The Computing Practical Project

School Timetabler

Analysis

Background to and Identification of Problem

Hitchin Boys’ School is a secondary educational establishment with a sixth form and is part of the Hitchin Sixth Form consortium along with Hitchin Girls’ School and The Priory School. The school provides a secondary level education with the aim to carry out GCSEs, followed by further education towards AS and A2 exams and I am currently studying A2 Computing at Hitchin Boys’ School sixth form and Mrs Lansdown is the head of the sixth form. My client for the timetable is Mrs Lansdown, Head of Sixth Form at Hitchin Boys’ School who constructs timetables for the whole of the school. The problem with the current process is that the timetables are generated with no knowledge of the school layout and teachers often spend a great deal of time travelling between distant classrooms.

Description of the Current System

My client currently uses a third party tool called Nova T6 to construct timetables for the whole of the school and, in conjunction with the other heads of sixth in the consortium, the sixth forms at the 3 schools. Nova T6 works in conjunction with the school management system SIMS, which has an SQL server containing staff and student information which Nova T6 has access to. Nova T6’s automatic timetable construction functionality causes staff and students to travel unnecessary distances between lessons, even though alternative classrooms are available within a small distance. The timetable is initially constructed by Nova T6 then manually adjusted to minimise the distance. In order to improve the automatic timetabling construction functionality, the new system will take in to account the distance between classrooms to reduce the distance travelled by staff over a day.

Identification of the Primary User

My client will be the primary user of the system, but it will be designed to allow for fast tutoring in the usage of the system, to ensure that all future heads of sixth form can continue to use the system. The IT department of school will not use the system directly, but will set up and maintain the system. They will configure the system to the primary user's individual needs, in terms of how the system will find and read the data about the school.

Identification of User Needs and Acceptable Limitations

By having the following interview with the client, I was able to ascertain the needs of the user.

- Me: What would you like the new system to do?

- Client: The current system timetables staff and classes into classrooms based on availability. This works, but leads to staff having to travel between lessons, which no one wants. So, if the new system took distances into account that would be wonderful.

- Me: Ok. Being a system which was created by a big company, SIMS might have restrictions on how I will be able to access the data stored by it. Would it be ok if the data was inputted directly into the new system, but allows for bulk loading from files?

- Client: So long as I can add data quickly, that should be fine.

-Me: How would you like the results displayed?

-Client: Currently, the system displays it as a grid which is easy to read, but if I have to transcribe data, it would be better in a table.

-Me: I will display the data in a table as it is likely data will need to be transcribed What other features would be of benefit to the system?

-Client: Currently, I can limit each member of staff to teach a set number of subjects, and set the subject taught in a classroom.

-Me: That makes it easier actually to timetable. Is there anything else I need to know?

-Client: Nothing at the moment.

Data Sources and Destinations

The Nova T6 platform utilises a Microsoft SQL server to store all data for use within the system and this is updated using the interface provided by the system. Unfortunately, the SIMS system and its database requires a licence to access the database and interface provides by SIMS and this requires a fee to be paid to Capita and a non-free training course in how to use the interface and database. Because of this I will have to create a bespoke data entry interface and database for the new data.

To input data in to the new Timetabler system, the user will be able to use a graphical user interface, or import directly from a CSV file. By using the CSV format, the user can quickly create the data to be imported into the system using a spreadsheet program, such as Microsoft Office Excel. Excel is widely used and the school already supply it to all members of staff. The graphical interface will allow the user to input individual entries into the system.

The configuration file will also describe the types of the data stores the Timetabler supports. The initial implementation will support MariaDB although this could easily be extended to support Microsoft SQL if a license was purchased to allow access to the SIMS database.

The Timetabler system needs a means of understanding the layout of the school and the position of the classrooms within the school buildings. CSV files are a simple and portable way of achieving this goal. By using CSV files for the map, the user can continue to use their desired spreadsheet software to create and visualise the maps.

Map and configuration files will be treated as static data, as these files will rarely change and are very unlikely to need changing at runtime. All other data will be treated as dynamic data, as the data will be able to be modified at runtime and will mostly likely change between runs. As the Timetabler system cannot directly interact with the Nova T6 timetable suite, the final timetable will be displayed in the user interface in a table-based layout for the user to transcribe to the Nova T6 timetable suite.

Data Volumes

The system will need to store and process large amounts of data about the school, such as subject, staff, class, period and layout data. This data will be stored in a relational database, MariaDB, which includes a compression subsystem to reduce the storage space required to store the data, which will be approximately 400MiB in size. Due to the nature of the problem, the size can change depending on the changes at the school, e.g. changes in the number of staff members or subjects taught.

The data will be accessed infrequently, roughly once per year. This allows the data to be stored off the user’s primary computer between runs as it does not need to be accessed as frequently as other data on the computer.

Each distinct collection of data will have varying sizes, for example the subject data will contain approximately 13 entries whereas the lesson data will contain up to 1000 entries. To reduce the size of each entry and increase data integrity, the data will be in third normal form. The configuration file will be a small text based file in the YAML format, which will be approximately 1KiB.

Data Dictionary

The following is the data dictionary of the Nova T6 system.

Staff Data

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Field Purpose** | **Field Type** |
| Code | The unique code for the member of staff, used for identification. | String |
| Name | The name of the staff member. | String |
| Subjects | The subjects taught by the member of staff, in order of speciality. | List of strings |
| Hours | The number of hours the member of staff can teacher per cycle of the timetable. | Integer |
| Preferred Room | The code of the room where the teacher prefers to teach in | String |

Subject Data

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Field Purpose** | **Field Type** |
| Code | The code for the subject, which is not unique. | String |
| Name | The name of the subject. | String |

Period Data

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Field Purpose** | **Field Type** |
| Day | The day of the week the period is in. | Day |
| Number | Each day is split into a defined number of periods, this value represents an individual period. | Integer |

Classroom Data

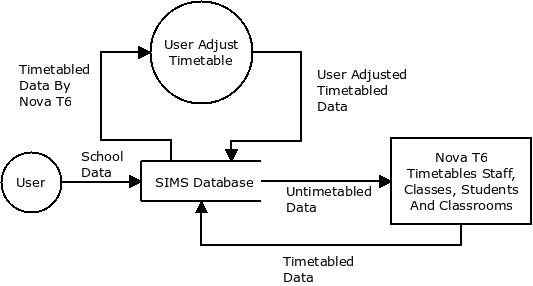
|  |  |  |
| --- | --- | --- |
| **Field Name** | **Field Purpose** | **Field Type** |
| Subjects | The codes of the subjects which can be taught in the room. | List of Strings |
| Room | The name of the classroom. | String |
| Code | The unique identification code for the classroom. | String |
| Site | The code of the site the classroom is in. | String |

