	✓
2.11	Read questionnaires	✓
2.12	Create questionnaire	✓
2.13	Read questionnaire information	✓
2.14	Update questionnaire information	✓
2.15	Delete questionnaire	✓
2.16	Send questionnaire reminder (e-mail)	×
System		
3.1	Validate login	✓
3.2	Calculate team competence synergy	✓
3.3	View suggested team members for each team	✓
3.4	Sort by members most fitting to the team	✓
3.5	Display distribution of personalities in the team	✓

Table 9.1. Evaluation of system requirements

Table 9.1 is categorised by the actor who has access to the functions. The *system* category is added to the table. If a functional requirement is not initialised by a user, but initialised by the system, it is in this category. Each requirement and whether or not it is fulfilled, will be explained below.

ID 1.1 Take Questionnaire

The take questionnaire functionality is what enables the employee to take a questionnaire and submit it, so managers can use the employees competences and personality type to create efficient teams based on a set of competence criteria. This functionality has been fulfilled with little to no issues raised in the development process. A walk-through of the function is also provided in section 8.3.3.

ID 1.2-1.3 Employee dashboard

ID 1.2-1.3 makes up for additional functionality belonging to the employee, which is limited to viewing what teams the employee is a part of, as well as displaying the general information. This functionality is quite simple, as it mainly consists of reading data, and was therefore easily fulfilled.

ID 2.1-2.15 Manager CRUD-operations

The majority of functionality in the system is based on the managers input, therefore many CRUD operations are accessible to the manager. In the Employee, Questionnaire and Team page, the manager can get an overview of all elements belonging to the page, creating individual elements, such as employees in the employee page, and furthermore view, edit or delete data. Razor Pages supports functionality called scaffolding that lets one create such CRUD-operations on the basis of a *Model* (Microsoft, 2019). The core challenges was related to custom functionality on top of these basic operations, which will be expanded further upon in section 9.4, but ultimately these requirements were also fulfilled.

ID 2.16 Send Questionnaire Reminder

This requirement was not fulfilled. It is not possible for the manager to send out a reminder to take a questionnaire to the employees related to said questionnaire. The system should have had a time cycle function, where the questionnaire would automatically have been sent from the system to the employees that are assigned to the questionnaire. This was not possible as the system had to be hosted online, before being able to use cron-jobs[5], which would let the manager maintain and send e-mails. The online hosting was not possible due to unexpected challenges regarding deployment to the host.

ID 3.1 Validate login

The login functionality validates the login information by checking if the user exists in the system, and the information in the system matches the information provided. It also checks if the user is an employee or a manager. If the requirement is fulfilled, the user is granted access to the system dependent on their role. The function is described in section 5.2 and the implementation in section 8.3.1.

ID 3.2-3.5 Creating teams

These requirements help visualise the team composition process for the manager. When a manager wants to compose a team, they are shown the page which is displayed in figure 9.1. In box 1 the manager sets the criteria for the team. Here it is possible for them to choose which competences the team prioritises. In box 2 the suggested employees are sorted by their synergy score. This fulfills requirement 3.3 and 3.4. In box 3.1 the employee's calculated synergy is shown, and the team competence synergy in shown in box 3.2. This fulfills requirement 3.2. The display of the distributed personality types in the team is shown in box 4. This fulfills requirement 3.5.

[5]Program that runs on the server at planned intervals.

