

Employee Training Management System

A system to improve the management of employee training and automate the communication of notices, regarding overdue and soon to be overdue training from employees.



Korab Beqiraj

**EMA: Table of Contents**

[**1 Project Report** 3](#_Toc69756155)

[**1.1 Title** 3](#_Toc69756156)

[**1.2 Problem Description** 3](#_Toc69756157)

[**1.2.1 Nature and Context of the Problem** 3](#_Toc69756158)

[**Existing knowledge** 4](#_Toc69756159)

[**1.2.2 Proposed solution** 4](#_Toc69756160)

[**Overview** 4](#_Toc69756161)

[**Project output** 4](#_Toc69756162)

[**Key ICT aspects** 5](#_Toc69756163)

[**1.2.3 Analysis of Likely Impact** 5](#_Toc69756164)

[**1.4 Project Work** 6](#_Toc69756165)

[**1.4.1 Analysis** 6](#_Toc69756166)

[**1.4.1.1 Use Cases** 6](#_Toc69756167)

[**1.4.1.2 Use Case Diagram** 6](#_Toc69756168)

[**1.4.1.3 Requirements** 7](#_Toc69756169)

[**1.4.1.4 System Scope and boundaries** 9](#_Toc69756170)

[**1.4.1.5 Architecture** 10](#_Toc69756171)

[**1.4.2 Synthesis** 11](#_Toc69756172)

[**1.4.2.1 Core System (Store)** 11](#_Toc69756173)

[**1.4.2.2 Database and RESTful API** 12](#_Toc69756174)

[**1.4.3 Evaluation** 14](#_Toc69756175)

[**1.4.3.1 Testing Use Cases** 14](#_Toc69756176)

[**1.4.3.2 Test data** 14](#_Toc69756177)

[**1.4.4 Legal, social, ethical and professional issues** 19](#_Toc69756178)

[**1.4.4.1 Legal Issues: Data Protection Act 2018 (DPA 2018)** 19](#_Toc69756179)

[**1.4.4.2 Possible Professional and Ethical Issues** 20](#_Toc69756180)

[**1.4.4.3 Social and Ethical Issues** 20](#_Toc69756181)

[**1.4.4.4 Personal Professional Conduct** 20](#_Toc69756182)

[**2 Review** 21](#_Toc69756183)

[**2.1 Current state of project** 21](#_Toc69756184)

[**2.1.2 Skills and Resources Required to Finish the Project.** 22](#_Toc69756185)

[**2.2 Project Management Review** 23](#_Toc69756186)

[**2.2.1 Project Management Lifecycle** 23](#_Toc69756187)

[**2.2.2 Project Plan** 23](#_Toc69756188)

[**3 Appendix** 27](#_Toc69756189)

[**Appendix A: Supplementary evidence:** 27](#_Toc69756190)

[**A1: Use Case Descriptions** 27](#_Toc69756191)

[**A2: Core System (Store) Implementation model** 34](#_Toc69756192)

[**Class Diagram** 34](#_Toc69756193)

[**Class descriptions** 34](#_Toc69756194)

[**Invariants** 47](#_Toc69756195)

[**A3: RESTful API Documentation** 48](#_Toc69756196)

[**A4: Use Case Test Cases** 53](#_Toc69756197)

[**5 References** 75](#_Toc69756198)

# **1 Project Report**

## **1.1 Title**

**Employee Training Management System**

## **1.2 Problem Description**

### **1.2.1 Nature and Context of the Problem**

##### **Context**

A mobile phone sales company specialising in Smart Phones and Tablets, known as Phones2U operates in the United Kingdom. The sales employees are required to advise customers on current mobile devices and technologies. For the employees to advise customers, they must be appropriately trained by the company. Therefore, the company requires staff to complete training and as an incentive the company offers the employees a pay rise and commission on sales after training has been completed. The training requires staff to pass initial tests within a set time frame. There are also annual re-sits of tests to make sure that employees are up to the company standard. If employees miss any deadlines, their commission earnings are suspended until employees take and pass the required test/s.

##### **Training**

Consists of compulsory and optional tests that start as soon as employees are hired. Before any of the tests can be attempted, a training event needs to be booked and attended by the employee. The following table shows the compulsory tests employees need to pass and the deadline for each test.

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Deadline (relative to start date) | | Reward |
| Junior Sales Assistant | | 3 months | Pay increase |
| Sales Assistant | | 6 months | Pay increase |
| Qualified Sales Assistant\* | | 12 months | Earn commission on sales |
| Senior Sales Assistant | | 18 months | Pay increase |

**Table: Compulsory tests for employees.**

**Note**: The qualified sales assistant test is the only test which requires an annual resit.

##### **Current Problems**

1. Currently staff are reminded of training deadlines with a letter sent to their respective store. The problem with this is that the letters are often overlooked, misplaced or forgotten by employees, this sometimes leads to employees missing a deadline for a test.
2. Store managers do not have direct access to employee training progress and must keep track of employee training manually in store using printouts and other information stored physically in folders.
3. Booking employees on training events is done over the phone, which can sometimes take time away from an already busy schedule of a store manager, especially if the required member of the training team is not available at the time of the call.
4. The training department stores staff training records on spreadsheets, which is easy to back up, but very difficult to ensure consistency, in a large and growing company. Furthermore, tasks such as adding, removing and updating employees/training events can become difficult using the limited organisational capabilities of a spreadsheet.

This reliance on storing information on physical files and sending letters via post can cause many problems, as mistakes, like misplacing a letter can cause problems for staff, e.g. losing commission, that may affect staff moral and impact performance and quality of service provided to the customer.

#### **Existing knowledge**

I have had first person experience of this problem as a member of staff going through the training phases, and eventually as a manager in charge of training staff at a store of a similar, but not the same company, so a solution is based on experience as well as elicitation. The elicitation is mainly done via family, friends that work in fields that requires ongoing training and my own experience. I have attempted interviews with some of the for mentioned people so that I can gain a better understanding of which features will be most helpful for the Employee Training Management System.

### **1.2.2 Proposed solution**

#### **Overview**

The proposed solution includes a system that can track staff training and notify staff members and their respective managers via email and text message of training deadlines. The system will repeatedly send messages reminding staff of training deadlines, for example, if the deadline is within 4 weeks, a message is sent every other day until the member of staff is booked on a training event.

The system will also keep track of training electronically, which will allow training staff to post training events, managers to book employees on those training events and allow manager to view their employees training status from the computer stationed in the manager’s office.

#### **Project output**

In order to deliver a solution for the proposed problem I believe that a system consisting of 4 sub-systems is the best choice, the following are the subsystems:

1. **Core system (Store):** A Java based program deployed in stores that allows a manager to book their employees on training events and view their employee’s training progress.
2. **Core system (Training Department):** Like the core system deployed in stores, but with more features and privileges, for example, adding employees and training events and viewing training status of all employees in the company.
3. **Database system:** Deployed using an IaaS provider to host the,
   1. database (SQL relational database) and
   2. a RESTful API (PHP) which will provide services from the database to the clients (Core system, Communication system).
4. **Communication system**: A Java based system that will:
   1. Send text messages using a third-party service that can be accessed via the internet through HTTPs.
   2. Send e-mails using an email sever.

#### **Key ICT aspects**

The main ICT aspects for this project will be

* **communication** between all the systems based on different technologies,
* providing the appropriate **authorisation and authentication** to the sub-systems so that only authorised employees can successfully carry out certain tasks and
* **security**, so that only the intended employees have access to the system.
  + **Encryption,** so that data can be protected during transmission from/to the core system to/from the database and from the database to the communication system.
* **data storage,** an efficient, secure, consistent and reliable way to store data about employees and training.

### **1.2.3 Analysis of Likely Impact**

The proposed solution should help improve customer satisfaction and staff moral and in turn help to boost business due to better performance. This is because the current method of dealing with training causes a problem for employees and the company, as the employee’s earnings are considerably lower without the commission, which is suspended as a result of not completing training on time; The new system can help to mitigate this with reminders sent directly to the employees. This means that the company can keep its promise of a well-trained sales team to the customers and deliver the best possible service, thanks, in part to the level of training achieved and maintained by employees.

Furthermore, the system would reduce the amount of time required by training staff and store managers to handle training related tasks, which would give them extra time to focus on other aspects of the business. For example, store managers would have more time to focus on driving sales and improving customer satisfaction. Training staff would have more time to focus on improving the training provided, like, improving the training material or the support given to employees for training. Since the system would enable this extra time, the for mentioned improvements could be realise at minimal or no extra cost, as the existing staff could handle those tasks.

## **1.4 Project Work**

### **1.4.1 Analysis**

The first part of the analysis is derived from the elicitation stage, which resulted in the use cases and requirements as an outcome of the activity. This part of the report starts with the use cases and moves onto the functional and non-functional requirements that I think should be included in the final version of the system. Then the second part concentrates on the system boundaries, explaining which part of the system the use cases should be implemented in, with a brief explanation of why this specific architecture was chosen.

#### **1.4.1.1 Use Cases**

The following is a list of all the use cases I have identified for the system, including a use case diagram

**List of Use cases**

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use Case ID | Use Case | Sub-system |
| 1 | UC020 | Add Employee | Core |
| 2 | UC040 | Remove Employee | Core |
| 3 | UC060 | Edit Employee | Core |
| 4 | UC080 | View Employees | Core |
| 5 | UC100 | Book Training Event | Core |
| 6 | UC200 | Cancel Training Event | Core |
| 7 | UC300 | View Training Progress | Core and Communication |
| 8 | UC400 | Send Training Reminder | Communication |
| 9 | UC500 | Update Training Record | Core\* |
| 10 | UC600 | Add Training Event | Core\* |
| 11 | UC700 | Remove Training Event | Core\* |
| 12 | UC800 | Get Training Events | Core |

\* Only the Training Department version of the core system will implement these use cases.

**Note:** The database system has been omitted from the sub-system section in the table and use case diagram, as the database is responsible for managing all the data in the system, so will directly or in-directly be involved in every use case.

[**[Full list of Use case description: Appendix A1]**](#_A1:_Use_Case)

#### **1.4.1.2 Use Case Diagram**

The following image shows the use case model, illustrating how the use cases and actors interact with each other.

****

**Image: Use case diagram for the Training Management System.**

#### **1.4.1.3 Requirements**

##### **Functional Requirements**

|  |  |
| --- | --- |
| No. | Functional Requirements |
| FR1 | The Core System shall display a list of all employees for a given store.  **Fit criterion:**  The list of employees shall be identical to the actual employees in the physical store. |
| FR2 | The Core System shall display a list of all training events for a given date range.  **Fit criterion:**  The list of training events corresponding to the date range are displayed. |
| FR3 | The system shall order training events by date, time or location.  **Fit criterion**  The displayed events are sorted by any of the following criteria: date, time or location. |
| FR4 | The Core system shall book an employee on a training event.  **Fit criterion:**  The employee is booked on the training event. |
| FR5 | The Core system shall cancel a training event for a given employee.  **Fit criterion:**  The employee is removed from the training event. |
| FR6 | The system shall display the training progress of an employee, including, completed training and overdue training.  **Fit criterion:**  The training progress displayed shall be identical to the employees actual training progress. |
| FR7 | The Communication system shall automatically check the training progress of all employees twice a day, once at 9 am and again at 5pm.  **Fit criterion:**  Training is checked and reminders are sent to any overdue employees. |
| FR8 | The Communication system shall create a list of all employees with their name, mobile phone number and email address, for all employees that are overdue or have an upcoming training deadline within a month.  **Fit criterion:**  The generated list shall contain all employees that are overdue or have an upcoming deadline within a month. |
| FR9 | The Communication System shall send a message to remind staff of overdue or upcoming training deadlines.  **Fit criterion:**  Staff with overdue or upcoming deadlines receive a text message informing them of the training deadline. |
| FR10 | The System shall allow training staff to update the training record of an employee.  **Fit criterion:**  The training record of an employee is updated. |
| FR11 | The system shall allow a training staff member to add training events.  **Fit criterion**  A training event is added to the database. |
| FR12 | The system shall allow a training staff member to remove a training event.  **Fit criterion**  The training event is removed from the database. |

##### **Non-functional requirements**

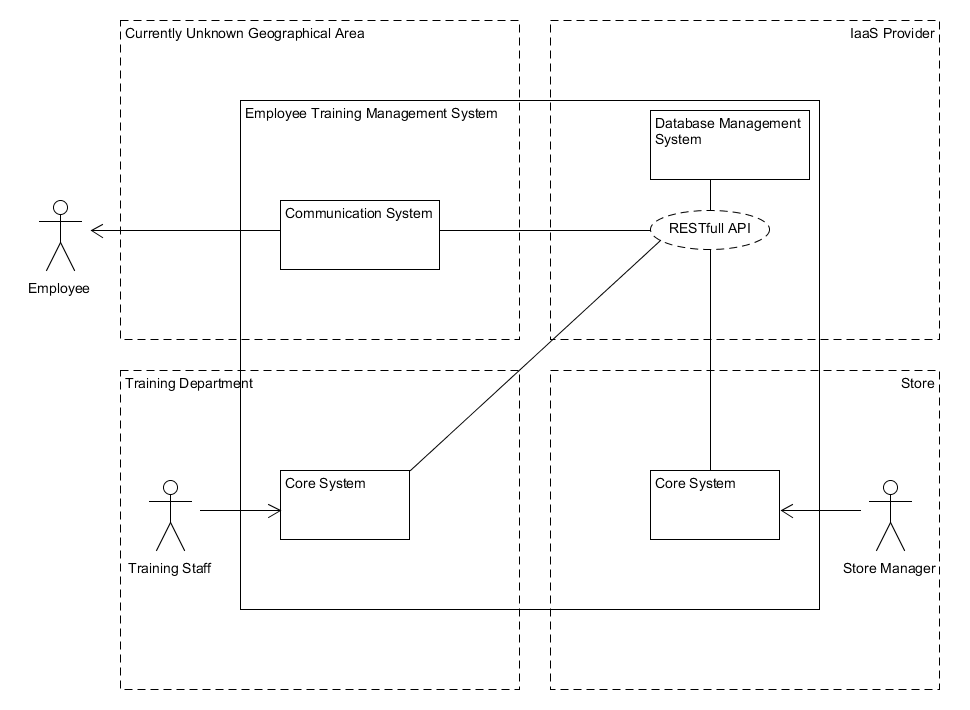
|  |  |
| --- | --- |
| No. | Non-functional requirements |
| NFR1 | The System shall provide consistent data to everyone accessing the data regardless of which store or training department requests the data.  **Fit criterion:**  In 99 % of cases the data accessed shall be equal to the same data accessed by another user from a different location. |
| NFR2 | The Database System shall have access controls in the form of authorisation and authentication.  **Fit criterion:**  Only stores and training department can access the Database System through the RESTful API. |
| NFR3 | Inter-system communication shall be encrypted to protect data during transmission from one sub-system to another.  **Fit criterion:**  In all cases data being sent from one sub system to another is unintelligible without the encryption key. |
| NFR4 | The database system shall have reliability features to minimise the potential loss of data.  **Fit criterion:**  The Database System shall have at least one complete backup of the database in a different geographical location and at least on other complete backup locally, which is isolated from the system. |
| NFR5 | The Core System shall be easy to use.  **Fit criterion:**  A manager or training department employee shall be able to learn how to use the Core System within 3 hours using instructions in text form. |
| NFR6 | The System shall be able to run on existing hardware of the company.  **Fit criterion:**  The system shall run at an acceptable speed and reliability on a computer that has a minimum of 4GB ram, 128GB Hard drive space and a 1.4ghz CPU. |
| NFR7 | The system shall run on all major operating systems.  **Fit criterion:**  The system shall be operational on Windows, Mac OS and Linux Ubuntu. |