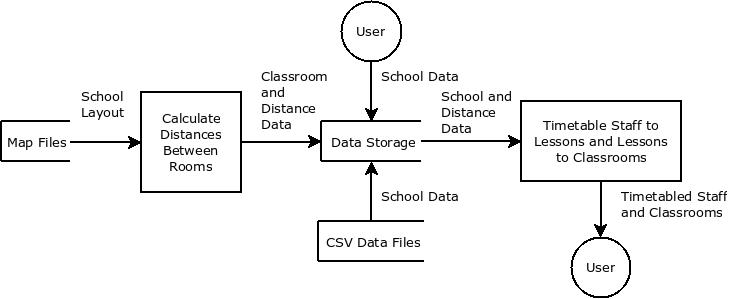
Site Data

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Field Purpose** | **Field Type** |
| Code | The unique identification code for the site. | String |
| Description | The name of the site. | String |
| Year Group Preferences | The preferences towards year groups in the site. Defines which year groups can be taught at the site. | List of Strings |

Dataflow Diagram

Current System



Proposed System

Entity Relationship Model



Within the new Timetabler system each member of staff can be teaching many lessons, but each lesson can have only one member of staff teaching it. Each classroom can have many lessons taught within it, but each lesson can only have a single classroom for it to be taught in.

**Staff(**staffId, staffName, subjectName, hoursPerWeek)

**Classroom**(id, roomName, buildingName, subjectName)

**Lesson(**lessonId, dayOfWeek, startTime, endTime, setName, yearName, subjectName, staffId, classroomId)

Object Orientation planning

| **Class Name** | **Inheritance** | **Overriding** | **Description** |
| --- | --- | --- | --- |
| Main | None | None | The entry point to the program, which binds the code base together. |
| ConfigParser | None | None | A way to parse a configuration file into usable objects within the language. This should not be a concrete class, and should be used as a super class to all configuration parsers |
| ConfigType | None | None | The file of the configuration file to be parse by an implementation ConfigParser. |
| CellType | None | None | A type of cell which can appear in the map of the school. This should not be a concrete class, and should be used as a super class to all cells to appear in the map of the school. |
| ImportantCell | CellType | isTraversable | A cell which is used for the distance calculations between classrooms, such as classrooms and building entrances. This should not be a concrete class, and should be used as a super class to all cells need in the distance calculations. |
| BuildingCell | ImportantCell | getDistances  isTraversable | A cell within a high level school map which references a building map. It should be able to calculate the distances between all ImportantCells within its building map. |
| ClassroomCell | ImportantCell | getDistances  isTraversable | A cell that represents a classroom. Each classroom must have a unique name to avoid conflicts. There must only be a single traversable cell adjacent to a classroom to avoid doubling the distance calculations. |
| Path | CellType | isTraversable | A cell that represents an area that can be traversed and links important cells together. |
| Wall | CellType | isTraversable | A cell that represents an area which cannot be traversed and fills the empty space in the map. |
| SchoolMap | None | None | A wrapper class which wraps the map of the school, or a building, and provides utility methods to access the map. |
| Walker | None | None | A class which 'walks' between two coordinates on a given map in order to find the shortest distance, between them using a recursive method. |
| Dao | None | None | The Data Access Object (DAO) encapsulates the mechanism for the program to interact with a data type from a data source. This should not be a concrete class, and should be used as a super class to all data access object implementations. |
| MutableDao | Dao | None | The way for the program to interact with a data type from a data source, where the data can be changed. This should not be a concrete class, and should be used as a super class to all data access objects with mutable data. |
| DaoManager | None | None | The manager of all the DAOs for a specific data source. This should not be a concrete class, and should be used as a super class to all data access object manager implementations, each for a different data source. |
| DayDao | Dao | getAll  getById | The way for the program to interact with day data from a data source. This should not be a concrete class, and should be used as a super class to all data access objects which interacts with day data. |
| PeriodDao | Dao | getAll  getById | The way for the program to interact with period data from a data source. This should not be a concrete class, and should be used as a super class to all data access objects which interacts with period data. |
| BuildingDao | MutableDao | getAll  getById  insert  update  delete  loadFile | The way for the program to manipulate building data from a data source. This should not be a concrete class, and should be used as a super class to all data access objects which interacts with building data. |
| ClassroomDao | MutableDao | getAll  getById  insert  update  delete  loadFile | The way for the program to manipulate classroom data from a data source. This should not be a concrete class, and should be used as a super class to all data access objects which interacts with classroom data. |
| DistanceDao | MutableDao | getAll  getById  insert  update  delete  loadFile | The way for the program to manipulate distance data from a data source. This should not be a concrete class, and should be used as a super class to all data access objects which interacts with distance data. |
| LearningSetDao | MutableDao | getAll  getById  insert  update  delete  loadFile | The way for the program to manipulate learning set data from a data source. This should not be a concrete class, and should be used as a super class to all data access objects which interacts with learning set data. |
| LessonPlanDao | MutableDao | getAll  getById  insert  update  delete  loadFile | The way for the program to manipulate lesson plan data from a data source. This should not be a concrete class, and should be used as a super class to all data access objects which interacts with lesson plan data. |
| SchoolYearDao | MutableDao | getAll  getById  insert  update  delete  loadFile | The way for the program to manipulate school year data from a data source. This should not be a concrete class, and should be used as a super class to all data access objects which interacts with school year data. |
| StaffDao | MutableDao | getAll  getById  insert  update  delete  loadFile | The way for the program to manipulate staff data from a data source. This should not be a concrete class, and should be used as a super class to all data access objects which interacts with staff data. |
| SubjectDao | MutableDao | getAll  getById  insert  update  delete  loadFile | The way for the program to manipulate subject data from a data source. This should not be a concrete class, and should be used as a super class to all data access objects which interacts with subject data. |
| SubjectSetDao | MutableDao | getAll  getById  insert  update  delete  loadFile | The way for the program to manipulate subject set data from a data source. This should not be a concrete class, and should be used as a super class to all data access objects which interacts with subject set data. |
| DataType | None | None | A type of data stored by the system. This should not be a concrete class, and should be used as a super class to all objects which represent data in a data source. |
| Building | DataType | None | A single building at the school. |
| Classroom | DataType | None | A single classroom at the school. |
| Day | DataType | None | A single working day. |
| Distance | DataType | None | A single distance between two classrooms. |
| LearningSet | DataType | None | A single set taught at the school. |
| LessonPlan | DataType | None | A single lesson which has an allocated member of staff, classroom, period and SubjectSet. |
| Period | DataType | None | A single period when a lesson can be taught. |
| SchoolYear | DataType | None | A single year group taught at the school. |
| Subject | DataType | None | A class which stores data about a single subject taught at the school. |
| Staff | DataType | None | A class which stores data about a single member of staff at the school. |
| SubjectSet | DataType | None | A single ‘class’ taught at the school. |

The following describes the class members and methods for each class in the Timetabler.