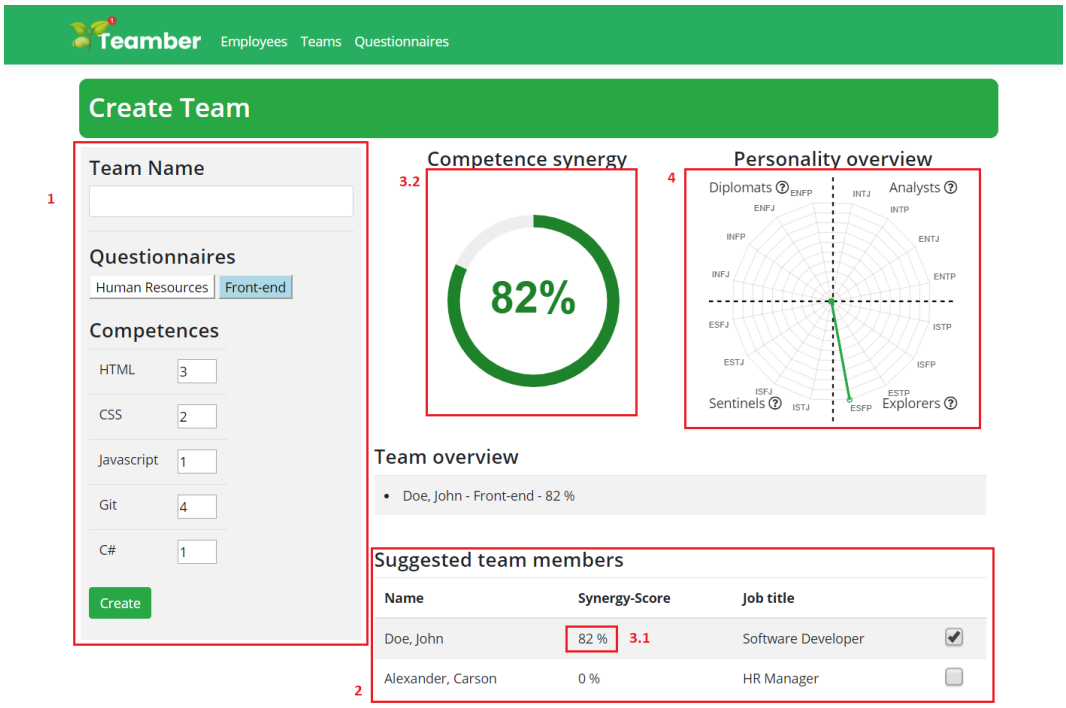

Figure 9.1. Create team page

9.3 Deprioritised functionality

In our wireframes seen in section 6.2.3 the initial idea was to have a *search feature*, that would let the manager search between employees, teams as well as questionnaires. This feature was later deemed less desirable, as it would not add much value compared to the time it would take to implement. Therefore the search feature has been set to a possible future implementation. Likewise, there was ideas of having *pagination*[6] to easier navigate between pages, but as this system is considered to be an alpha release, which seeks to build the core functionality. Navigation between multiple pages of the same content is not a must-have at the current moment.

9.4 Error identification

In relation to the efforts to iteratively identify errors related to the requirements collected in table 9.1. In this section we will expand further upon an excerpt of issues that surfaced towards the later testing-stages, in the development process. These errors might seem trivial, the reason being that the iterative testing efforts, quickly identified bigger errors related to the core functionality of the system, which was dealt with much earlier in the process. This section primarily exists to display the train of thought of troubleshooting and removing errors in the system.

[6]Dividing a page into discrete pages, which can be navigated in by using page numbers.

Crashes on wrong username/password

Later in the development, it was observed that in relation to the login-function with the ID 3.1, that some rather critical errors surfaced. This happened upon entering a wrong username, which would result in an out of index error supplied by our development environment as seen in figure 9.2.