#### **1.4.1.4 System Scope and boundaries**

The next part of the analysis stage was to decide how the system should be divided into subsystems, while considering which tasks each sub-system should be responsible for. This section continues and expands on section [**1.2.1 Nature and Context of the Problem**](#_1.2.1_Nature_and) and [**1.2.2 Proposed solution**](#_1.2.2_Proposed_solution), as these sections are aimed at breaking down the problem into smaller parts in order to produce a solution that can also be broken down into smaller sub-systems. The following diagram shows how the system is broken down into the four sub-systems:

1. Core System (Training Department),
2. Core System (Store),
3. Database System and
4. Communication System.

With the solid lined box representing the system/systems and the dashed line boxes representing an example of the real-world physical location of the sub-systems.



**Diagram: High-level abstract view of the Training Management System.**

The following table provides an example of the locations.

|  |  |  |
| --- | --- | --- |
| Sub-system | Business location | Geographical location example |
| Core | Physical Store | London, Fulham SW11 1AA |
| Core | Training Department | Guildford, GU11 1ER |
| Database | IaaS solutions like Amazon AWS or Microsoft Azure. | Manchester, City, MA11 1XX |
| Communication | Currently Unknown | Currently unknown |

**Table: Example of real-world locations for the sub-systems.**

#### **1.4.1.5 Architecture**

The Architecture chosen for the project is based on the client server architecture, this is because the clients (core system and communication system) request a service via the RESTful API from the database and receive a response from the server (database and RESTful API). This architecture was chosen as it allows for the system to be broken down into sub-systems that can use the same data source to ensure consistency. Another benefit to this architecture is that any sub-system can be swapped for another one based on a different technology, for example, the core system can be written in python instead of java and no changes would be required to any other sub-systems. Another more complicated scenario could also be possible like changing from a Relational database to a document database like mongoDB, which would require data to be imported to the new database in the new format and the RESTful API would have to be altered, but as long as the API responds to the requests via the same URLs then the other sub-systems (Core and communication systems) do not need to be altered.

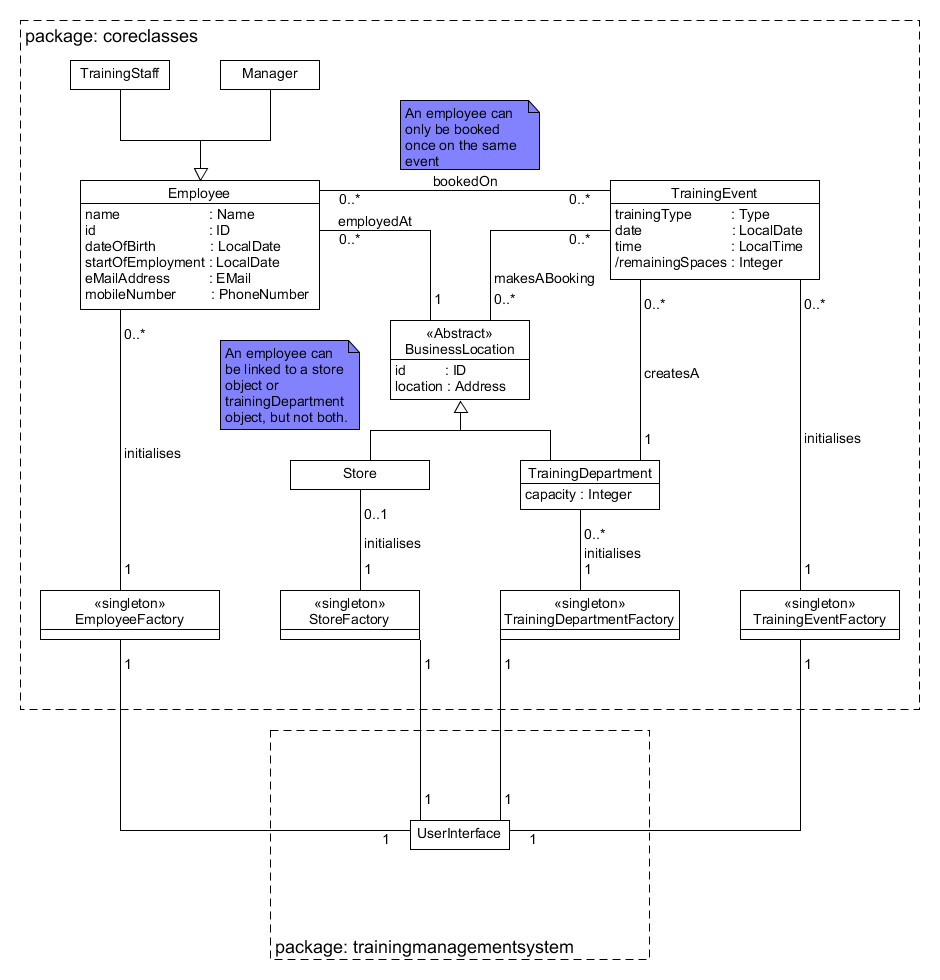
An alternative architecture was considered; Data-centered architecture.At that point I was planning the system using this kind of architecture, mainly because of the ‘blackboard’ variant of this architecture, which meant that the datacenter is responsible for notifying the clients of any changes, e.g. as soon as an employee becomes overdue than the database would be responsible for notifying the client (communication system) which would then notify the employee. I discarded this method, for an implementation that relies on the client to query the database at set intervals. This means that I can control when the communication queries the database, which will help in improving overall performance, as the communication system can query the database after business hours when the traffic to the database is at its lowest.

### **1.4.2 Synthesis**

This part of the report focuses on what work I have completed in order to find an appropriate solution to the problem as stated in [1.2 Problem Description](#_1.2_Problem_Description). I will start by giving a more detailed description of each sub-system and explain exactly what the purpose of each sub-system is and how they will achieve the tasks as outlined in the previous analysis section.

#### **1.4.2.1 Core System (Store)**

The core system is still not a finished product and requires a few more iterations to reflect the user’s expectations as stated in the sections [1.2 Problem Description,](#_1.2_Problem_Description) [1.4.1.3 Requirements](#_1.4.1.3_Requirements) and [1.4.1.1 Use Cases](#_1.4.1.1_Use_Cases). Currently the core system has been developed for the store and not the training department, so the bulk of the use cases are possible. The following is the Class diagram for the Core system.



**Image: Core system (Store) class diagram.**

[**[Please see a full list of class description in Appendix A2]**](#_A2:_Core_System)

#### **1.4.2.2 Database and RESTful API**

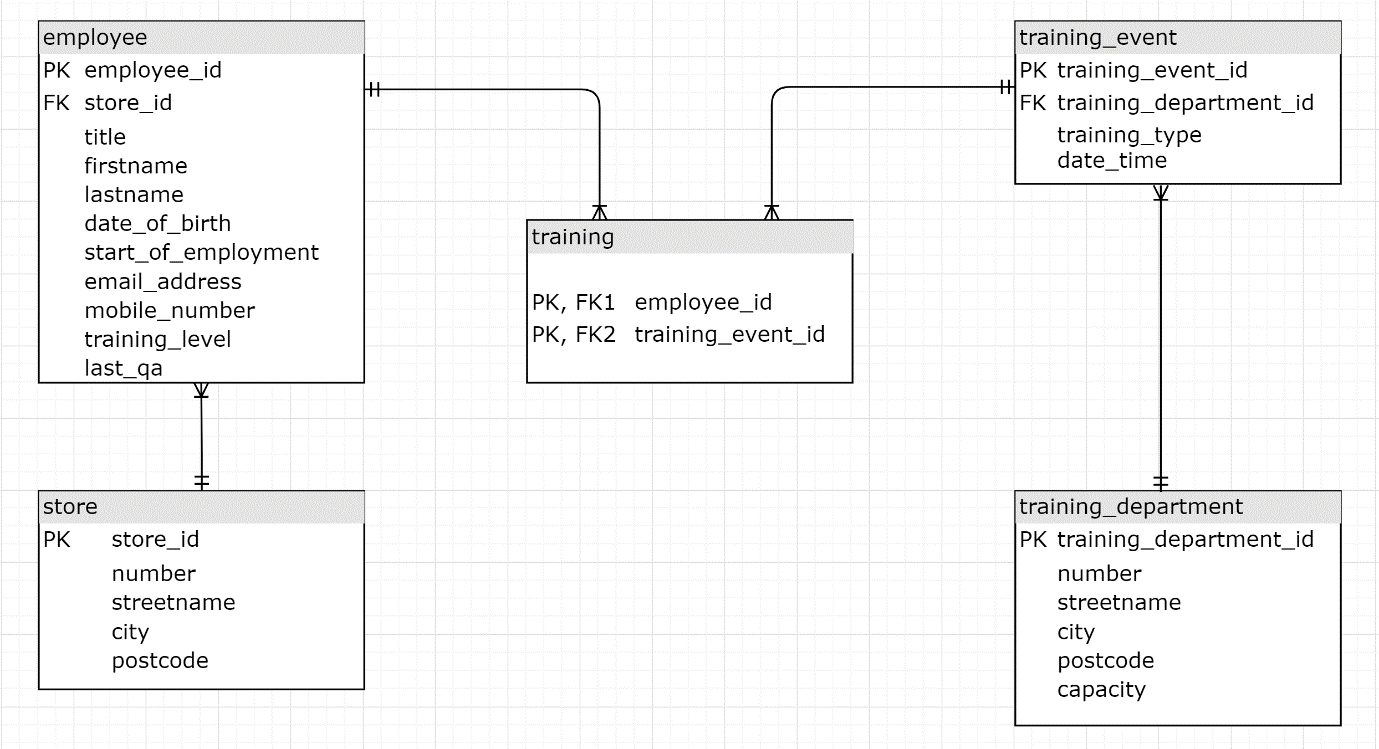
The database system is a relational SQL database known as MariaDB and uses 5 tables to store information about employees and training, and another table to store user passwords. I have chosen to add a password table so that all the clients (core systems), can have unique log in credentials so that the security of the system is improved. Creating users can also add the possibility of future updates that can allow users to log in from other devices to retrieve information from the system, such as a manager using their smartphone to retrieve training progress for their employees.

In order to accommodate password checks and server-side invariant checks I have chosen to communicate with the database through a RESTful API. The following are the key ideas behind the design of the database system.

1. Security: Users password are sent to the database system via HTTPS (encrypted). The password is then hashed at the server-side and a unique salt is added to every user in order to protect against attacks that use look-up tables or attacks that analyze data in transmission (Edmunds, 2016, Chapter 3). (Not in the current prototype)
2. Constraints: In order to respect certain business rules, for example, a training event can only have a certain number of attendees requires some server-side processing to figure out the capacity of the event and decide if the booking should be made, since it is possible that many stores will want to book an employee on the same event, that will have a limited capacity depending on which training department hosts the event.
3. Flexibility: The RESTful API is language independent. It uses HTTPs as the transport protocol and stores the transported data as JSON; both are standardized and are compatible with many different computer languages.

[**[RESTful API documentation: Appendix A3]**](#_A3:_RESTful_API)

The following is an entity relationship diagram of the tables used to store all the necessary data normalized to third normal form.



**Diagram: Entity relationship diagram for the database system.**

All information is stored in the database. This is so that all the different systems and individual stores can have consistent data. Another reason is that all stores will be able to recover all data about employees in case of data loss at the store or training department computers.

### **1.4.3 Evaluation**

In order to evaluate the project and determine whether the product meets the user’s requirements, I have decided that the best way of achieving this is to test the use cases and show that the tested use cases correspond to the user requirements. It is worthwhile noting that these tests are designed as part of system testing and I am assuming that all the necessary unit and integration tests have been completed. Therefore, for the majority of the tests I will not attempt boundary testing nor equivalence partitioning to create input data, except for use cases UC300 and UC400. The following tests show that the system does what the user would expect and would be carried out before the user attempted acceptance testing.

#### **1.4.3.1 Testing Use Cases**

The table below lists the tests I have completed to show the use cases functioning as a user might expect. The table also shows the Use Case ID of the use Case the test is based on, the purpose of the test and whether the test was successful.

|  |  |  |  |
| --- | --- | --- | --- |
| Test ID | Use Case ID | Purpose | Status |
| 100 | UC020 | Adding an employee to the system | Pass |
| 200 | UC040 | Removing an employee from the system. | Pass |
| 300 | UC060 | Update information of an employee. | Pass |
| 400 | UC080 | View a list of all the employees that are employed at the store of the running instance. | Pass |
| 500 | UC100 | Book an employee on a training event | Pass |
| 600 | UC200 | Cancel an employee’s booking on an event. | Pass |
| 700 | UC300 | View all the employees that are overdue for any test for the store of the running instance. | Pass |
| 800 | UC400 | Send Reminders to all the employees that are overdue or 1 month away from being overdue on any test. | Pass |

**Table: List of tests completed.**

[**[Full test cases can be found in Appendix A4]**](#_A4:_Use_Case)

Tests 700 and 800 were more involved as the corresponding use cases describe the functionality that is essential to the system. This is the communication of the overdue or near to overdue employees on their training. So, I have decided to include this in the main body of the report.