**Main Class**

Class members

| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| --- | --- | --- | --- | --- |
| Private | configType | ConfigType | YAML | The data type of the configuration file, by default it is YAML. This can be overridden by a command line argument. |
| Private | daoManager | DaoManager | Null | The DaoManager to be used throughout the system. |

Class methods

| **Access Type** | **Method Name** | **Parameters** | **Return Values** | **Description** |
| --- | --- | --- | --- | --- |
| Public | main | Command line arguments | None | The entry point of the program, which handles the command line arguments. |
| Public | Init | None | None | Initialises the program. Parses the configuration file, data files and map files. |
| Public | Start | None | None | The main body of the program. It contains the user interface and event handlers. |
| Public | Stop | None | None | Stops the program. Exports the database back to the data files and destroys the objects still alive. |

**ConfigParser Class**

Class methods

| **Access Type** | **Method Name** | **Parameters** | **Return Values** | **Description** |
| --- | --- | --- | --- | --- |
| Public | Parse | None | Map of string/string key/value entries | Parses a configuration file into a map containing the key/value entries from the file. |
| Public | getParser | The config type and the file path | An implementation of ConfigParser | Gets the correct implementation of ConfigParser for the given ConfigType if it exists, giving it the given FilePath. |

**Config Type**

Class members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Public | YAML | ConfigType | YAML | An enum-style value which represents a YAML configuration file type. |

**CellType Class**

Class methods

| **Access Type** | **Method Name** | **Parameters** | **Return Values** | **Description** |
| --- | --- | --- | --- | --- |
| Public | isTraversable | None | Boolean | Returns true if a cell is traversable |

**Important Cell Class**

Class methods

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Method Name** | **Parameters** | **Return Values** | **Description** |
| Public | getDistances | None | A map where the key is a cell | Returns a map containing the distances between this cell and other important cells |
| Public | isTraversable | None | Boolean | Returns true if a cell is traversable |

**Building Cell Class**

Class members

| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| --- | --- | --- | --- | --- |
| Private | important | List of important cells | Null | A list of the important cells within this building |
| Public | Name | String | Null | The name of the building, which must be globally unique |

Class methods

| **Access Type** | **Method Name** | **Parameters** | **Return Values** | **Description** |
| --- | --- | --- | --- | --- |
| Public | getDistances | None | A map where the key is a cell | Returns a map containing the distances between this cell and other important cells within the same map |
| Public | isTraversable | None | Boolean | Always returns true |

**Classroom Cell Class**

Class members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Public | Number | String | Null | The unique classroom number |
| Private | Distances | Map of distance, where the key is the cell | Null | A map of the distances between this cell and the other cells in the same map file |

Class methods

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Method Name** | **Parameters** | **Return Values** | **Description** |
| Public | getDistances | None | A map where the key is a cell and the value is the distance | Returns a map containing the distances between this cell and other important cells in the same map |
| Public | isTraversable | None | Boolean | Always returns true |

**Path**

Class methods

| **Access Type** | **Method Name** | **Parameters** | **Return Values** | **Description** |
| --- | --- | --- | --- | --- |
| Public | isTraversable | None | True | Returns true as is a traversable cell |

**Wall**

Class methods

| **Access Type** | **Method Name** | **Parameters** | **Return Values** | **Description** |
| --- | --- | --- | --- | --- |
| Public | isTraversable | None | False | Returns false as is not a traversable cell |

**SchoolMap**

Class members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Private | Width | Integer | 0 | The width of the map |
| Private | Height | Integer | 0 | The height of the map |
| Private | schoolGrid | 2D array of CellType | Contains only walls | The map of the school represented in an ‘easy to populate and use’ form |

Class methods

| **Access Type** | **Method Name** | **Parameters** | **Return Values** | **Description** |
| --- | --- | --- | --- | --- |
| Public | getCell | X and Y coordinates | The cell at the given coordinates | Gets the cell at the given coordinates |
| Public | getCoordinates | Cell | The coordinates of the given cell | Gets the coordinates of the given cell |
| Public | getRoomCoordinates | Room name | The coordinates of the given room | Gets the coordinates of the given room |
| Public | getAllClassrooms | None | A list of the all the classrooms | Gets a list of the classrooms in this map |
| Public | getAllBuildings | None | A list of all the buildings | Gets a list of all the buildings in this map |
| Public | getAllImportantCells | None |  | Gets a list of the buildings in this map |

**Walker**

Class members

| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| --- | --- | --- | --- | --- |
| Private | schoolMap | A 2D array of CellType | All walls | A copy of the map to be 'walked' to calculate the distances for |
| Private | finalDistance | Integer | Maximum possible value for integer | The final shortest distance between two cells. It must the maximum possible integer as the walk move method checks if its value for distance is smaller than finalDistance |

Class methods

| **Access Type** | **Method Name** | **Parameters** | **Return Values** | **Description** |
| --- | --- | --- | --- | --- |
| Public | setMap | A map to be used for distance calculations | None | Sets the variable schoolMap to the given map |
| Public | Walk | Two coordinates, the first is the initial coordinate and the second is the final coordinate | The shortest distance between the two given coordinates | Traverses the map to find the shortest distance between the two coordinates by only travelling in traversable cells. It must call move, and reset the finalDistance variable before calling move. |
| Private | move | Three coordinates; current, final and previous coordinates; and the current distance travelled | The shortest distance between the initial coordinate specified in walk, to the final coordinate | A recursive method which determines if the destination cell is adjacent to the current cell. Once the destination cell is found it checks if finalDistance is higher than the current distance travelled. If so then set finalDistance to current distance travelled.  Then winds up the stack looking for another route.  If the destination cell is not adjacent to the current cell, then it moves to the next adjacent traversable cell, checking north, south, east and west in that order, not moving to the cell if it was the previously traversed cell.  Once walker begins to wind up the stack, it will stop and try an alternative route at every point available, checking if it is a shorter route or even a possible route. |

**Building**

Class members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Public | id | Integer | -1 | The unique identification number for this building. |
| Public | buildingName | String | Empty String | The name of this building. |

**Classroom**

Class members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Public | id | Integer | -1 | The unique identification number for this classroom. |
| Public | classroomName | String | Empty String | The name of this classroom. |
| Public | subject | Subject | -1 | The subject taught in this classroom. |

**Day**

Class members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Public | id | Integer | -1 | The unique identification number for this day. |
| Public | dayName | String | Empty String | The name of this day. |

**Distance**

Class members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Public | id | Integer | -1 | The unique identification number for this distance. |
| Public | startRoom | Classroom | Null | The room where the distance is calculated from. |
| Public | endRoom | Classroom | Null | The room where the distance is calculated to. |
| Public | distance | Integer | -1 | The distance between the two classrooms. |