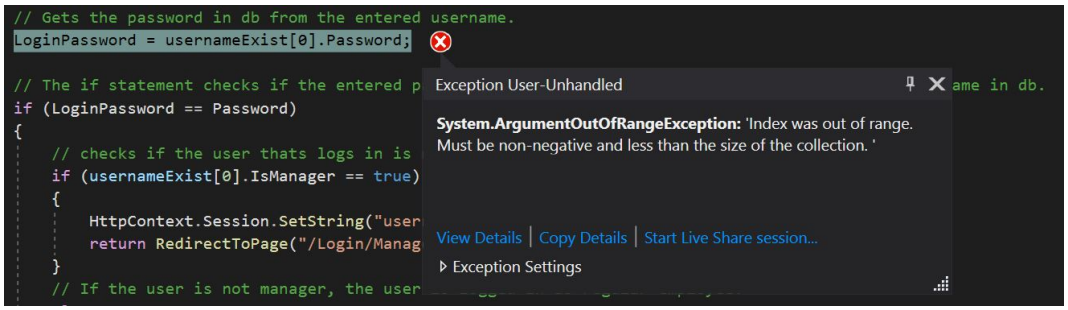

Figure 9.2. Error upon wrong username or password

This event additionally lead to the user encountering an error page in the browser, which would be catastrophic in production. Even though the error is critical, the solution is rather simple by enclosing the statement in a try/catch statement as seen in listing 9.1 and handle bad user inputs. This is done by refreshing the application and showing an error message, as shown on line 8-9, rather than causing the application to crash.

```
1           try
2           {
3               // Gets the password in db from the entered username.
4               LoginPassword = usernameExist[0].Password;
5           }
6           catch
7           {
8               ErrorMessage = "Username or Password invalid";
9               return Page();
10          }
```

Listing 9.1. Try/catch statement from file-destination Pages/Login/Login.cshtml.cs

Personality types are set to blank if they are not changed on edit

In a later iterations of testing, it turned out that the personality type of an employee was reset, if it was not changed when editing an employee. The previously asserted personality type was shown, but once the employee information was changed without changing that specific option it was reset. An occurrence of this can be seen in figure 9.3.

Figure 9.3. Personality type is reset when the employee is edited, without changing their personality type

Upon exploration of the code, which can be seen in listing 9.2, it was hard to tell what went wrong in the beginning, as no error message indicated an error.

```
1  <div class="form-group">
2                          <p>Personality type</p>
3                          <select name="Employee.PersonalityType">
4                              <option value="">@Model.Employee.PersonalityType</option>
5                              @{ string[] personalityTypes = { "ISTJ", "INFJ", "INTJ", "ENFJ", "ISTP",
                                   "ESFJ", "INFP", "ESFP", "ENFP", "ESTP", "ESTJ", "ENTJ", "INTP",
                                   "ISFJ", "ENTP", "ISFP" };
6                                  foreach (var personality in personalityTypes)
7                                  {
8                                      <option value="@(Html.Raw(personality))"
                                           name="Employee.PersonalityType">@(Html.Raw(personality))</option>
9                                  }
10                             }
11                         </select>
12                     </div>
```

Listing 9.2. Personality type options from file-destination Pages/Employees/Edit.cshtml.cs

Upon closer inspection it was noticed, that while the previous assigned personality type was shown in the browser, it would not be saved upon form submission. This was because the value of the *option* tag, which did not save anything as seen on line 4. Thus the fix was as simple as entering the previous assigned personality type to the value attribute of the option tag. as seen in listing 9.3.

```
1  <option value="@Model.Employee.PersonalityType">@Model.Employee.PersonalityType</option>
```

Listing 9.3. Option value from file-destination Pages/Employees/Edit.cshtml.cs

9.5 Evaluation summary

With our previously described success criteria of the system being the fulfilment of the entries in table 9.1, we can conclude that the system is considered a success. The majority of the system requirements is fulfilled, with the only outlier being the ability to send reminders to employees, when they should take a questionnaire. This was primarily caused by unforeseen hurdles, because online hosting was needed for this feature to be implemented, which was not possible because of unforeseen problems.

Even though the system implementation is considered successful, it should be noted that the testing only extends to the core functionality. It is tested via heuristic evaluation by the developers. The system is functional, these results does not however reflect how usable it is, additional users should be included in testing. This is though included in the future work chapter, as the system is an alpha release, not ready for production before additional testing.

Future Work 10

With the implementation and evaluation of the system completed, we will reflect upon the potential future work. The perspective of the future work is in regards to a potential implementation of the system in an organisation.

10.1 Usability

As mentioned in section 7.1.2 the focus has been on the functionality of the system, which is why usability testing has been deprioritised. If an organisation wants to adopt our system in the future, usability would be a high priority. The reason being that it is important that the potential users have an optimal experience. If the system experience is not optimal the system's value to the users will decrease. Subsequently a contract with a potential organisation is very likely to demand that the system is highly usable. Not fulfilling their requirements can be costly.

A way to ensure that our system is highly usable, would be to conduct a usability test. With a representative set of users and tasks of the primary use of the system, it would be possible to determine if the system is usable as demanded by a potential organisation.