#### **1.4.3.2 Test data**

The following test data was created to check boundary values. The reasoning is based on 4 different training levels of

1. junior sales assistant (JSS),
2. sales assistant (SA),
3. qualified sales assistant (QSA) and
4. senior sales assistant (SSA).

However, because a new employee has no training, a 5th option consisting of

1. no training (NT)

must be considered. Then there are the boundaries which are

1. 1 day before the test is overdue (-1),
2. on the same day (0) and
3. one day after (+1),

giving 3 possible values. So, there should be 3 x 5 = 15 tests. As demonstrated by viewing the possible test data as a cartesian product of the boundaries and types of tests.

Training = {JSS, SA, QSA, SSA, NT}

Boundaries = {-1, 0, +1}

Training x Boundaries = {(JSS, -1), (JSS, 0), (JSS, +1), (QSA, -1) …, (NT, +1)}

There is however a problem with this approach regarding the qualified assistant test, as this test requires to be tested separately as well, because the qualified assistant test requires employees to re-sit the tests annually therefore a separate value is stored in the database for that purpose with the date of the last time the test was completed.

Therefore, the test which considers the qualified assistant test would require 15 x 3 = 45 tests. This is because for each possible test considered in the previous test there is also the need to test the qualified assistant test with the 3 boundaries of one day before, on the day and one day after the due date, therefore multiplying the test data of the previous test by 3. This can be explained easier using set notation as before.

QSA Test = (Training x QSA) x Boundaries

**Note:** the cartesian products shows the amount of test data required which can be executed in 2 test cases, one including the Training x boundaries test data and another including QSA Test test data.

I will use the first part of this approach in the next 2 tests, as the first of the following tests creates a list of overdue employees for the manager to view in his/her computer in their office. The second will create a list of all employees that are 1 month from being overdue or overdue for the communication system which will send emails and text message to the respective employees in the list.

|  |  |
| --- | --- |
| Test ID | 700: View all the employees that are overdue for any test for the store of the running instance. |
| Use Case ID | UC300: View Training Progress. |
| Pre-condition | Core system is running and connected to the internet. |
| Test Data | Employees of the store where the instance is deployed in:  Store ID: st00000001  Test Date: 2019-09-10  **Employees**   |  |  |  | | --- | --- | --- | | Employee | Training level | Start date (Y-m-d) | | Dave Tanner | None | 2019-06-09 | | Simon Lister | None | 2019-06-10 | | George Willis | None | 2019-06-11 | | Dana Donnelly | Junior Sales Assistant | 2019-03-09 | | Eric Ericsson | Junior Sales Assistant | 2019-03-10 | | Stan Still | Junior Sales Assistant | 2019-03-11 | | Erika Stein | Sales Assistant | 2018-09-09 | | Mike Johnson | Sales Assistant | 2018-09-10 | | Mandy Muse | Sales Assistant | 2018-09-11 | | Ken Smith | Qualified Sales Assist. | 2018-03-09 | | Sue O’Conner | Qualified Sales Assist. | 2018-03-10 | | Ed Baker | Qualified Sales Assist. | 2018-03-11 | | Frank Heinz | Senior Sales Assistant | 2018-03-09 | | Norman Averyman | Senior Sales Assistant | 2018-03-10 | | Justin Case | Senior Sales Assistant | 2018-03-11 | |
| Expected Result | * Mr. Dave Tanner is overdue for the Junior Sales Assistant test. * Miss. Dana Donnelly is overdue for the Sales Assistant test. * Ms. Erika Stein is overdue for the Qualified Sales Assistant test. * Mr. Ken Smith is overdue for the Senior Sales Assistant test. |
| Steps | 1. Displayed on the main screen or Navigate to Training sub section Training Progress. |
| Actual Result | A list of all the employees that are overdue is returned. |
| Status | Passed |

|  |  |
| --- | --- |
| Test ID | 800: Send Reminders to all the employees that are overdue or 1 month away from being overdue on any test. |
| Use Case ID | UC400: Send training reminder. |
| Pre-condition | Communication system is running and connected to the internet. |
| Test Data | Employees of the company:  Test Date: 2019-09-10  **Employees**   |  |  |  | | --- | --- | --- | | Employee | Training level | Start date (Y-m-d) | | Dave Tanner | None | 2019-07-09 | | Simon Lister | None | 2019-07-10 | | George Willis | None | 2019-07-11 | | Dana Donnelly | Junior Sales Assistant | 2019-04-09 | | Eric Ericsson | Junior Sales Assistant | 2019-04-10 | | Stan Still | Junior Sales Assistant | 2019-04-11 | | Erika Stein | Sales Assistant | 2018-10-09 | | Mike Johnson | Sales Assistant | 2018-10-10 | | Mandy Muse | Sales Assistant | 2018-10-11 | | Ken Smith | Qualified Sales Assist. | 2018-04-09 | | Sue O’Conner | Qualified Sales Assist. | 2018-04-10 | | Ed Baker | Qualified Sales Assist. | 2018-04-11 | | Frank Heinz | Senior Sales Assistant | 2018-04-09 | | Norman Averyman | Senior Sales Assistant | 2018-04-10 | | Justin Case | Senior Sales Assistant | 2018-04-11 | |
| Expected Result | The following employees and the tests they are overdue on with their communication details.   * Mr. Dave Tanner, Junior Sales Assistant, email: davetanner@mymail.com, mobile: 07675463631. * Miss. Dana Donnelly, Sales Assistant, email: dana@mailing.com, mobile: 07678967555. * Ms. Erika Stein, Qualified Sales Assistant, ericasdsww@eee.com, mobile: 07676545454. * Mr. Ken Smith, Senior Sales Assistant, email: ksmith@mail.com, mobile: 07656463951 |
| Steps | * Run the communication system |
| Actual Result | A list of all the employees that are 1 month from being overdue or overdue is returned. |
| Status | Passed |

### **1.4.4 Legal, social, ethical and professional issues**

#### **1.4.4.1 Legal Issues: Data Protection Act 2018 (DPA 2018)**

According to the DPA 2018 the following issues must be considered for the Employee Training Management System:

* Can the person be identified by the data provided (email and telephone)?
  + While phone numbers may not directly identify someone, they can be used for identification; simply by calling the number and asking. Email addresses may also identify someone through communicating with the person, so according to the DPA 2018 section 3, this is personal data.
* The System will process data, as it collects data to monitor training, identify staff and retrieves data about a data subject in order to inform staff of training that is overdue or of any upcoming deadlines.

Further constraints on data security is made clear in Gov.uk, (2019) as one section states,

“*Everyone responsible for using personal data has to follow strict rules called ‘data protection principles’. They must make sure the information is handled in a way that ensures appropriate security, including protection against unlawful or unauthorised processing, access, loss, destruction or damage*”.

This basically means that the stored data must be protected by encryption, authorisation and authentication methods and properly backed up, all of which have been planned for the final release of the system.

A further point to note regards the automated feature of the System. If the automated part of the System is the main factor of a decision made about the data subject, then the data subject needs to be informed in writing about the automated decision process.

*“(4) Where a controller takes a qualifying significant decision in relation to a data subject based solely on automated processing—*

1. *the controller must, as soon as reasonably practicable, notify the data subject in writing that a decision has been taken based solely on automated processing,”*

Data Protection Act 2018 section 14 point 4.

Currently the only such decision is to withdraw commission earnings if the subject does not complete the Qualified Sales Assistant test every year, and future updates may also allow for automated decisions with significant consequences. For example, an update to the system creates a list of employees that will lose their employment as a result of not completing the required training 4 months after the deadline. This kind of operation will fall under section 14.4 of the DPA 2018 and requires written notice to the employee, which could easily be automated using the system.

Note: The DPA 2018 is a modification of the GDPR which is highlighted by the following statements “*the Act exercises a number of agreed* ***modifications*** *to the GDPR*” (Service.gov.uk, 2018). This means that the GDPR (General Data Protection Regulation) must be read alongside the DPA 2018.

#### **1.4.4.2 Possible Professional and Ethical Issues**

* A message sent at an inappropriate time. For example, a message is sent at 3 am. This should be avoided and certain actions such as turning the communication system off at night and back on again in the morning can be a possible solution.
* No message sent at all due to a system problem. Again, this could cause employees to lose their commission earnings, however a simple daily test message to the managers should help to reduce the possibility of this problem. In the case that no message is sent the managers can contact the IT/Training department and notify them that they did not receive a message, therefor the System is not working as intended.

#### **1.4.4.3 Social and Ethical Issues**

The system could be a factor in making certain employees redundant, particularly the people employed in the training department. This is not the intent of the system; however, such outcomes are possible due to the automation provided by the system. Considering that the main task of the system is to automate part of a job otherwise done by a person; making people redundant is possible.

Furthermore, the modular nature of the system architecture would make it possible to update the system in various ways, for example, machine learning algorithms could be used to better predict when an employee is likely to start losing interest in training materials and forget about training altogether. As helpful as machine learning algorithms may be to businesses, such algorithms can also reduce the number of employees required in various areas of a business, hence creating a social and ethical issue, whereby the developer and the owners of the business need to consider the effect on society before committing to profit.

Considering that the intent of the System is to reduce the workload of the training department so that the training program can be improved, I am not concerned about employees losing their jobs in this instance, but such actions may ultimately become reality and it is important to be aware of them.

#### **1.4.4.4 Personal Professional Conduct**

All the tasks I have attempted have been taken seriously and I have maintained a professional attitude towards the projects. This can be seen by many aspects, some of which are:

* + All research material has been retrieved from trustworthy sources provided by the OU library.
  + All sources of information have been scrutinized for their credibility, as can be seen in the information sources section.
  + I have considered the rights of the people using the system and planned security methods to ensure that personal data of the subjects is not revealed to any unauthorized sources.
  + I have maintained the quality of the project to the highest possible, and where I was not sure I clearly state any limitations, or any assumptions made regarding the project work, for example I make assumptions of Unit testing being complete and stated this, as I did not have the time to do all the required testing.
  + I have acknowledged all sources of information and clearly stated those sources where it was relevant, evident mainly in the information source, but also in other sections of the report.
  + I have not discriminated against any group of people and attempted to make the product as accessible as possible for the task at hand (Bcs.org, 2019), by providing an extra title of “mx” for people that do not wish to disclose their gender, or marital status, and I have also planned to use a color scheme that is usable for people that are color blind.
  + I have also generated relevant documentation like the RESTful API documentation that would be useful for any developer that would need to use the API for improvements to the system.

In following these rules, I believe that I have met the requirements of the BCS code of conduct that are relevant for my project (Bcs.org, 2019).

# **2 Review**

## **2.1 Current state of project**

The current project is still at a prototype stage. I believe this was due to the lack of experience I have working with larger projects, as I attempted a project which I believe is too large and requires knowledge across many disciplines to be tackled by one person. Regardless, I am happy at the stage I have managed to get the project to and if I had more time these are the things that still need to be done and a possible timeframe, assuming I was to continue the project myself.

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Current state | Work remaining | Time required |
| User interface | Early prototype | * Add labels for input and output areas. * Create a design and test user interaction | 2 weeks |
| Core System (Training department) | Not attempted | * Create based on the store variant of the core system. * Add the three extra use cases which will also require that they are added to the RESTfull API. * Test the system | 3 weeks |
| Security | Planning phase | * Change request URLs to HTTPs instead of HTTP. * Create a table to store the password for users. | 2 week |
| Communication System | Proof of concept | * Set up an email server for sending emails to employees about overdue training. * Create a way of sending Text messages to employees. | 12 weeks\* |

**Table: list of tasks remaining**

\*I am assuming I would have to set up an email server and create the system for sending text messages, e.g. an Arduino projects. However, if the email server existed and a service was used to send text messages than the time required would be 1 week.

### **2.1.2 Skills and Resources Required to Finish the Project.**

|  |  |  |
| --- | --- | --- |
| Skill / Resource | | Current choice and justification |
| Resource | Potential users: Regular individuals with little experience of using computers and other related technology. | Using friends and family is the most suitable solution, as I can be sure of their level of IT competence, due to my relationship with them. Furthermore, selecting people with little IT experience will help to improve the usability of the product as required by NRF5. |
| Skill | Creating a usable user interface | Using Johnson’s (2014) Designing with the mind in mind should be enough to finish the project as the user interface is not designed to attract users; it is designed to be usable. |
| Skill | Database: Security | Deploying using an IaaS provider like Amazon AWS or Microsoft Azure, should help security as these companies are actively involved in providing the best possible security (Rountree and Castrillo, 2014, p. 10), unlike an inhouse solution which would be limited to the funds available. |
| Resource  /Skill | Communication system: email. | The most secure solution I could find to send emails is to set up an email server. My current understanding is limited, and this would require me to invest a lot of my time, but I believe that the benefits outweigh the risks. |
| Resource  /Skill | Communication system: text. | I believe an inhouse solution is the best option, this is because it should be cheaper and more secure than using a service, since the data does not leave the company. There is a lot of information on how to accomplish this online, which should make this a viable option. |
| Resource  /Skill | Communication system security | Creating an inhouse solution would be the best choice, as this would allow for the company to secure the staff data, and they would not have to rely on a third party handling the data. |
| Resource | Hardware | According to NFR7 the product must be able to run on Windows, Mac OS and Linux Ubuntu. This will require extensive testing but should be possible for Windows and Linux Ubuntu since they can be run as virtual machines, however Macintosh does not currently support running its OS on a virtual machine which may prove challenging when it comes to testing on macOS. |

## **2.2 Project Management Review**

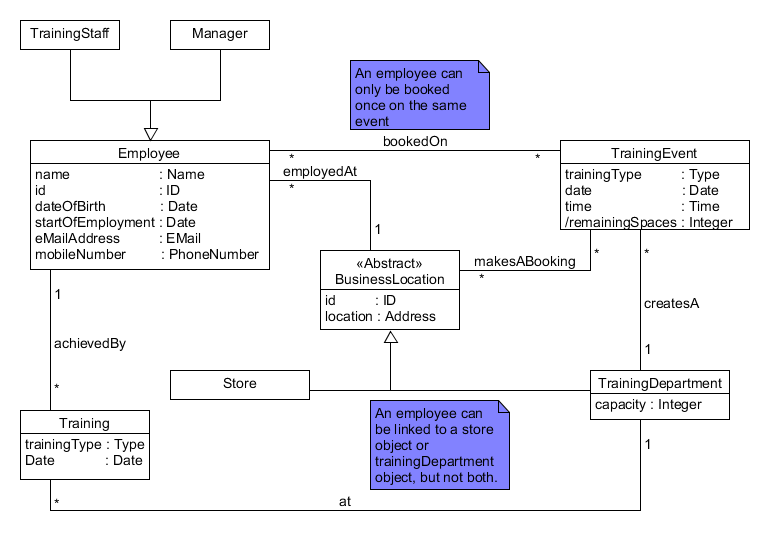
### **2.2.1 Project Management Lifecycle**

I chose to use the Rational Unified Process (RUP), the reason why I would use this methodology is mainly to do with the flexibility and level of documentation produced by following the RUP method. The one of the reasons I cited were that RUP allows for early prototypes, which can be used in lieu of a fully working final version; That’s exactly what happened with my project, as I have a prototype which can be used but is not a finished product. The alternatives I had considered were the waterfall method, which would have meant that I would not have any working prototype, as I would most likely be at an earlier stage of the project. An Agile method, would have enabled me to have a working prototype, but the lack of documentation would have made it difficult for me to work on this project, because I had 3 other modules to study for and switching between those modules and the project would have been very difficult, if I did not have the proper documentation to remind me of where I was in the project and what needs to be done.