**LearningSet**

Class members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Public | id | Integer | -1 | The unique identification number for this set. |
| Public | setName | String | Empty String | The name of this set. |

**LessonPlan**

Class members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Public | id | Integer | -1 | The unique identification number for this day. |
| Public | staff | Staff | Null | The member of staff teaching this lesson. |
| Public | classroom | Classroom | Null | The classroom the lesson is being taught in. |
| Public | period | Period | Null | The period the lesson is being taught in. |
| Public | subjectSet | SubjectSet | Null | The subjectSet being taught in the lesson. |

**Period**

Class members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Public | id | Integer | -1 | The unique identification number for this day. |
| Public | day | Day | Null | The day the period is in. |
| Public | startTime | Time | Null | The time the period starts at. |
| Public | endTime | Time | Null | The time the period ends at. |

**SchoolYear**

Class members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Public | id | Integer | -1 | The unique identification number for this year group. |
| Public | yearName | String | Empty String | The name of this year group. |

**Staff**

Class members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Public | name | String | Null | The name of this member of staff |
| Public | id | String | Null | The unique identifier for this member of staff |
| Public | subjectId | Integer | -1 | The unique identifier for the subject taught by the staff member. |

**Subject**

Class members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Public | id | Integer | -1 | The unique identification number for this subject. |
| Public | subjectName | String | Empty String | The name of this subject. |

**SubjectSet**

Class members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Public | id | Integer | -1 | The unique identification number for this subject set. |
| Public | subject | Subject | Null | The subject taught in this subject set. |
| Public | learningSet | LearningSet | Null | The set being taught in this subject set. |
| Public | schoolYear | SchoolYear | Null | The year group being taught in this subject set. |

Objectives

|  |  |
| --- | --- |
| **Objective Number** | **Details** |
| **1** | The system must be able to parse a map of the school and calculate the distances between each classroom to every other classroom. The map does not need to be kept in memory after the distances are calculated as the distances are the only information which is required from the map. |
| **2** | The system must be able to parse files containing information about the school, such as subject, staff and class data. This data should be stored in a database rather than in memory. |
| **3** | The system must allow the user to modify the school data at runtime to allow the user to perform validation on the data within the system before any processing is done to the data. |
| **4** | The system must allow the user to input a predetermined lesson timetable where the lessons are already timetabled. This must be done using a table-based interface. |
| **5** | The system must allow the user to specify the file type of the school data, the location of the map files, and the top map at runtime. |
| **6** | The system must display the fully timetabled in a table-based layout for the user to quickly transcribe the timetabled lessons. |

Potential Solutions

| **Suggested Solutions** | **Positives** | **Negatives** |
| --- | --- | --- |
| Visual Basic.NET application with an Access Database Backend | VB.NET allows an interface with the Microsoft Office Suite, which is designed to be used with VB.NET and C#. This makes the interface simple to use and is designed with the languages strengths in mind. | I would have been required to take an addition course in VB.NET to undertake the project as it was not covered at AS level. The system would be confined to Microsoft Windows and Microsoft Office. This could limit the life span of the system. If the user migrates from Windows or Office, the system would no longer be functional. The system would also be confined to an offline system without heavy alterations to the code base and possibly the inclusion of another separate system to execute the system in an online form. |
| Java Swing Application with a MariaDB Backend | I already have a grounding in the Java programming language, its standard library and object orientated programming. This means I do not have to learn a new language or programming concepts before undertaking the project.  MariaDB is a relational database system forked from MySQL, and adjusted to have increased scalability and performance in system critical situations. By using MariaDB, I have been able to access to a relational database via SQL using the official JDBC (Java Database Connectivity) driver.  Also, Java also allows the design of a system without being confined to any operating system. In order to transfer the system, it would only require a change of the version of MariaDB used, as it is a natively compiled system. | By using a language which runs within a virtual machine, the system will have increased runtime requirements compared to a natively compiled language.  The system would need to be confined to being an offline system, unless the system used HTML5 within JavaFX to create the user interface. This means the user interface requires minor alterations to allow for the system to be transferred to an online solution.  Due to the way the language is designed, a large part of the initial code base will be 'boilerplate' code, which only serves to make the further code easier to write. |
| HTML5, Bootstrap, PHP and MariaDB | The system will be access using a web browser, this will allow the system to be accessed without direct access the machine running the system. By using HTML5 and Bootstrap, any device can use the system without the code base changing as Bootstrap modifies the HTML5 'on-the-air' to accommodate the user's requirements.  PHP allows direct modification of the interface on a per-request basis, allowing the interface to be modified based on the database using the SQL functions within the language.  As stated above, the use of MariaDB allows the system to be scalable and responsive when under heavy load, and stable when used in the system critical situation it will be under. | The system will only be accessible by using a web browser, which can be resource intensive in comparison to a bespoke system to accomplish the same task.  Object orientated programming in PHP is not the recommended approach as it was not initially designed to be object orientated, which can be evident in the design of the standard library and language. |

Chosen Solution

After looking into alternative solutions, I have decided to use the Java programming language, using MariaDB as the SQL-based relational database system and JavaFX as the container for user interface designed with HTML5 and Bootstrap 3. I have chosen this due to its scalability and use of high performance technologies. With a firm grounding with the Java programming language, the JavaFX mark-up language and HTML5 coupled with Bootstrap 3, I will leverage my existing skills and knowledge when undertaking the project. This will help reduce the risk of not meeting the final delivery requirements. A VB.NET based system where I would have limited experience could lead to delays due to lack of experience.

By using JavaFX with HTML5 and Bootstrap 3, the system will use current technologies to achieve the user interface, which will result in better performance and overall aesthetic, compared to using Java Swing which is now no longer actively maintained and should not be used if JavaFX is available.