The implementation of the *questionnaire cycle* method could improve the system's usefulness. This function would automatically send a questionnaire reminder via email to the employees after a given time period. This would reduce the managers workload. Before this function can be implemented, the system needs to be hosted on a server. The email reminder wont work, if the system is hosted locally, which is the case as of now. Getting the system to run on a server, is one of the first potential tasks in the future work.

Additionally, a search function would also be beneficial. When managing employees in the system, it would be more convenient and faster to search for an employee, instead of scrolling through a large list.

10.2 Security

Security is an important topic in the event of the system being implemented in an organisation. But as mentioned in section 7.1.2, the security has been deprioritised, because the focus has been on developing the core functionality. If the system is to be used in an organisation, security features such as encryption of the database should be implemented. It is important that the person's sensitive data is protected. With the non-encrypted database, user information (e.g. password) can be read and exploited if unauthorised access is gained to the database. Besides that a comprehensive test, of the systems overall security, could be conducted.

10.3 More attributes

The current algorithm for calculating the employee synergy is rather simplistic; it only calculates the competence synergy. However, this algorithm could be made more comprehensive by adding additional attributes to account for. These attributes could include gender, age, distance, years in company, salary, education and number of currently assigned teams. This could make the team compositions more diverse, stable, and cost-efficient. The core challenge when applying more attributes in the algorithm is that companies have different priorities. For example, one company might prioritise stability, whereas another might prioritise cost-efficiency. This could be accounted for by adding more functionality to the create- and edit team pages. This functionality could be a prioritisation-scale from 1-5 for *diversity*, *stability* and *cost-efficiency*, where the algorithm then would take these ratings into account.

10.4 Tracking employee development

As of now the application only gives the manager a snapshot of the employees' current competences. This makes for a static viewpoint of something that is dynamic in nature. To further aid the manager in picking better teams, it would be beneficial to track the employees' competences over time. This gives a reflection of how well the employee perform over time. The addition of tracking the employees progress, would also make the system more valuable to employees using the application, as it can aid them in gaining a better insight into their personal development. This feature could be implemented by having a separate table that stores previous questionnaire answers and then on the front-end of the application provide a graph, which visualise the different data points.

Conclusion 11

In our problem analysis we came to the decision that we wanted to build an IT-system that supports the team composition process based on the employees' competences and personality types. This led to the creation of our system definition, where the system functionalities were stated. A core functionality that the system should include was to calculate the team competence synergy based on the employees competence match. This should be based on a functionality that enables the employees to take a questionnaire, where they rate their competences.

In the pursuit of creating the system, thorough analysis of the problem domain, application domain, and architectural design was conducted. For example the following classes were identified: employee, manager, questionnaire and team. This process laid a foundation for the database structure and the implementation of the core functionality. This made it easier to understand which tables were needed in the database, and how they were related. Furthermore it made the implementation of classes and functions in the system easier and more coherent with the system's requirements.

Overall, it can be concluded that the implementation resulted in a system that corresponds with the system definition, and the stated functionality requirements. This means that a manager is able to create employees, questionnaires and teams, and create teams based on the competence synergy score and personality types. However, as specified in the evaluation and future work the only unfulfilled requirement relates to sending email reminders. We were not able to fulfill this requirement because of unforeseen complications in deploying the system to an online host. Finally, it can be concluded that if the system is to be implemented in an organisation, more examination and testing is needed in regards to usability and security.

Bibliography

Benyon, **2014**. David R. Benyon. *Designing Interactive Systems*. ISBN: 978-1-292-01384-8, PDF. Pearson, 2014.

Fairfield, **2012**. Kent D. Fairfield. *Cases and Excercises in Organization Development change*. ISBN: 9781483387444. SAGE Publications, 2012.

Farmeena Khan, **2012**. Mohd Ehmer Farmeena Khan. *A Comparative Study of White Box, Black Box and Grey Box Testing Techniques*. 2012. URL `https://www.researchgate.net/publication/270554162_A_Comparative_Study_of_White_Box_Black_Box_and_Grey_Box_Testing_Techniques`.