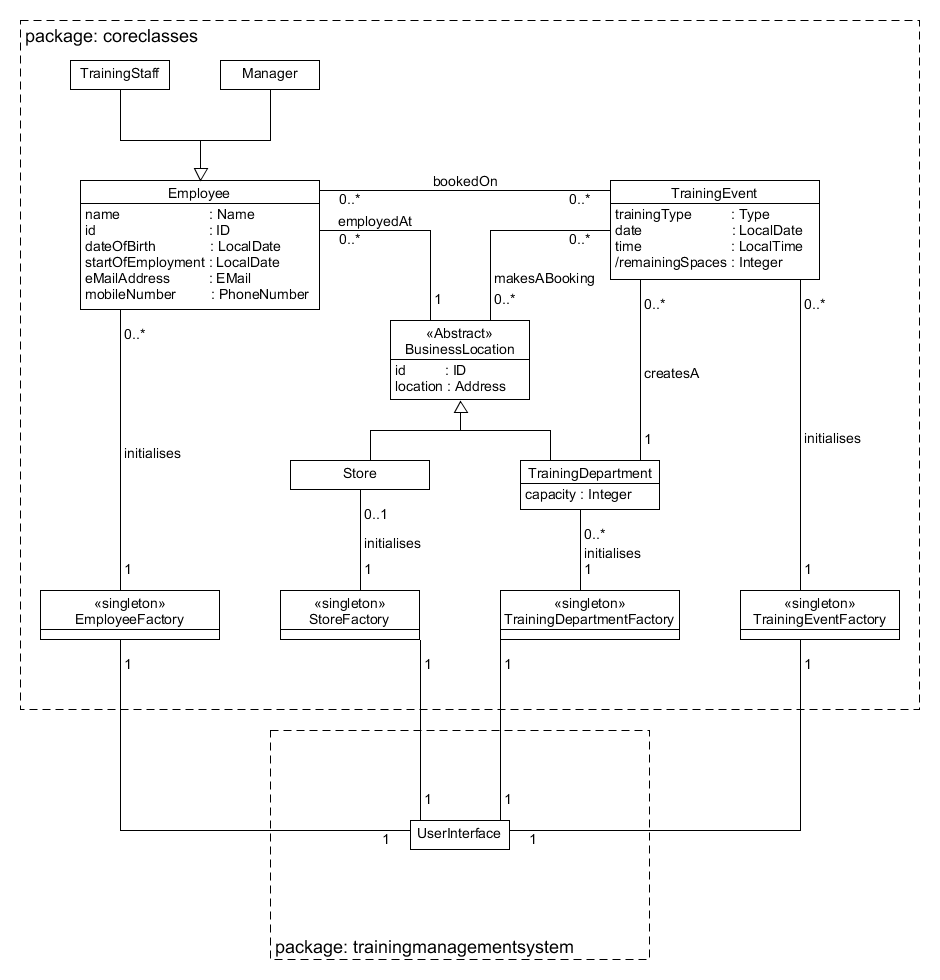
### **2.2.2 Project Plan**

I have managed to stick to the project plan up to the point where I realized that I would need to introduce a RESTful API. The reason for the API is to increase security by allowing the system to store users and their log in credentials in a secure form. This unforeseen addition caused me a great setback, as I had to remodel the core system and create a RESTful API. Had I originally thought of this I would have still used Java and created a similar system, because, the core system can be described as a thick client and it handles a lot of the processing and temporary storage of data. This reduces the load on the database server and increases overall system performance, since the alternative would mean that for every use case a call to the database system would be made, unlike the thick client that initializes at start up and stores data for the system to deal with the majority of the use cases locally without having to query the database.

[**[The full plan including the adjustments can be seen in Appendix C]**](#_Appendix_C:_Schedule:)



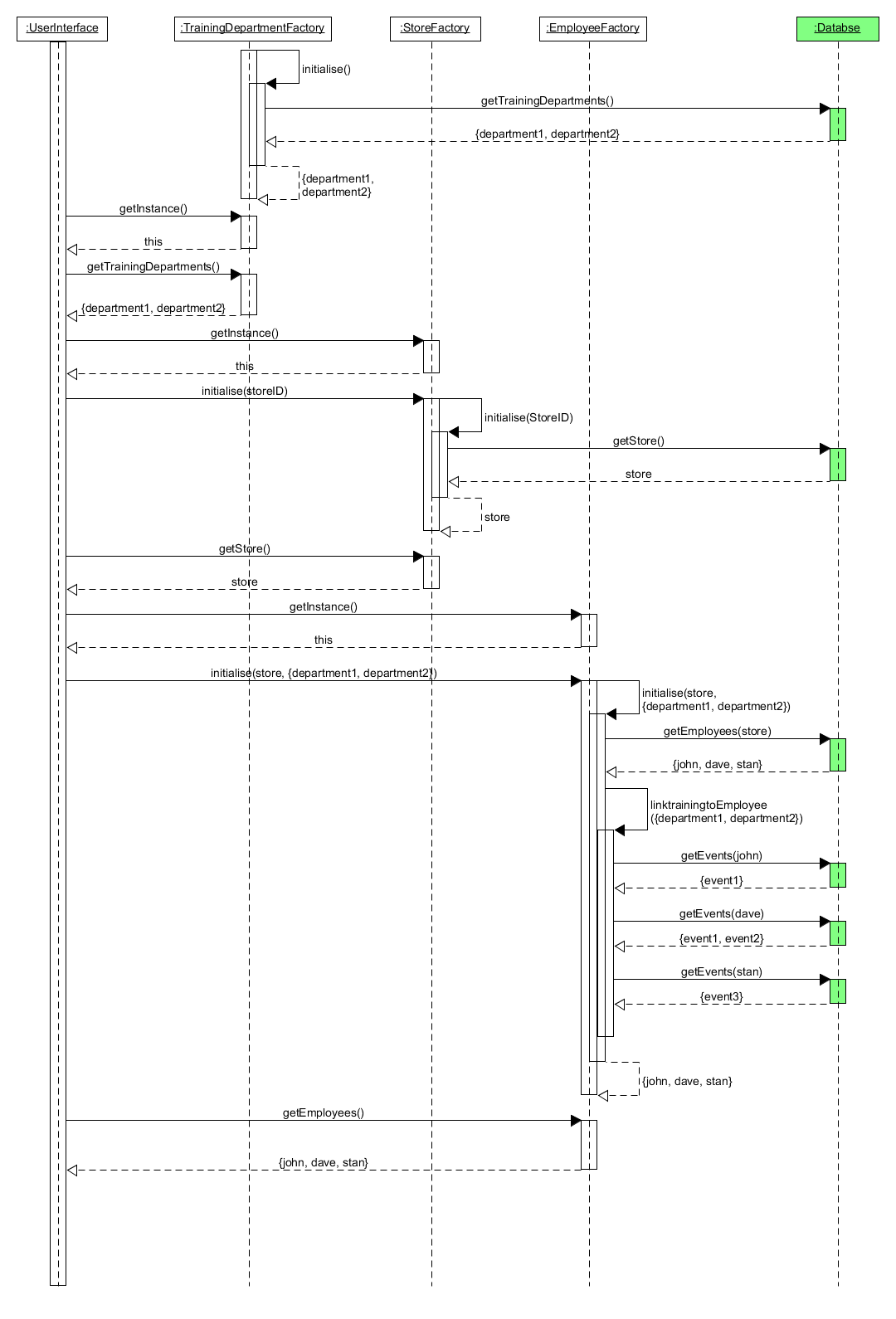
**Image: Before, Class diagram of the Core system before the introduction of a RESTful API.**



**Image: After, Class diagram of the Core system after the introduction of a RESTful API.**

The main difference are the classes labelled as factories. These classes will handle the communication of the system with the RESTful API and are responsible for creating the relevant objects and associations between them when the system starts up.

The following sequence diagram show how these factories are used to initialize the system at start up.



**Image: Sequence diagram showing the initialization process at system start up.**

**Note:**

* All the message names going to the database have been introduced for the purpose of abstracting away the true nature of the https calls from java to the database.
  + The methods corresponding to the messages do not exist in the system.
* All the classes ending with factory are singletons, therefore no object names are given, as there is no need to differentiate between objects of the class.
* All messages returned are objects, but the creation was left out as that would have introduce too much information in the diagram, and again for the sake of abstraction I have chosen to exclude showing the creation of Employee, TrainingDepartment and Store objects.

# **3 Appendix**

## **Appendix A: Supplementary evidence:**

### **A1: Use Case Descriptions**

The following tables show further details of the use cases based on Armour and Miller’s. (2000) Advanced Use Case Modeling: Vol. 1: Software Systems.

|  |  |
| --- | --- |
| Use case name | Add Employee |
| Use case ID | UC020 |
| Primary Actors | Manager and training staff |
| Secondary Actor | None |
| Brief Description | A new employee is added to the training system. |
| Pre-condition | None |
| Flow of events | 1. The training staff/manager logs in to the computer, opens the Application (Core System), navigates to Employee, sub section Add. 2. The system displays a form that requires all the necessary information to create a new employee. 3. The training staff/manager enters the employee id, full name, date of birth, start of employment date, email address and phone number and submits the form. 4. The system validates the input. 5. The system adds the employee to the database and currently running instance of the core system. |
| Post-condition | A new employee is added to the core system and database. |
| Alternative flow and exceptions | 1. The input for any of the required information is invalid. 2. The system informs the user of the invalid data that was entered. 3. Go to 4 |
| Assumptions | Employee ID is not generated by the system and is provided from another system, e.g. payroll. |
| Issues | None |

|  |  |
| --- | --- |
| Use case name | Remove Employee |
| Use case ID | UC040 |
| Primary Actors | Manager and training staff |
| Secondary Actor | None |
| Brief Description | An existing employee is removed from the training system. |
| Pre-condition | None |
| Flow of events | 1. Training staff/manager logs in to the computer, opens the Application (Core System), navigates to Employee, sub section Remove. 2. The core system displays a list of employees belonging to the store where the core system is deployed in. 3. The training staff/manager selects the employee to remove. 4. The system removes the employee from the database and the currently running instance. |
| Post-condition | The employee is removed from the core system and database. |
| Alternative flow and exceptions | None |
| Assumptions | Not possible to choose an employee that does not exist as the training staff/manager input is restricted by the user interface to a provided list of employees, that originates from the database. |
| Issues | Training departments must have an extra option for providing a store ID for viewing employees by store. |

|  |  |
| --- | --- |
| Use case name | Edit Employee |
| Use case ID | UC060 |
| Primary Actors | Manager and training staff |
| Secondary Actor | None |
| Brief Description | An existing employee’s information is edited on the system. |
| Pre-condition | None |
| Flow of events | 1. The training staff/manager logs in to the computer, opens the Application (Core System), navigates to Employee, sub section Edit. 2. The core system displays a list of employees belonging to the store where the core system is deployed in. 3. The training staff/manager selects the employee to edit. 4. The system populates an editable form with all the employee’s information that can be edited by the core system. 5. The manager makes necessary amendments to the information and confirms the changes. 6. The core system validates the information. 7. The database system updates the information of the employee and informs the user of the successful update. |
| Post-condition | The employee information is updated in the core system and database. |
| Alternative flow and exceptions | 1. The system informs the user that the new changes consist of data that is not valid. 2. Go to 5 |
| Assumptions | Not possible to choose an employee that does not exist as the training staff/manager input is restricted by the user interface to a provided list of employees, that originates from the database. |
| Issues | Training departments must have an extra option for providing a store ID for viewing employees by store. |

|  |  |
| --- | --- |
| Use case name | View employees. |
| Use case ID | UC080 |
| Primary Actors | Manager and training staff |
| Secondary Actor | None |
| Brief Description | The system retrieves all the employees linked to a business location. |
| Pre-condition | None |
| Flow of events | 1. The training staff/manager requests a list of employees. 2. The system finds all the employees of the store and displays them. |
| Post-condition | None |
| Alternative flow and exceptions | None |
| Assumptions | None |
| Issues | None |

|  |  |
| --- | --- |
| Use case name | Book training event. |
| Use case ID | UC100 |
| Primary Actors | Manager and training staff |
| Secondary Actor | None |
| Brief Description | The training staff/manager books a training event for an employee. |
| Pre-condition | The training event must have open space for participants and the employee has not already completed the training for which the training event is for. |
| Flow of events | 1. The training staff/manager logs in to the computer, opens the Application, navigates to Training, subsection Make Booking (Core System) 2. The system returns a list of all the employees belonging to the store. 3. The training staff/manager selects the employee that is to attend the training event from a list of employees, and the training type. 4. The Core System returns a list of available events for the training type. 5. The training staff/manager selects the relevant event and clicks on confirms the booking by clicking on a button. 6. The Core System updates the Database System with the booking information, and the currently running instance of the Core system. |
| Post-condition | The employee is booked on a training event and the list of employees attending the training event is updated with the selected employee. |
| Alternative flow and exceptions | 1. The Core System cannot update the database with the information as the event has reached its maximum capacity by the time the manager confirms the booking. 2. The core system informs the manager that the event is fully booked and that the employee was not booked on the event. |
| Assumptions | None |
| Issues | None |