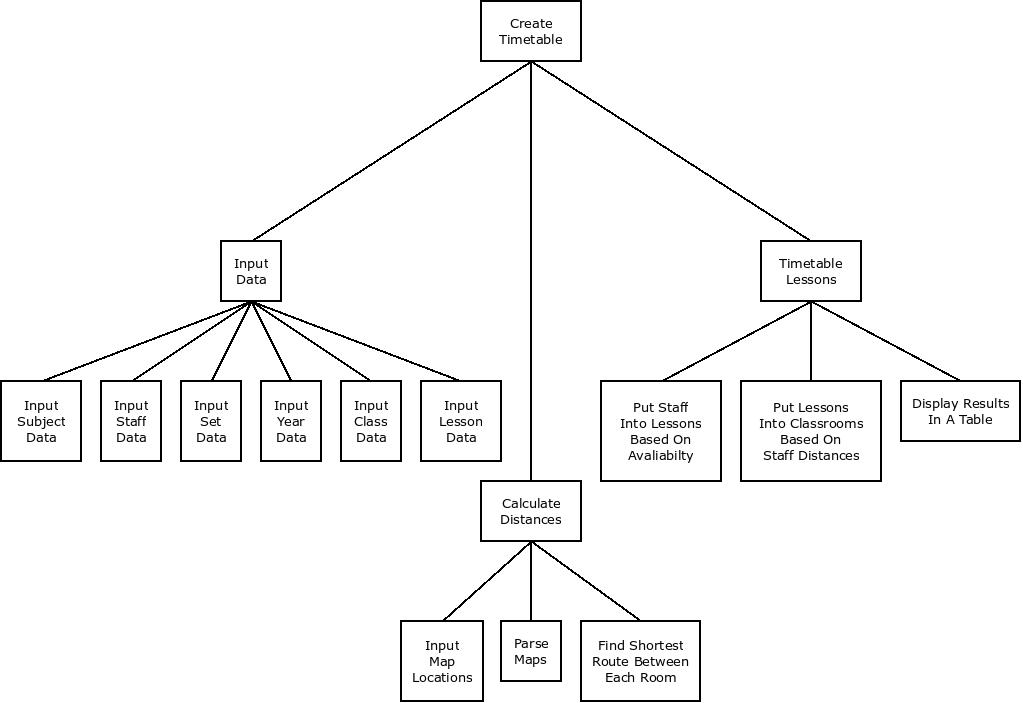
Design

Overall System Design

|  |  |
| --- | --- |
| **Input** | **Processes** |
| Staff data  Name  Subject speciality  Max number of contact hours per week  Subject list  Layout of buildings and classrooms  Building list  Classroom data  Name  Subject taught in the room  Set numbers  Year list  Timetabled lessons without staff | Calculate distances between all classrooms to all other classrooms  Create optimal path between lessons reducing distance travelled, most likely will involve not moving staff between lessons  Add any data to the database through the graphical user interface  Remove any data from the database through the graphical user interface |

|  |  |
| --- | --- |
| **Storage** | **Output** |
| Staff data  Subject data  Building data  Classroom data  Distances between classrooms  Year data  Set data  Timetabled lessons | Staff data  Subject data  Building data  Classroom data  Year data  Set data  Timetabled lessons without staff  Timetabled lessons with staff |

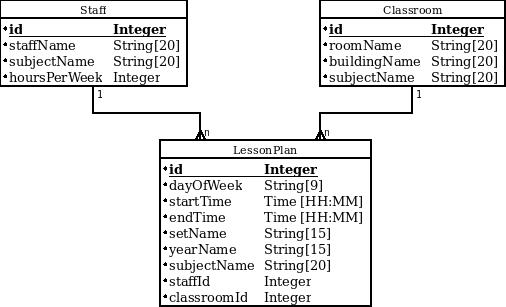
Modular Design



Database Design including Normalised Diagrams

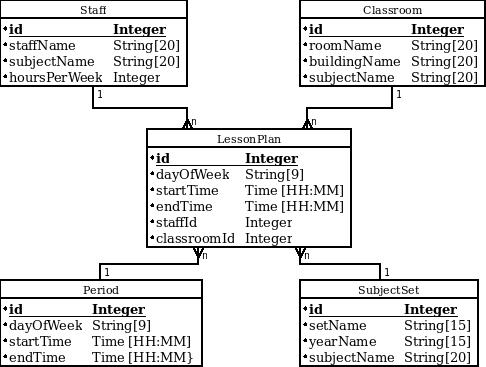
In order to have an optimal and efficient database structure, it must undergo the normalisation process. This will remove redundant data and form atomic data structures for each entity.

First Normal Form

To normalise a database into the first normal form, one must ensure there is no repeating data, each related set of data has its own table, and give each table a primary key. This was done by keeping staff and classroom data in their own tables, each with their own primary key.

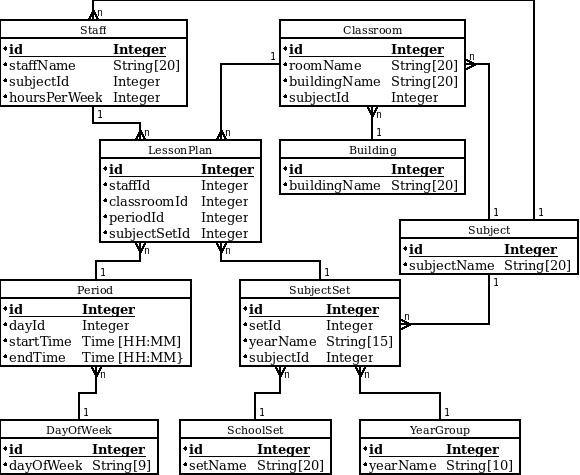
Second Normal Form

To normalise a database into the second normal form, one must ensure the requirements of the first normal form are met, and the data in each table must have at least a partial relationship with the primary key. I achieved this by modifying my first normal form database, distinguishing period and subjectSet data into their own tables.



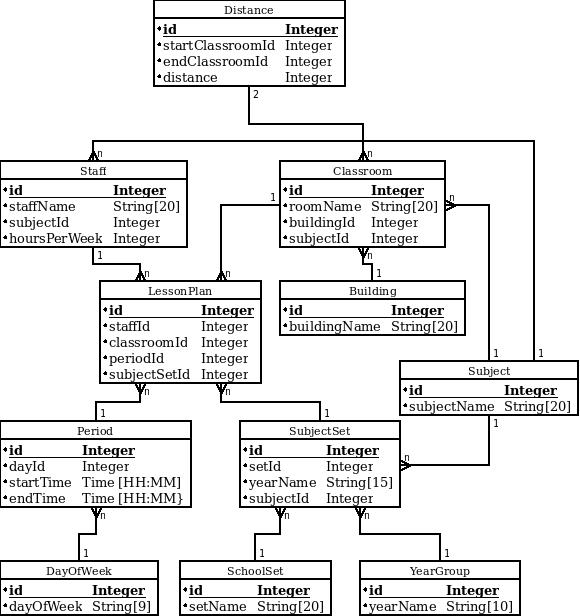
Third Normal Form

For a database to be in the third normal form, one must ensure the database meets the requirements of the second normal form, and all data in a table must be fully dependent on its primary key. I achieved this by distinguishing subject, schoolSet, yearGroup and dayOfWeek data into their own tables.



With All Data and Third Normal Form

The following diagram shows the database in the third normal form, as shown above, with the remaining tables to store data for processing. In this case, I added the distance table to store the distances between two classrooms.