Fowler, **2002**. Martin Fowler. *Patterns of Enterprise Application Architecture - Page Controller*. 2002. URL `https://martinfowler.com/eaaCatalog/pageController.html`.

Halfhill et al., **2003**. Terry Halfhill, Joseph W. Huff, Eric Sundstrom og Tjai M. Nielsen. *GROUP PERSONALITY COMPOSITION AND WORK TEAM EFFECTIVENESS: KEY FACTOR IN STAFFING THE TEAM-BASED ORGANIZATION?* Elsevier Science Ltd., 9, 147–167, 2003. URL `https://www-emerald-com.zorac.aub.aau.dk/insight/content/doi/10.1016/S1572-0977(02)09009-X/full/pdf?title=group-personality-composition-and-work-team-effectiveness-key-factor-in-staffing-the-tean`

Lars Mathiassen, **2000**. Peter Axel Nielsen Jan Stage Lars Mathiassen, Andreas Munk-Madsen. *Object oriented analysis design*. ISBN: 9788777511530. Metodica ApS, 2000.

Microsoft, **2017**. Microsoft. *Simpler ASP.NET MVC Apps with Razor Pages*. 2017. URL `https://docs.microsoft.com/en-us/archive/msdn-magazine/2017/september/asp-net-core-simpler-asp-net-mvc-apps-with-razor-pages`.

Microsoft, **2019**. Microsoft. *Razor Pages with Entity Framework Core in ASP.NET Core*. 2019. URL `https://docs.microsoft.com/en-us/aspnet/core/data/ef-rp/intro?view=aspnetcore-3.1&tabs=visual-studio`.

NERIS Analytics Limited, **a**. NERIS Analytics Limited. *16personalities*. URL `https://www.16personalities.com`. Lokaliseret: 13/5-20.

NERIS Analytics Limited, **b**. NERIS Analytics Limited. *Our framework*. URL `https://www.16personalities.com/articles/our-theory`. Lokaliseret: 13/5-20.

Nielsen, **1993**. Jakob Nielsen. *Usability Engineering*. ISBN: 978-0-12-518406-9, New edition. Elsevier Science and teknologi, 1993.

Oke et al., **2016**. A. E. Oke, S. O. Olatunji, A. O. Awodele, J. A. Akinola og M. Kuma-Agbenyo. *IMPORTANCE OF TEAM ROLES COMPOSITION TO SUCCESS OF CONSTRUCTION PROJECTS*. International Journal of Construction Project

Management, 8(2), 141–152, 2016. URL https://search-proquest-com.zorac.aub.aau.dk/docview/1864049796/fulltextPDF/844C0A7803D54A1DPQ/1?accountid=8144.

Rick D. Craig, **2002**. Stefan P. Jaskiel Rick D. Craig. *Systematic Software Testing*. ISBN: 1580535089. Artech House Publishers, STQUE publishing, 2002.

Robert R. McCrae, **1989**. Jr. Robert R. McCrae, Paul T. Costa. *Reinterpreting the Myers-Briggs Type Indicator From the Perspective of the Five-Factor Model of Personality*. Journal of Personality., 57, 17–40, 1989. URL https://kbdk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/proquest57684333.

Robert R. McCrae, **2008**. Paul T. Costa Jr. Robert R. McCrae. *Empirical and Theoretical Status of the Five-Factor Model of Personality Traits*. ISBN: 9781849200462. SAGE Publications, 2008.

Saborio, **2017**. Will Saborio. *Today's HR tech landscape*. 2017. URL https://medium.com/s/story/no-please-help-yourself-981058f3b7cf.

Savelsbergh et al., **2012**. Chantal Savelsbergh, Josette M. P. Gevers, Beatrice I. J. M. van der Heijden og Rob F. Poell. *Team Role Stress: Relationships With Team Learning and Performance in Project Teams*. Sage, 37(1), 67–100, 2012. URL https://journals-sagepub-com.zorac.aub.aau.dk/doi/pdf/10.1177/1059601111431977.

Sinclair, **2019**. Marshall Sinclair. *Why the Self-Help Industry Is Dominating the U.S.* 2019. URL https://medium.com/s/story/no-please-help-yourself-981058f3b7cf.