|  |  |
| --- | --- |
| Use case name | Cancel training event. |
| Use case ID | UC200 |
| Primary Actor | Manager and training staff |
| Secondary Actor | None |
| Brief Description | The training staff/manager cancels a training event for an employee. |
| Pre-condition | The employee must be booked on the training event. |
| Flow of events | 1. The training staff/manager logs in to the computer, opens the Application (Core System), navigates to Training, sub section Cancel Booking. 2. The system returns a list of all the employees belonging to the same store as the manager. 3. The training staff/manager selects the employee that wants to cancel the training event. 4. The core system returns a list of all the events the employee is booked on. 5. The training staff/manager selects the training event that should be cancelled. 6. The Core System removes the event from the running instance of the core system and communicates with the Database System to remove the employee from the training event. |
| Post-condition | The employee’s booking on a training event is cancelled. The training event is updated with the cancellation. |
| Alternative flow and exceptions | None |
| Assumptions | None |
| Issues | None |

|  |  |
| --- | --- |
| Use case name | View Training Progress |
| Use case ID | UC300 |
| Primary Actors | Manager and training staff |
| Secondary Actor | None |
| Brief Description | The training staff/manager retrieves the training progress of an employee. |
| Pre-condition | The employee must be employed at the same store as the manager |
| Flow of events | 1. The training staff/manager logs in to the computer and opens the Application (Core System) 2. The system returns a list of overdue employees, consisting of employee name and which training type the employee is overdue on. |
| Post-condition | None |
| Alternative flow and exceptions | 1. The training staff/manager logs in to the computer and opens the Application (Core System) and navigates to Training sub section View Progress 2. The system returns a list of overdue employees, consisting of employee name and which training type the employee is overdue on. |
| Assumptions | The employee must be employed at the same store as the manager if the manager is attempting to view the training progress. Training staff can view the training progress of all staff. |
| Issues | None |

|  |  |
| --- | --- |
| Use case name | Send training reminder. |
| Use case ID | UC400 |
| Primary Actor | Communication System |
| Secondary Actor | Employee |
| Brief Description | The Communication System informs any employee with overdue or upcoming training deadlines of the deadline. |
| Pre-condition | None |
| Flow of events | 1. The Communication System checks the training progress of all employees, by querying the Database System. 2. A list of employees with their phone numbers and email addresses is created for any employee that is overdue or has less than 1 month to complete any of the 4 required tests. 3. Every employee on the list is sent a text message and email reminding them of their upcoming or overdue training. |
| Post-condition | None |
| Alternative flow and exceptions | None |
| Assumptions | None |
| Issues | None |

|  |  |
| --- | --- |
| Use case name | Update training record. |
| Use case ID | UC500 |
| Primary Actor | Training Staff |
| Secondary Actor | None |
| Brief Description | Training staff update the training record of employees to state whether the training was achieved at an acceptable standard after a training event or in case of an error in the record. |
| Pre-condition | The employee must have attended the training event. |
| Flow of events | 1. A member of the training department logs in to the computer, opens the Application, navigates to Training, subsection Update training record. 2. The training department member enters the training event ID. 3. The Core System displays a list of all employees that were booked on the event. 4. The training department member selects an employee. 5. The core system displays a form asking whether the employee attended and what score they achieved. 6. The training department member submits the form. 7. The Core System updates the information in the database and currently running instance of the core system. |
| Post-condition | The training record of the employee is updated with the new training achieved. |
| Alternative flow and exceptions | 1. Training staff enters an employee ID. 2. Go to 5 |
| Assumptions | None |
| Issues | None |

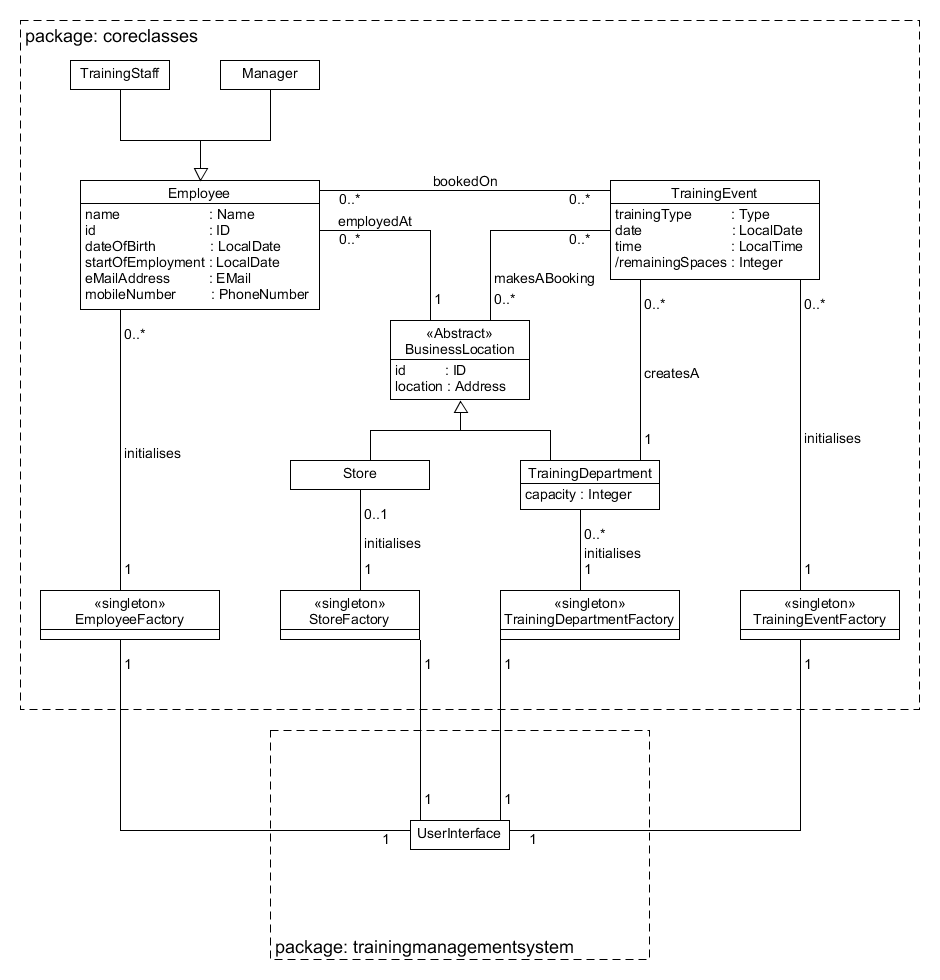
|  |  |
| --- | --- |
| Use case name | Add Training Event |
| Use case ID | UC600 |
| Primary Actor | Training Staff |
| Secondary Actor | Data base system |
| Brief Description | Training staff add training events for employees to attend. |
| Pre-condition | None |
| Flow of events | 1. A member of the training department logs in to the computer, opens the Application, navigates to Training, subsection Add Event. 2. The core system displays a form asking for the date, time of the event and which type of training its covers, e.g. Junior sales assistant, qualified sales assistant… 3. A member of the training department enters and submits the information. 4. The system communicates with the database and checks if an event with the same information does not already exist. 5. The system informs the training staff that the event was created. |
| Post-condition | A new Training event is created on the database system. |
| Alternative flow and exceptions | 1. The system informs the staff that the event has already been created. 2. Go to 3. |
| Assumptions | None |
| Issues | None |

|  |  |
| --- | --- |
| Use case name | Remove Training Event |
| Use case ID | UC700 |
| Primary Actor | Training Staff |
| Secondary Actor | Data base system |
| Brief Description | Training staff remove training events from the database. |
| Pre-condition | None |
| Flow of events | 1. A member of the training department logs in to the computer, opens the Application, navigates to Training, subsection Remove Event. 2. The system displays a list of all training events that have been added for the training department. 3. The member of the training department chooses or searches for the training event to remove and submits this information. 4. The system communicates with the database and checks if an event with the same information exists. 5. The system informs the training staff that the event was removed. 6. The mobile number and email address of every member of staff that was booked on the event is recorded in the database. 7. The communication system sends a message to every employee that was booked on the event, informing them of the cancelation. |
| Post-condition | An existing Training event is removed from the database and core system. |
| Alternative flow and exceptions | 1. The system informs the staff that the event does not exist 2. Go to 3. |
| Assumptions | None |
| Issues | None |

|  |  |
| --- | --- |
| Use case name | Get Training Events |
| Use case ID | UC800 |
| Primary Actor | Training staff, manager |
| Secondary Actor | Data base system |
| Brief Description | Training staff or managers view training event on the database system. |
| Pre-condition | None |
| Flow of events | 1. Training staff/manager log in to their computer and open the Application (Core System) 2. Training staff/manager select the option to view training events. 3. The system returns a list of all the available events. |
| Post-condition | None |
| Alternative flow and exceptions | None |
| Assumptions | None |
| Issues | None |

### **A2: Core System (Store) Implementation model**

#### **Class Diagram**



**Diagram: Class diagram for the core system.**

#### **Class descriptions**

**Package:** coresystem

|  |  |  |
| --- | --- | --- |
| **Class** | Employee | An employee of the business  **Generalises** Manager and TrainingStaff |
| **Attributes** | | |
|  | Private Name name  Name of the employee | |
|  | Private Id id  The unique id of the employee | |
|  | Private LocalDate startOfEmployment  The first day of employment | |
|  | Private LocalDate dateOfBirth  Date of birth of the employee | |
|  | Private EMail emailAddress  Email address of the employee | |
|  | Private PhoneNumber mobileNumber  Mobile phone number of the employee | |
| **Links** | | |
|  | Private Store store  References the linked store object. | |
|  | Private Collection<TrainingEvent> trainingEvents  References a collection of all linked TrainingEvent objects. | |
| **Constructors** | | |
|  | Public Employee(Name aName, ID anId, LocalDate aDateOfBirth, LocalDate aStartOfEmployment, EMail anEMailAddress, PhoneNumber aMobileNumber, Store aStore)  Post-condition: Initialises a new Employee object with the given attribute values. | |
|  | Public Employee(Name aName, ID anId, LocalDate aDateOfBirth, LocalDate aStartOfEmployment, EMail anEMailAddress, PhoneNumber aMobileNumber, TrainingDepartment aTrainingDepartment)  Post-condition: Initialises a new Employee object with the given attribute values. | |
| **Protocol** | | |
|  | Public void addTrainingEvent(TrainingEvent aTrainingEvent)  Post-condition: A reference to aTrainingEvent is recorded. | |
|  | Public Collection<TrainingEvent> getTrainingEvents()  Post-condition: Returns all linked TrainingEvent objects. | |
|  | Public void removeTrainingEvent(TrainingEvent aTrainingEvent)  Post-condition: Removes the reference to aTrainingEvent from trainingEvents. | |
|  | Public void removeStore()  Post-condition: Removes the reference to the linked store object. | |
|  | Public Name getName()  Post-condition: Returns name. | |
|  | Public ID getId() | |
|  | Post-condition: Returns id as an ID object. | |
|  | Public String getIdAsString()  Post-condition: Returns id as a String object. | |
|  | Public LocalDate getDateOfBirth()  Post-condition: Returns dateOfBirth. | |
|  | Public LocalDate getStartOfEmployment()  Post-condition: Returns startOfEmployment. | |
|  | public String geteMailAddress()  Post-condition: Returns emailAddress as a String object. | |
|  | public PhoneNumber getMobileNumber()  Post-condition: Returns mobileNumber. | |
|  | public void setName(Name aName)  Post-condition: sets name to aName. | |
|  | public void setDateOfBirth(LocalDate aDateOfBirth)  Post-condition: sets dateOfBirth to aDateOfBirth. | |
|  | public void setStartOfEmployment(LocalDate aStartOfEmployment)  Post-condition: sets startOfEmployment to aStartOfEmployment. | |
|  | public void seteMailAddress(EMail anEMailAddress)  Post-condition: sets emailAddress to anEMailAddress. | |
|  | public void setMobileNumber(PhoneNumber aMobileNumber)  Post-condition: sets mobileNumber to aMobileNumber. | |
|  | public Collection<String> employeeInformation()  Post-condition: Returns a list of strings containing the string value of name, mobileNumber, emailAddress, startOfEmployment, dateOfBirth and the linked trainingEvents. | |
|  | public String toString()  Post-condition: Returns a string representation of name. | |

|  |  |  |
| --- | --- | --- |
| **Class** | Manager | A manager in one of the stores  **Specialises** Employee |
| **Constructor** | | |
|  | public Manager(Name aName, ID anId, LocalDate aDateOfBirth, LocalDate  aStartOfEmployment, EMail anEMailAddress, PhoneNumber aMobileNumber,  Store aStore)  Post-condition: Initialises a new Manager object with the given attribute  values. | |

|  |  |  |
| --- | --- | --- |
| **Class** | TrainingStaff | A member of the training staff  **Specialises** Employee |
| **Constructor** | | |
|  | public TrainingStaff(Name aName, ID anId, LocalDate aDateOfBirth, LocalDate  aStartOfEmployment, EMail anEMailAddress, PhoneNumber aMobileNumber,  TrainingDepartment aTrainingDepartment)  Post-condition: Initialises a new Manager object with the given attribute  values. | |