Data Dictionary and Validation

The following contains all the data which will be stored by the database and loaded by the system. Not all data listed will be entered by the user. The distance, classroom and building tables will be populated with data parsed from the map files and calculated by the distance calculation system, while the day and period data will be created when the database is created and will be immutable. Also, all ids will be automatically generated by the database, but in order to allow the user to create entries with dependencies, such as a member of staff, the user interface will have a menu with more useful information, such as the subject name, for the user to select. The system will then carry out a lookup to find the correct id to use.

Subject Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Data Type** | **Size** | **Validation Check** | **Validation Description** | **Valid Data** | **Erroneous Data** |
| ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence | Each subject must have an id, which is unique in the table | 152 | -52 |
| Subject Name | String | 20 characters | Presence, Length | Each subject must have a name, and they are mostly less than 20 characters. | Maths | (Blank) |

Building Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Data Type** | **Size** | **Validation Check** | **Validation Description** | **Valid Data** | **Erroneous Data** |
| ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence | Each building must have an id, which is unique in the table | 1901 | -824 |
| Building Name | String | 20 characters | Presence, Length | Each building must have a name, and they are mostly less than 20 characters. | Science Labs | (Blank) |

Year Group Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Data Type** | **Size** | **Validation Check** | **Validation Description** | **Valid Data** | **Erroneous Data** |
| ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence | Each year group must have an id, which is unique in the table | 42 | -69 |
| Year Group Name | String | 10 characters | Presence, Length | Each building must have a name | Year 7 | (Blank) |

School Set Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Data Type** | **Size** | **Validation Check** | **Validation Description** | **Valid Data** | **Erroneous Data** |
| ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence | Each year group must have an id, which is unique in the table | 82 | -49 |
| Set Name | String | 20 characters | Presence, Length | Each building must have a name, and are mostly much less than 20 characters | Set 4 | (Blank) |

Day Of Week Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Data Type** | **Size** | **Validation Check** | **Validation Description** | **Valid Data** | **Erroneous Data** |
| ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence | Each day must have an id, which is unique in the table | 447 | -321 |
| Day Name | String | 10 characters | Presence, Length | Each day must have a name, and are normally very short | Monday | (Blank) |

Distance Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Data Type** | **Size** | **Validation Check** | **Validation Description** | **Valid Data** | **Erroneous Data** |
| ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence, Unique | Each distance must have an id, which is unique in the table | 95 | -75 |
| Start Classroom ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence, Lookup (Classroom table’s IDs) | Each distance must have a starting classroom, which is a valid classroom | 24 | -93 |
| End Classroom ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence, Lookup (Classroom table’s IDs) | Each distance must have a ending classroom, which is a valid classroom | 87 | -312 |
| Distance | Unsigned Integer | Maximum Size for Unsigned Integer | Presence | Each distance must have a distance value | 15 | -427 |

Classroom Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Data Type** | **Size** | **Validation Check** | **Validation Description** | **Valid Data** | **Erroneous Data** |
| ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence, Unique | Each classroom must have an id, which is unique in the table | 52 | -35 |
| Room Name | String | 20 characters | Presence | Each classroom must have a name | Room 14 | (Blank) |
| Building ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence | Each classroom must be in a valid building | 21 | 85 |
| Subject ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence | Each classroom must have a valid subject to be taught within it | 73 | 645 |

Subject Set Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Data Type** | **Size** | **Validation Check** | **Validation Description** | **Valid Data** | **Erroneous Data** |
| ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence, Unique | Each subject set must have an id, which is unique in the table | 26 | -3 |
| Subject ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence, Lookup (Subject Tables’s IDs) | Each subject set will teach a subject | 860 | -71 |
| Set ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence, Lookup (SchoolSet Table’s IDs) | Each subject set will teach a group of students in a set | 684 | -15 |
| Year Group ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence, Lookup (School Year Table’s IDs) | Each subject set will teach a group of students from a year group | 32 | -81 |

Lesson Plan Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Data Type** | **Size** | **Validation Check** | **Validation Description** | **Valid Data** | **Erroneous Data** |
| ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence, Unique | Each lesson plan must have an id, which is unique in the table | 9 | -57 |
| Staff ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence, Lookup (Staff Table’s IDs) | Each lesson plan must have a valid teacher to teach it | 65 | -5 |
| Classroom ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence, Lookup (Classroom Table’s IDs) | Each lesson plan must have a valid classroom to teach in | 37 | -6 |
| Subject Set ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence, Lookup (Subject Set Table’s IDs) | Each lesson plan must have a valid subject set to teach | 95 | -7 |
| Period ID | Unsigned Integer | Maximum Size for Unsigned Integer | Presence, Lookup (Period Table’s IDs) | Each lesson plan must have a valid period to teach in | 3 | -95 |

**Configuration File**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Data Type** | **Size** | **Validation Check** | **Validation Description** | **Valid Data** | **Erroneous Data** |
| Data Source Type | String | Indefinite | Presence, Lookup (Valid DAO Manager implementation | There must be a valid implementation of a data source’s DAO collection | MARIADB | (Blank) |
| Address | String | Indefinite | Presence | There must be a valid way for the system to find the data source | 127.0.0.1 | (Blank) |
| Port | Integer | Indefinite | Data Type | If the port is required, the port must be an integer | 3306 | PORT |
| Database | String | Indefinite | Data Type | If the database name, the name must be a string | school | (Blank) |
| Username | String | Indefinite | Data Type | If the system needs an username to access the data source, it must be a string | root | (Blank) |
| Exec | File Path | Indefinite | Data Type | If the system needs to start a process for the data source, it must be a valid file path | /usr /bin/ mysqld | …….\/\/\/ |
| Args | List of Strings | Indefinite | Data Type, Form | If ‘exec’ is defined, the system may need arguments for the executable, which must be a list of strings starting with ‘-’ | --no-defaults --basedir =/usr/bin | arguments |

Sample of Planned SQL Queries

In order for the user interface to display the list of available subjects at the school, the system will query the database for the entries in the subject table. The id value will not be displayed but must be queried as it is used to identify the subject quickly, unlike the name of the subject will take longer to identify the subject with.

SELECT id, subjectName FROM subject;

The system will be able to modify the database at playtime through a form in the user interface. In order to add a new member of staff at playtime, the user will fill out the form and the information will be used, in conjunction with the following SQL statement to add the member of staff to the database.

INSERT INTO staff (id, staffName, subjectID) values (?,?,?);

For the system to be able to update database entries, the system will use the following SQL statement to update an entry in the class table. The same form will be used to add a new entry to the database and edit an entry in the database.