|  |  |  |
| --- | --- | --- |
| **Class** | BusinessLocation | One of many physical building operating under the business name.  **Generalizes** Store and TrainingDepartment |
| **Attributes** | | |
|  | Private ID id  A unique id for the premises | |
|  | Private Address location  The address of the premises. | |
| **Links** | | |
|  | Private Map<ID, Employee> employees  References all linked employee objects | |
| **Constructor** | | |
|  | Public BusinessLocation(ID anID, Address aLocation)  Post-Condition: Initialises a new BusinessLocation object with the given attribute values. | |
| **Protocol** | | |
|  | public Map<ID, Employee> getEmployees()  Post-condition: Returns a Map of ID and Employee objects. Each ID object in the key value pair (ID, Employee) corresponds to the employees ID in the pair. | |
|  | public void setEmployees(Map<ID, Employee> employeeList)  Post-condition: employees references the provided employeList | |
|  | public void addEmployees(IEmployee anEmployee)  Post-condition: A reference to anEmployee is recorded. | |
|  | public ID getId()  Post-condition: Returns id as an ID object. | |
|  | public String getIDAsString()  Post-condition: Returns id as a string object. | |
|  | public Address getLocation()  Post-condition: Returns location | |
|  | public void setLocation(Address aLocation)  Post-condition: sets location to aLocation. | |
|  | public String toString()  Post-condition: returns a string representation of id, location and employees. | |

|  |  |  |
| --- | --- | --- |
| **Class** | Store | A physical store of the business  **Specialises** BusinessLocation |
| **Constructor** | | |
|  | public Store(ID anId, Address aLocation)  Post-condition: Initialises a new Store object with the given attribute values. | |
|  | public Store(Store aStore)  Post-condition: Initialises a new Store object with the same attribute values as aStore. | |
| **Protocol** | | |
|  | public void removeEmployee(Employee anEmployee)  Post-condition: The reference to anEmployee is removed. | |

|  |  |  |
| --- | --- | --- |
| **Class** | TrainingDepartment | A training department for the business  **Specialises** BusinessLocation |
| **Attributes** | | |
|  | Private int capacity  The maximum number of people the department can accommodate. | |
| **Links** | | |
|  | private Collection <TrainingEvent> trainingEvent  References all linked employee objects. | |
| **Constructor** | | |
|  | public TrainingDepartment(ID anId, Address aLocation, int aCapacity)  Post-Condition: Initialises a new TrainingDepartment object with the given attribute values. | |
| **Protocol** | | |
|  | public void addTrainingEvent(TrainingEvent aTrainingEvent)  Post-condition: Records a reference to aTrainingEvent. | |
|  | public int getCapacity()  Post-condition: Returns capacity. | |
|  | public void setCapacity(int aCapacity)  Post-condition: Sets the value of capacity to that of aCapacity. | |
|  | public String toString()  Post-condition: Returns id, location, capacity, and number of linked trainingEvents as a string. | |

|  |  |  |
| --- | --- | --- |
| **Class** | TrainingEvent | A training event for employees to attend. |
| **Attributes** | | |
|  | private int id  A unique id for the training event created by the database system. | |
|  | private Type trainingType  The type of training covered at the event | |
|  | private LocalDate date  The date of the event | |
|  | private LocalTime time  The start time of the event | |
|  | [int / placesRemaining  Number of places still available for booking at the training event] | |
| **Links** | | |
|  | private TrainingDepartment trainingDepartment  References linked training department. | |
|  | Private Collection <Employee> employees | |
|  | References all linked employees. | |
| **Constructors** | | |
|  | public TrainingEvent(Type aTrainingType, TrainingDepartment aTrainingDepartment, LocalTime aTime, LocalDate aDate)  Post-Condition: Initialises a new TrainingEvent object with the given attribute values. | |
|  | public TrainingEvent(Type aTrainingType, TrainingDepartment aTrainingDepartment, LocalTime aTime, LocalDate aDate, int anID)  Post-Condition: Initialises a new TrainingEvent object with the given attribute values. | |
| **Protocol** | | |
|  | public void addEmployee(Employee anEmployee)  Pre-condition: capacity of trainingDapartment is less than the number of referenced objects in employees, else an exception is thrown.  Post-condition: A reference to anEmployee is recorded and anEmployee records a reference to this object. | |
|  | public void removeEmployee(Employee anEmployee)  Post-condition: The reference to anEmployee is removed and anEmployee removes its reference to this object. | |
|  | public int getPlacesRemaining()  Post-condition: Returns placesRemaining. | |
|  | public Type getTrainingType()  Post-condition: Returns trainingType. | |
|  | public LocalDate getDate()  Post-condition: Returns date. | |
|  | public LocalTime getTime()  Post-condition: Returns time. | |
|  | public int getId()  Post-condition: Returns id. | |
|  | public Collection <Employee> getEmployees()  Post-condition: Returns employees. | |
|  | public String toString()  Post-condition: Returns a String representation of trainingType, date, time, and the reference of the location attribute from trainingDepartment. | |

|  |  |  |
| --- | --- | --- |
| **Enum** | Type | Types of training Events available |
|  | **Values** | JuniorSalesAssistant, SalesAssistant, QualifiedSalesAssistant, and SeniorSalesAssistant |

|  |  |  |
| --- | --- | --- |
| **Class** | EmployeeFactory | Responsible for Retrieving, adding and editing employees from the database and initialising the system on start up with the other factory classes.  **Singleton** |
| **Links** | | |
|  | private static EmployeeFactory employeeFactory  creates a static reference to employeeFactory. | |
|  | private Map<ID, Employee> employees  References all linked employee objects. | |
| **Constructor** | | |
|  | private EmployeeFactory ()  Post-condition: Creates an instance of EmployeeFactory if one does not exist, otherwise returns employeeFactory. | |
| **Protocol** | | |
|  | public static Coordinator getInstance() | |
|  | Post-condition: Returns the only instance of employeeFactory. | |
|  | public void initialise(Store aStore)  Post-condition:   1. aStore records a reference to all Employee objects retrieved from the Database System which are employeed in the real world store, and 2. all Employee objects record a Reference to aStore. | |
|  | private void linkTrainingToEmployee(Collection<TrainingDepartment> trainingDepartments)  Post-condition:   1. For each Employee object referenced by employees record a reference to TrainingEvent trainingEvent retrieved from the Database system. 2. For each trainingEvent referenced by employee find the TrainingDepartment object from trainingDepartments that the trainingEvent is hosted in and record a reference to the trainingDeartment in trainingEvent. | |
|  | public void addEmployee(Employee anEmployee, Store aStore)  Post-condition:   1. Database system records the attribute values of anEmployee and the storeID attribute value from aStore.    1. anEmployee a new Employee object is created    2. a referenced to aStore is recorded in anEmployee    3. a reference to anEmployee is recorded in aStore. | |
|  | public void updateEmployee(Employee anEmployee)  Post-condition:   1. If an employee object referenced by employees exists with the same employeeID attribute value as anEmployee than the employee objects attribute values are set to that of anEmployees attribute values. 2. The database updates the employee values corresponding to the employee with the same employeeID as anEmployee. | |
|  | public String removeEmployee(Employee anEmployee, Store aStore)  Post-condition:   1. An employee record from the database system is removed, if an employee with the same id as the employeeID attribute value of anEmployee exists, else returns a message consisting of the error thrown by the database. 2. The reference to anEmployee in aStore and employees is removed. | |
|  | public Map<ID, Employee> getEmployees()  Post-condition: Returns Employees | |
|  | public String toString()  Post-condition: employees as a string. | |

|  |  |  |
| --- | --- | --- |
| **Class** | StoreFactory | Responsible for creating Store object from the database and initialising the system on start up with the other factory classes.  **Singleton** |
| **Links** | | |
|  | private static StoreFactory storeFactory  creates a static reference to storeFactory. | |
|  | private Store store  References the linked store object. | |
| **Constructor** | | |
|  | private StoreFactory(ID anID)  Post-condition: Creates an instance of StoreFactory if one does not exist, otherwise returns storeFactory. | |
| **Protocol** | | |
|  | public static StoreFactory getInstance() | |
|  | Post-condition: Returns the only instance of storeFactory. | |
|  | private void initialise(ID anID)  Post-condition:   1. A new Store object aStore is created. 2. The attribute values of aStore are set to the corresponding values found in the database that are found in the record with the storeID the same as anID. | |
|  | public Store getStore()  Post-condition: Returns store. | |
|  | public String toString()  Post-condition: Returns store as a string. | |

|  |  |  |
| --- | --- | --- |
| **Class** | TrainingDepartmentFactory | Responsible for creating TrainingDepartment objects from the database and initialising the system on start up with the other factory classes.  **Singleton** |
| **Links** | | |
|  | private static TrainingDepartmentFactory trainingDepartmentFactory  creates a static reference to trainingDepartmentFactory. | |
|  | private Collection <TrainingDepartment> trainingDepartments  References all the training departments in the system. | |
| **Constructor** | | |
|  | private TrainingDepartmentFactory()  Post-condition:   1. Creates an instance of TrainingDepartmentFactory if one does not exist, otherwise returns trainingDepartmentFactory. 2. TrainingDepartment objects have been created and a reference recorded for Training departments that correspond to every entry for a training department in the Database. | |
| **Protocol** | | |
|  | public static TrainingDepartmentFactory getInstance() | |
|  | Post-condition: Returns the only instance of trainingDepartmentFactory. | |
|  | private void initialise()  Post-condition: Post-condition: TrainingDepartment objects have been created and a reference recorded for Training departments that correspond to every entry for a training department in the Database. | |
|  | public Collection <TrainingDepartment> getTrainingDepartments()  Post-condition: Returns trainingDepartments. | |
|  | public String toString()  Post-condition: Returns trainingDepartments as a string. | |

|  |  |  |
| --- | --- | --- |
| **Class** | TrainingEventFactory | Responsible for creating and managing TrainingEvent objects retrieved from the database and initialising the system on start up with the other factory classes.  **Singleton** |
| **Links** | | |
|  | private static TrainingEventFactory trainingEventFactory  creates a static reference to trainingEventFactory. | |
| **Constructor** | | |
|  | private TrainingEventFactory()  Post-condition: Creates an instance of TrainingEventFactory if one does not exist, otherwise returns trainingEventFactory. | |
| **Protocol** | | |
|  | public static TrainingEventFactory getInstance() | |
|  | Post-condition: Returns the only instance of trainingEventFactory. | |
|  | public Set<TrainingEvent> findTrainingEvents(Type aType, Set<TrainingDepartment> trainingDepartments)  Post-condition: Post-condition: Returns a list of TrainingEvent objects with the attribute value of type equal to aType and appropriate associations are created between TrainingEvent and TrainingDepartment objects. | |
|  | public boolean makeBooking(Employee anEmployee, TrainingEvent anEvent)  Post-condition:   1. The database is updated with the booking for anEmployee on anEvent. 2. anEmployee records a reference to anEvent and anEvent records a reference to anEmployee. | |
|  | public Boolean cancelBooking(Employee anEmployee, TrainingEvent anEvent)  Post-condition:   1. The database is updated with the cancelation for a booking for anEmployee on anEvent. 2. anEmployee removes the reference to anEvent and anEvent removes the reference to anEmployee. | |
|  | public String toString()  Post-condition: Returns a string describing the purpose of the object. | |

**Package:** utility

|  |  |  |
| --- | --- | --- |
| **Class** | Address | A real-world UK address |
| **Attributes** | | |
|  | private int number  Property number. | |
|  | private String streetName  The name of the street | |
|  | private String city  Name of the city | |
|  | private String postcode  The postcode | |
| **Constructor** | | |
|  | public Address(int aNumber, String aStreetName, String aCity, String aPostcode)  Post-Condition: Initialises a new TrainingEvent object with the given attribute values. | |
| **Protocol** | | |
|  | public int getNumber()  Post-condition: Returns number | |
|  | public String getStreetName()  Post-condition: Returns streetName. | |
|  | public String getCity()  Post-condition: Returns city. | |
|  | public String getPostcode()  Post-condition: Returns postcode. | |
|  | public void setNumber(int aNumber)  Post-condition: sets number to aNumber. | |
|  | public void setStreetName(String aStreetName)  Post-condition: sets streetName to aStreetName. | |
|  | public void setCity(String aCity)  Post-condition: sets city to aCity. | |
|  | public void setPostcode(String aPostcode)  Post-condition: sets postcode to aPostcode. | |
|  | private String validate(String aPostCode)  Post-condition: Returns aPostcode, if aPostcode is a valid postcode, else throws an exception. | |
|  | public String toString()  Post-condition: Returns number, streetName, city, and postcode. | |

|  |  |  |
| --- | --- | --- |
| **Class** | Email | An employee email address |
| **Attributes** | | |
|  | private String eMail  An email address | |
| **Constructor** | | |
|  | public EMail(String anEMail)  Post-condition: a new EMail object is created if the value of the provided attribute is a valid email address, else throws an exception. | |
| **Protocol** | | |
|  | public String geteMail()  Post-condition: Returns eMail. | |
|  | public void seteMail(String anEMail)  Post-condition: sets email to anEmail if the String is a valid email address, else an exception is thrown. | |
|  | private String validate(String anEMail)  Post-condition: Returns anEmail if anEmail is a valid email address, else throws an exception. | |
|  | public String toString()  Post-condition: Returns eMail. | |

|  |  |  |
| --- | --- | --- |
| **Class** | ID | IDs of employees, stores and training departments |
| **Attributes** | | |
|  | private IDClass classifier  A classifier at the beginning of an id that classifies stores (st), people (pn) and training departments (td). | |
|  | private String code  A unique code | |
| **Constructor** | | |
|  | public ID(String aCode)  Post-Condition: Initialises a new ID object if the provided attribute value is a valid ID string, else throws an exception. | |
|  | public ID(IDClass aClassifier, String aCode)  Post-Condition: Initialises a new ID object if the provided attribute value is a valid ID string, else throws an exception. | |
| **Protocol** | | |
|  | private String validate(String aCode)  Post-condition: Returns aCode if aCode is a valid ID, else throws an exception. | |
|  | public String getID()  Post-condition: Returns classifier and code as a concatenated string. | |
|  | public IDClass getClassifier()  Post-condition: Returns classifier. | |
|  | public String getCode()  Post-condition: Returns code. | |
|  | public String toString()  Post-condition: Returns classifier and code. | |

|  |  |  |
| --- | --- | --- |
| **Enum** | IDClass | Different classifiers available for the ID class. |
|  | **Values** | st, pn and td |

|  |  |  |
| --- | --- | --- |
| **Class** | Name | Name of an employee |
| **Attributes** | | |
|  | private Title title  Title of a person such as Mr or Ms. | |
|  | private String firstName  First name of an employee | |
|  | private String lastName  Last name of an employee | |
| **Constructor** | | |
|  | public Name(Title aTitle, String aFirstName, String aLastName)  Post-condition: Initialises a new Name object with the given attribute values. | |
|  | public Name(String aTitle, String aFirstName, String aLastName)  Post-condition: Initialises a new Name object with the given attribute values. | |
| **Protocol** | | |
|  | public Title toTitle(String aTitle)  Post-condition: Returns a title object corresponding to the string if the string corresponds to any title, else returns null. | |
|  | public Title getTitle()  Post-condition: Returns title. | |
|  | public String getFirstName()  Post-condition: Returns firstName. | |
|  | public String getLastName()  Post-condition: Returns lastName. | |
|  | public void setTitle(Title aTitle)  Post-condition: sets title to aTitle. | |
|  | public void setFirstName(String aFirstName)  Post-condition: set firstName to aFirstName. | |
|  | public void setLastName(String aLastName)  Post-condition: sets lastName to aLastName. | |
|  | public String toString()  Post-condition: Returns title, firstName, lastName. | |

|  |  |  |
| --- | --- | --- |
| **Enum** | Title | Title of an employee |
|  | **Values** | Mr, Ms, Mrs, Miss and Mx |

|  |  |  |
| --- | --- | --- |
| **Class** | PhoneNumber | A mobile phone number of an employee |
| **Attributes** | | |
|  | private String mobileNumber  A phone UK mobile phone number | |
| **Constructor** | | |
|  | public PhoneNumber(String aMobileNumber)  Post-condition: a new EMail object is created if the value of the provided attribute is a valid email address, else throws an exception. | |
| **Protocol** | | |
|  | public String getMobileNumber()  Post-condition: Returns title. | |
|  | public void setMobileNumber(String aMobileNumber)  Post-condition: sets mobileNumber to aMobileNumber. | |
|  | private String validate(String aMobileNumber)  Post-condition: Returns aMobileNumber if aMobileNumber is a valid UK mobile phone number, else throws an exception. | |
|  | public String toString()  Post-condition: Returns mobileNumber. | |

#### **Invariants**

1. An Employee object cannot be linked to more than one TrainingEvent object with the same value for the attribute trainingType.
2. An Employee object cannot be linked to more than one TrainingEvent object with the same value for attributes date and time.
3. An Employee object can be linked to a TrainingDepartment object or Store object, but not both.
4. The number of linked Employee objects to a TrainingEvent object cannot be greater than the value of capacity of the linked TrainingDepartment object to the TrainingEvent object.
5. A TrainingDepartment object cannot be linked to more than one TrainingEvent object with the same values for attributes date and time.

### **A3: RESTful API Documentation**

This is a brief description of all the requests that are possible with the API. For convenience I have packaged these calls into groups Employee, Training Event, Training Department, Store, Training and Communication system. These groups have not been randomly chosen, they correspond almost identically to the grouping in the database and the core system, this was done to try and reduce complexity and improve maintainability. Please note this is based on the prototype and therefore does not yet implement any security features, hence the transport protocol is HTTP not HTTPs and no passwords are sent.

|  |
| --- |
| Employee |
| Create a new employee |
| Request method = Post  URL = http://<host>/tms\_rest\_api/api/employees/create.php  Parameters: As a JSON object.  employee\_id, title, firstname, lastname, date\_of\_birth, start\_of\_employment, email\_address, mobile\_number, training\_level, last\_qa, store\_id  Message Returned  {'message' : 'employee Created'}  or  {'message' : 'employee Not Created'} |
| Delete an employee |
| Request method = Post  URL = http://<host>/tms\_rest\_api/api/employees/delete.php  Parameters: As a JSON object.  employee\_id  Message Returned  {'message' : ' employee Deleted'}  or  {'message' : 'employee Not Deleted'} |
| Get all the employees for a given store |
| Request method = Get  URL = http://<host>/tms\_rest\_api/api/employees/get\_store\_employees.php?  Parameters: Appended to the URL.  store\_id  Message Returned  <A JSON Array with all the employees >  or  {'message' : 'No Employees found'}  Example of one item in the array  {"employee\_id":"pn00000008",  "title":"mr",  "firstname":"dave",  "lastname":"smith",  "date\_of\_birth":"1978-07-23",  "start\_of\_employment":"2019-07-12",  "email\_address":"anadress@gmail.com",  "mobile\_number":"075756845",  "training\_level":"js",  "last\_qa":"2018-12-13",  "store\_id":"st00000002"} |
| Get the training progress for a given employee |
| Request method = Get  URL = http://<host>/tms\_rest\_api/api/employees/training\_progress.php?  Parameters: Appended to the URL.  employee\_id  Message Returned  <A JSON object with the Matching Employee>  or  {'message' : 'No Employees found'}  Example of an employee returned  {"employee\_id":"pn00000008",  "training\_level":"js",  "last\_qa":"2018-12-13"} |
| Update employee information |
| Request method = Post  URL = http://<host>/tms\_rest\_api/api/employees/update.php  Parameters: As a JSON object  employee\_id, title, firstname, lastname, date\_of\_birth, start\_of\_employment, email\_address, mobile\_number  Message Returned  {'message' : 'employee Updated'}  or  {'message' : 'employee Not Updated'} |

|  |
| --- |
| Store |
| Get all the information for a store |
| Request method = Get  URL = http://<host>/tms\_rest\_api/api/stores/find\_store.php?  Parameters: Appended to the URL.  store\_id  Message Returned  <A JSON object with the matching store object>  or  {'message' : 'No stores found'}  Example of a store returned  {"store\_id":"st00000002",  "number":"22",  "streetname":"baker street",  "city":"london",  "postcode":"SW12 7RT"} |

|  |
| --- |
| Training |
| Cancel a booking on a training event for an employee. |
| Request method = Post  URL = http://<host>/tms\_rest\_api/api/training/cancel\_booking.php  Parameters: As a JSON object.  employee\_id, training\_event\_id  Message Returned  {'message' : 'training event cancelled'}  or  {'message' : 'training event could not be cancelled'} |
| Make a booking on a training event for an employee. |
| Request method = Post  URL = http://<host>/tms\_rest\_api/api/training/make\_booking.php  Parameters: As a JSON object.  employee\_id, training\_event\_id  Message Returned  {'message' : 'training booked'}  or  {'message' : 'training could not be booked'} |

|  |
| --- |
| Training Department |
| Get all the training departments |
| Request method = Get  URL = http://<host>/tms\_rest\_api/api/trainingdepartments/find\_all\_departments.php?  Parameters: Appended to the URL.  None  Message Returned  <A JSON Array with all the training department objects>  or  {'message' : 'No training Departments found'}  Example of one item in the array returned  {" td\_id":"td00000002",  "number":"22",  "streetname":"baker street",  "city":"london",  "postcode":"SW12 7RT",  "capacity":"20"} |

|  |
| --- |
| Training Events |
| Get all training events for a given training type |
| Request method = Get  URL = http://<host>/tms\_rest\_api/api/trainingevents/find\_training.php?  Parameters: Appended to the URL.  type  Message Returned  <A JSON Array with all the training event objects>  or  {'message' : 'No training Events found'}  Example of one item in the array returned  {" te\_id":"3",  "date\_time":"2019-12-13 09:00:00",  "type":"sa",  "td\_id":"td00000002"} |
| Find all the training events that an employee is booked on. |
| Request method = Get  URL = http://<host>/tms\_rest\_api/api/trainingevents/find\_employee\_training.php?  Parameters: Appended to the URL.  employee\_id  Message Returned  <A JSON Array with all the training event objects>  or  {'message' : 'No training Events found'}  Example of one item in the array returned  {" te\_id":"3",  "date\_time":"2019-12-13 09:00:00",  "type":"sa",  "td\_id":"td00000002"} |

|  |
| --- |
| Communication System |
| Get a list of all the employee that are overdue or are 1 month from being overdue for the Junior Sales Assistant test. |
| Request method = Get  URL = http://<host>/tms\_rest\_api/api/communicationsystem/js.php  Parameters: Appended to the URL.  None  Message Returned  <A JSON Array with all the overdue employees>  or  {'message' : 'No results found'}  Example of one item in the array returned  {"title":"mr",  "firstname":"dave",  "lastname":"smith",  "email\_address":"[anadress@gmail.com](mailto:anadress@gmail.com)",  "mobile\_number":"075756845"} |
| Get a list of all the employee that are overdue or are 1 month from being overdue for the Sales Assistant test. |
| Request method = Get  URL = http://<host>/tms\_rest\_api/api/communicationsystem/sa.php  Parameters: Appended to the URL.  None  Message Returned  <A JSON Array with all the overdue employees>  or  {'message' : 'No results found'}  Example of one item in the array returned  {"title":"mr",  "firstname":"dave",  "lastname":"smith",  "email\_address":"[anadress@gmail.com](mailto:anadress@gmail.com)",  "mobile\_number":"075756845"} |
| Get a list of all the employee that are overdue or are 1 month from being overdue for the Qualified Sales Assistant test. |
| Request method = Get  URL = http://<host>/tms\_rest\_api/api/communicationsystem/qa.php  Parameters: Appended to the URL.  None  Message Returned  <A JSON Array with all the overdue employees>  or  {'message' : 'No results found'}  Example of one item in the array returned  {"title":"mr",  "firstname":"dave",  "lastname":"smith",  "email\_address":"[anadress@gmail.com](mailto:anadress@gmail.com)",  "mobile\_number":"075756845"} |
| Get a list of all the employee that are overdue or are 1 month from being overdue for the Senior Sales Assistant test. |
| Request method = Get  URL = http://<host>/tms\_rest\_api/api/communicationsystem/ss.php  Parameters: Appended to the URL.  None  Message Returned  <A JSON Array with all the overdue employees>  or  {'message' : 'No results found'}  Example of one item in the array returned  {"title":"mr",  "firstname":"dave",  "lastname":"smith",  "email\_address":"[anadress@gmail.com](mailto:anadress@gmail.com)",  "mobile\_number":"075756845"} |

### **A4: Use Case Test Cases**

|  |  |
| --- | --- |
| Test ID | 100: Adding a new employee to the system |
| Use Case ID | UC020: Add Employee |
| Pre-condition | Core system is running and connected to the internet. |
| Test Data | Title = Mr  First name = John  Last name = Swanson  Employee ID = pn12007894  Date of birth = 17/09/1981  Start of employment = 29/05/2019  Email address = [john@mailing.com](mailto:john@mailing.com)  Mobile number = 07675857635 |
| Expected Result | A new employee with the provided tests data is created, added to the running instance and database. |
| Steps | 1. Navigate to Employee sub section Add      1. Enter test data into form and submit |
| Actual Result | A message from the RESTful API is returned informing the user that the employee is added to the system    The running instance is updated with the new employee    The database is updated with the new employee in the employee table (last entry in the table). |
| Status | Passed |

|  |  |
| --- | --- |
| Test ID | 200: Removing an employee from the system. |
| Use Case ID | UC040: Remove Employee |
| Pre-condition | Core system is running and connected to the internet. |
| Test Data | Employee object corresponding to Mr. John Swanson. |
| Expected Result | Mr. John Swanson is removed from the running instance and all data for John Swanson is removed from the database. |
| Steps | 1. Navigate to Employee sub section Remove   C:\Users\Home\Desktop\testing screens\remove_employee.jpg   1. Select the employee to remove |
| Actual Result | A message from the RESTful API is returned informing the user that the employee has been removed from the system.    The employee is removed from the running instances.    All the employee data is removed from the database, shown by the absence of the record corresponding to John Swanson. |
| Status | Passed |

|  |  |  |
| --- | --- | --- |
| Test ID | 300: Update information of an employee. | |
| Use Case ID | UC060: Edit Employee | |
| Pre-condition | Core system is running and connected to the internet. | |
| Test Data | Old Data | New Data |
| Title = Mr  First name = George  Last name = willis  Employee ID = PN00000013  Date of birth = 22/06/1965  Start of employment = 16/03/2010  Email address = george@gmail.com  Mobile number = 07474967555 | Title = Mr  First name = George  **Last name = Willis**  Employee ID = PN00000013  Date of birth = 22/06/1965  Start of employment = 16/03/2010  **Email address = gwillis@mail.com**  Mobile number = 07474967555 |
| Expected Result | The employee’s last name and email address are changed from what is stated in old data to that of the new data. The changes are reflected in the running instance of the core system and database. | |
| Steps | 1. Navigate to Employee sub section Edit      1. Select the employee to edit.      1. Make amendments and submit. | |
| Actual Result | A message from the RESTful API is returned informing the user that the employee has been updated in the database and another message confirming that the running instance has also updated the data of the employee.    The updated employee data is reflected in the running instance    The relevant fields are updated in the database. | |
| Status | Passed | |

|  |  |
| --- | --- |
| Test ID | 400: View a list of all the employees that are employed at the store of the running instance. |
| Use Case ID | UC080: View Employees |
| Pre-condition | Core system is running and connected to the internet. |
| Test Data | None |
| Expected Result | A list containing the following employees should be displayed:   * Mr. Dave Tanner * Mr. Simon Lister * Mr. George Willis * Miss. Dana Donnelly * Mr. Eric Ericsson * Mr. Stan Still * Ms. Erika Stein |
| Steps | 1. Navigate to Employee sub section View |
| Actual Result | A screen with the list of employees is displayed |
| Status | Passed |

|  |  |
| --- | --- |
| Test ID | 500: Book an employee on a training event |
| Use Case ID | UC100: Book training event |
| Pre-condition | Core system is running and connected to the internet. |
| Test Data | An employee to book on an event: Miss. Dana Donnelly.  A Training Event to book the employee on: Sales Assistant training event. |
| Expected Result | * Miss Dana Donnelly is booked on the event * The Sales Associate training event is linked to Miss. Dana Donnelly * Miss. Dana Donnelly is added to the event * Both the running instance and database reflect the above bullet points. |
| Steps | 1. Navigate to Training sub section Book Training      1. Choose the employee Miss. Dana Donnelly, the training type, the event and press book to submit the form. |
| Actual Result | A message from the RESTful API is returned stating that the operation was successful.    The Employee in the running instance has a new training event linked to its object (profile).    The database records the event in the training table.    (pn00000014 is Miss. Dana Donnelly’s ID)  \*Name was not stored in this table due to the normalization process. |
| Status | Passed |

|  |  |
| --- | --- |
| Test ID | 600: Cancel an employee’s booking on an event. |
| Use Case ID | UC200: Cancel training event. |
| Pre-condition | Core system is running and connected to the internet. |
| Test Data | An employee with at least one booking on an event: Miss. Dana Donnelly.  A Training Event to remove from the employee: Sales Assistant training event. |
| Expected Result | * Miss Dana Donnelly is no longer booked on the event * The Sales Associate training event is no longer linked to Miss. Dana Donnelly * Both the running instance and database reflect the above bullet points. |
| Steps | 1. Navigate to Training sub section Cancel Training      1. Choose the employee Miss. Dana Donnelly, the training event to cancel and press cancel to submit the form. |
| Actual Result | A message from the RESTful API is returned stating that the operation was successful.    The Employee in the running instance has the training event linked to its object (profile) removed.    The database removes the event from the training table.    (pn00000014 is Miss. Dana Donnelly’s ID) – not present.  \*Name was not stored in this table due to the normalization process. |
| Status | Passed |

The following test data was created to check boundary values. The reasoning is based on 4 different training levels of

1. junior sales assistant (JSS),
2. sales assistant (SA),
3. qualified sales assistant (QSA) and
4. senior sales assistant (SSA).