UPDATE subjectSet SET subjectId=?,setId=?,yearId=? WHERE id=?;

Identification of Storage Media

The total program is approximately 300MB in size, with approximately 280MB being the database only. With this in mind, the program will be distributed using a DVD as it can be installed easily from the media, and it allows an ‘autoplay’ program to be used for the installation process. The ‘autoplay’ feature allows for the program to be installed easily by the user without expatiation on which executable installs the program. This feature is only present on Windows based machines, which does restrict the initial release of the software, but the system could be distributed using the Internet. This would allow the user to install a version of the program for their operating system, without an extra install disk being deployed. Once the user installs the program, it will be stored on a hard drive as this is currently the most common storage medium for the systems the user will have access to. The program will play without the installation disk, but will require the dependencies installed with the program.

Identification of Algorithms

Map Walker

distance ← MAX\_INTEGER

PROCEDURE walk(currCoord, finalCoord, lastCoord, currDist)

currDist ← currDist + 1

IF currCoord.adjecentTo(lastCoord) THEN

distance = currDist

RETURN

ENDIF

IF currCoord.north != lastCoord AND currCoord.northCoord. IsTraversable THEN

walk(currCoord.north, finalCoord, currCoord, currDist)

ELSE IF currCoord.east != lastCoord AND currCoord .east.IsTraversable THEN

walk(currCoord.east, finalCoord, currCoord, currDist)

ELSE IF currCoord.south != lastCoord AND currCoord .south.IsTraversable THEN

walk(currCoord.south, finalCoord, currCoord, currDist)

ELSE IF currCoord.west != lastCoord AND currCoord .west.IsTraversable THEN

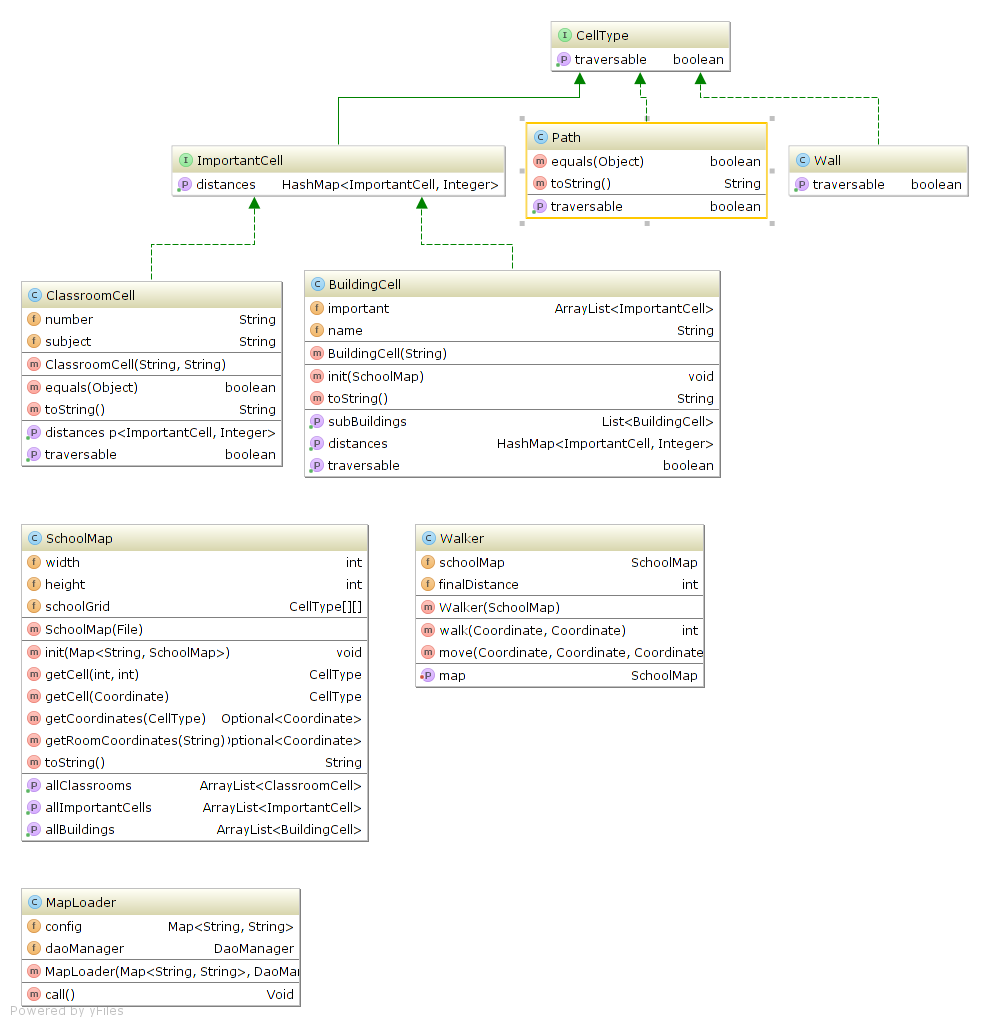
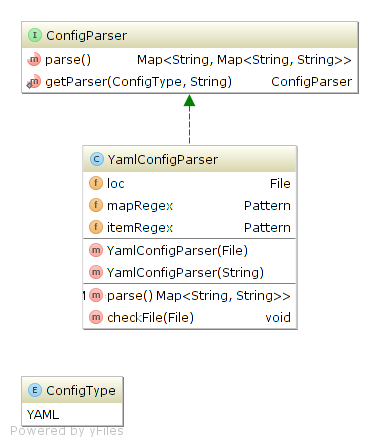
walk(currCoord.west, finalCoord, currCoord, currDist)