However, because a new employee has no training, a 5th option consisting of

1. no training (NT)

must be considered. Then there are the boundaries which are

1. 1 day before the test is overdue (-1),
2. on the same day (0) and
3. one day after (+1),

giving 3 possible values. So, there should be 3 x 5 = 15 tests. As demonstrated by viewing the possible test data as a cartesian product of the boundaries and types of tests.

Training = {JSS, SA, QSA, SSA, NT}

Boundaries = {-1, 0, +1}

Training x Boundaries = {(JSS, -1), (JSS, 0), (JSS, +1), (QSA, -1) …, (NT, +1)}

There is however a problem with this approach regarding the qualified assistant test, as this test require to be tested separately as well, since this requires employees to re-sit the tests annually and a separate value is stored in the database with the date of the last time the test was completed.

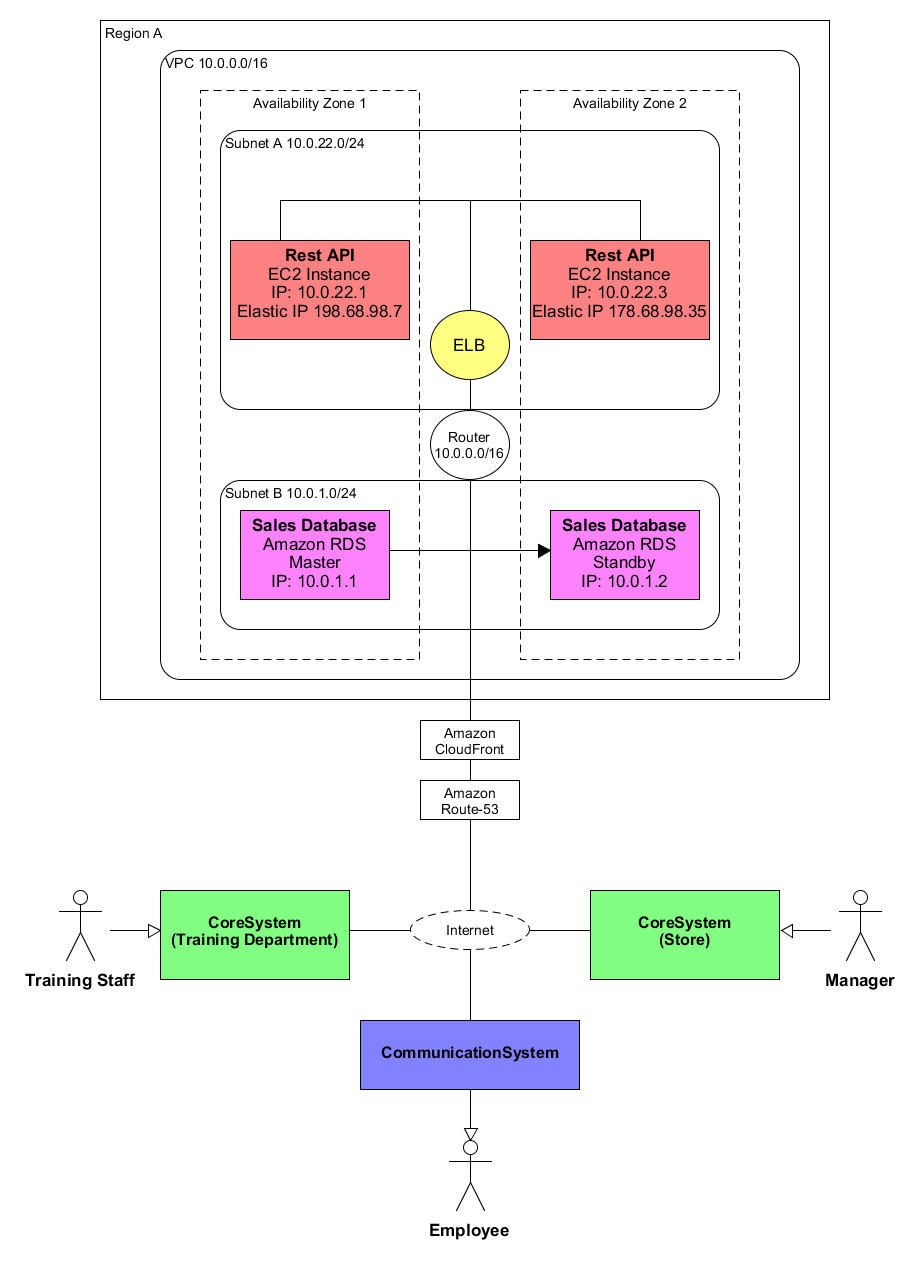
The test which considers the qualified assistant test would require 12 x 3 = 36 tests. This is because for each possible test considered in the previous test there is also the need to test the qualified assistant test with the 3 boundaries of one day before, on the day and one day after the due date, therefore multiplying the test data of the previous test by 3. This can be explained easier using set notation as before.

QSA Test = (Training x QSA) x Boundaries

Note: the cartesian products show the amount of test data required which can be executed in 2 test cases one including the Training x boundaries test data and another including QSA Test test data.

|  |  |
| --- | --- |
| Test ID | 700: View all the employees that are overdue for any test for the store of the running instance. |
| Use Case ID | UC300: View Training Progress. |
| Pre-condition | Core system is running and connected to the internet. |
| Test Data | Employees of the store where the instance is deployed in:  Store ID: st00000001  Test Date: 2019-09-10  **Employees**   |  |  |  | | --- | --- | --- | | Employee | Training level | Start date (Y-m-d) | | Dave Tanner | None | 2019-06-09 | | Simon Lister | None | 2019-06-10 | | George Willis | None | 2019-06-11 | | Dana Donnelly | Junior Sales Assistant | 2019-03-09 | | Eric Ericsson | Junior Sales Assistant | 2019-03-10 | | Stan Still | Junior Sales Assistant | 2019-03-11 | | Erika Stein | Sales Assistant | 2018-09-09 | | Mike Johnson | Sales Assistant | 2018-09-10 | | Mandy Muse | Sales Assistant | 2018-09-11 | | Ken Smith | Qualified Sales Assist. | 2018-03-09 | | Sue O’Conner | Qualified Sales Assist. | 2018-03-10 | | Ed Baker | Qualified Sales Assist. | 2018-03-11 | | Frank Heinz | Senior Sales Assistant | 2018-03-09 | | Norman Averyman | Senior Sales Assistant | 2018-03-10 | | Justin Case | Senior Sales Assistant | 2018-03-11 | |
| Expected Result | * Mr. Dave Tanner is overdue for the Junior Sales Assistant test. * Miss. Dana Donnelly is overdue for the Sales Assistant test. * Ms. Erika Stein is overdue for the Qualified Sales Assistant test. * Mr. Ken Smith is overdue for the Senior Sales Assistant test. |
| Steps | 1. Main screen or Navigate to Training sub section Training Progress. |
| Actual Result | A list of all the employees that are overdue is returned. |
| Status | Passed |

|  |  |
| --- | --- |
| Test ID | 800: Send Reminders to all the employees that are overdue or 1 month away from being overdue on any test. |
| Use Case ID | UC400: Send training reminder. |
| Pre-condition | Communication system is running and connected to the internet. |
| Test Data | Employees of the company:  Store ID: st00000001  Test Date: 2019-09-10  **Employees**   |  |  |  | | --- | --- | --- | | Employee | Training level | Start date (Y-m-d) | | Dave Tanner | None | 2019-07-09 | | Simon Lister | None | 2019-07-10 | | George Willis | None | 2019-07-11 | | Dana Donnelly | Junior Sales Assistant | 2019-04-09 | | Eric Ericsson | Junior Sales Assistant | 2019-04-10 | | Stan Still | Junior Sales Assistant | 2019-04-11 | | Erika Stein | Sales Assistant | 2018-10-09 | | Mike Johnson | Sales Assistant | 2018-10-10 | | Mandy Muse | Sales Assistant | 2018-10-11 | | Ken Smith | Qualified Sales Assist. | 2018-04-09 | | Sue O’Conner | Qualified Sales Assist. | 2018-04-10 | | Ed Baker | Qualified Sales Assist. | 2018-04-11 | | Frank Heinz | Senior Sales Assistant | 2018-04-09 | | Norman Averyman | Senior Sales Assistant | 2018-04-10 | | Justin Case | Senior Sales Assistant | 2018-04-11 | |
| Expected Result | The following employees and the tests they are overdue on with their communication details.   * Mr. Dave Tanner, Junior Sales Assistant, email: davetanner@mymail.com, mobile: 07675463631. * Miss. Dana Donnelly, Sales Assistant, email: dana@mailing.com, mobile: 07678967555. * Ms. Erika Stein, Qualified Sales Assistant, ericasdsww@eee.com, mobile: 07676545454. * Mr. Ken Smith, Senior Sales Assistant, email: ksmith@mail.com, mobile: 07656463951 |
| Steps | * Run the communication system |
| Actual Result | A list of all the employees that are 1 month from being overdue or overdue is returned. |
| Status | Passed |



**Image: Database system deployment model.**

**Brief explanation of the main components in the diagram and what their task would be.**

* **EC2**: Virtual machine to run the RESTful API.
* **Elastic Load Balancer (ELB):** Load balancer also known as Elastic Load Balancer (ELB), reduce the load on one instance by sharing the load between running instances.
* **Amazon RDS:** Database Servers to host the database.
* **Virtual Private Cloud,** network to hold all components, and allows for automatic scaling.
* **Public/Private key encryption.** Used to gain secure access to the instances.

**Note:** I have not attempted to deploy the system on a cloud, nor have I ever deployed anything on Amazons AWS, so this deployment diagram is a prototype based on untested ideas that will most likely need to be refined before they work as intended.

# **5 References**

Armour, F. (2019). *Assistant Professor*. [Online] American University. Available at: <https://www.american.edu/kogod/faculty/farmour.cfm> [Accessed 12 Jul. 2019].

Armour, F and Miller. (2000) *Advanced Use Case Modeling: Vol. 1: Software Systems* [Online]*.* Addison-Wesley’s. Available at: <https://pmt-eu.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=44OPN_ALMA_DS51125543760002316&context=L&vid=44OPN_VU1&search_scope=EVERYTHING&tab=default_tab&lang=en_US>

Bcs.org. (2019). *BCS Code of Conduct* [Online] Available at: <https://www.bcs.org/membership/become-a-member/bcs-code-of-conduct/> [Accessed 13 Sep. 2019].

Edmunds, B. (2016). *Securing PHP Apps*. [Online] Berkeley, CA: Apress. Available at: <https://pmt-eu.hosted.exlibrisgroup.com/permalink/f/gvehrt/TN_springer_s978-1-4842-2120-4_426377> [Accessed 13 Sep. 2019].

Gov.uk. (2019). *Data protection*. [Online] Available at: <https://www.gov.uk/data-protection> [Accessed 11 Apr. 2019].

Johnson, J. (2014) *Designing with the mind in mind: simple guide to understanding user interface design guidelines* [Online]. 2nd ed. Available at: <https://pmt-eu.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=44OPN_ALMA_DS51124076490002316&context=L&vid=44OPN_VU1&search_scope=EVERYTHING&tab=default_tab&lang=en_US>

Juneau, J. (2012) Java 7 recipes : a problem-solution approach [Online]. New York: Apress. Available at: <https://pmt-eu.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=44OPN_ALMA_DS51126105300002316&context=L&vid=44OPN_VU1&search_scope=EVERYTHING&tab=default_tab&lang=en_US>

Oreilly.com. (2019). *Sveta Smirnova.* [Online] Available at: <https://www.oreilly.com/pub/au/5049> [Accessed 12 Jul. 2019].

Rountree, D. and Castrillo, I. (2014) The basics of cloud computing : understanding the fundamentals of cloud computing in theory and practice [Online], Elsevier Science & Technology Books, Available at <https://pmt-eu.hosted.exlibrisgroup.com/permalink/f/h21g24/44OPN_ALMA_DS5187367160002316> [Accessed 22 Aug. 2019].

Service.gov.uk. (2018). *Data Protection Act 2018 Factsheet – Overview.* [Online] Available at: <https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/711162/2018-05-23_Factsheet_1_-_Act_overview.pdf>

Smirnova, S. (2012) *MySQL Troubleshooting* [Online]. 1st edn. O'Reilly Media. Available at: <https://pmt-eu.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=TN_safari_s1_9781449317836&context=PC&vid=44OPN_VU1&search_scope=EVERYTHING&tab=default_tab&lang=en_US>

Taman, M. (2015) *JavaFX Essentials* [Online]. Birmingham: Packt Publishing Ltd. Available at: <https://pmt-eu.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=TN_pq_ebook_centralEBC2081464&context=PC&vid=44OPN_VU1&search_scope=EVERYTHING&tab=default_tab&lang=en_US>

United Kingdom. *Data Protection Act 2018: Queen Elizabeth II. Chapter 12* (2018). N.d., n.p.

Uxmatters.com. (2019). *Jeff Johnson :: UXmatters Authors*. [Online] Available at: <https://www.uxmatters.com/authors/archives/2010/04/jeff_johnson.php> [Accessed 12 Jul. 2019].