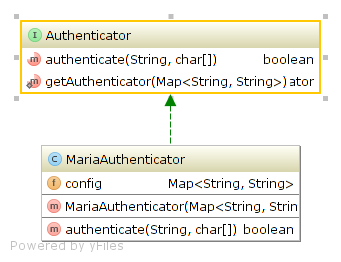
ENDFUNCTION

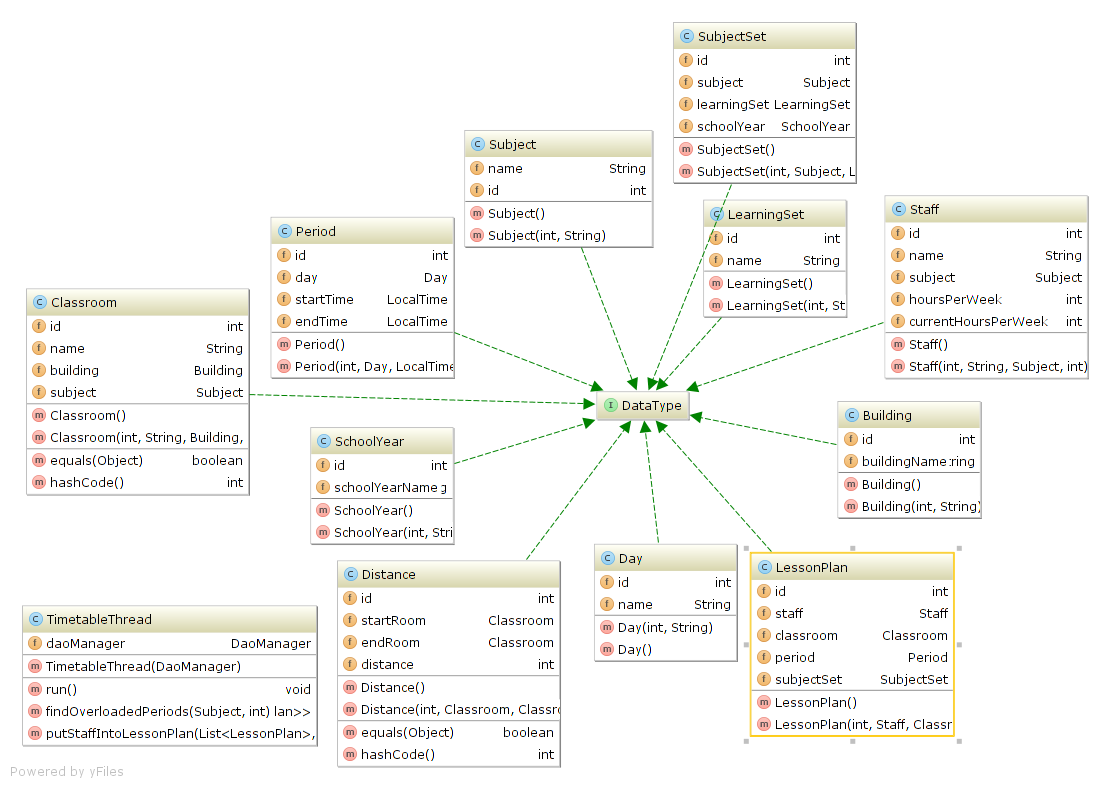
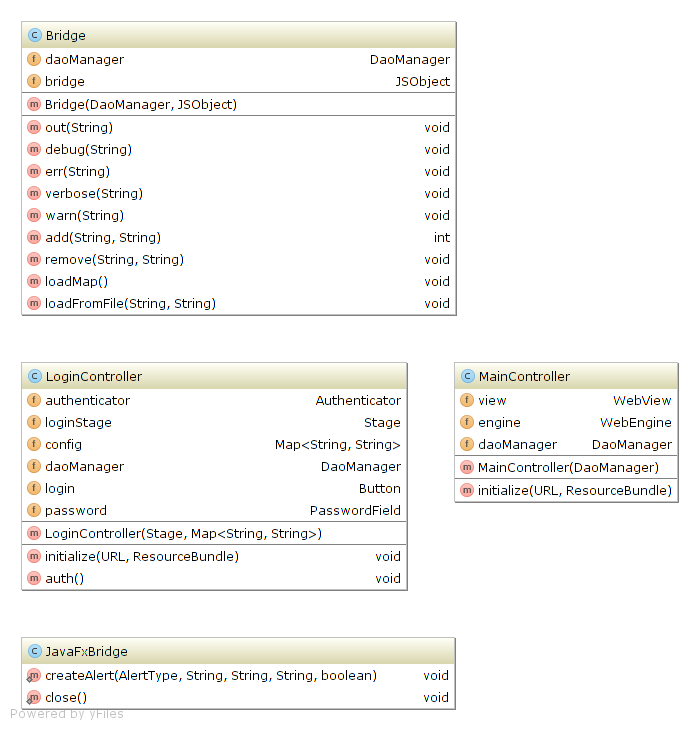
Class Definitions and Detail of Object Behaviours and Methods

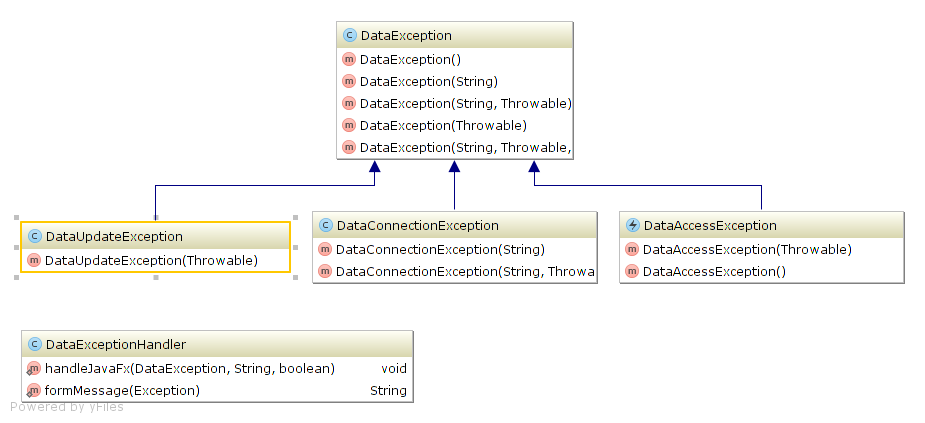
Timetabler

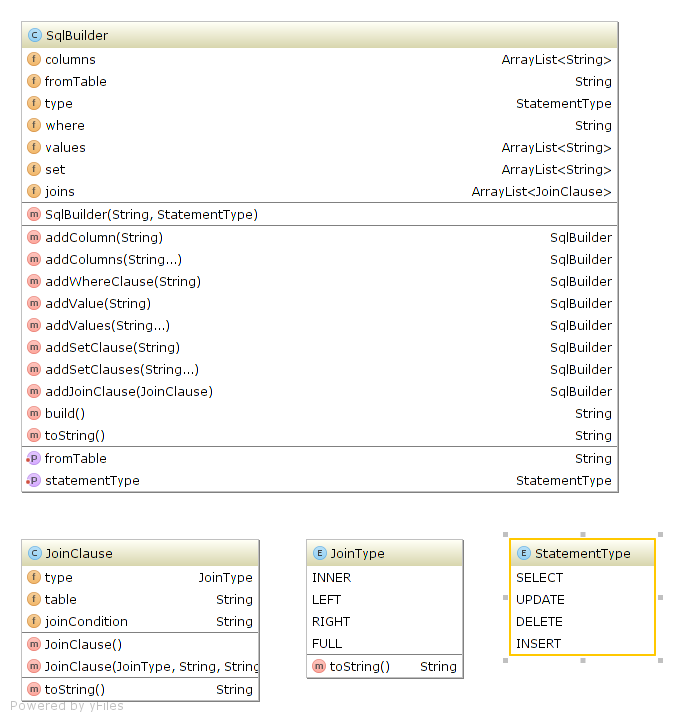
The following are the class definitions for the timetabler code. The green solid arrows represent where a class extends the class being pointed to by the arrow. The dashed green arrows show where a class implements the class being pointed to by the arrow.

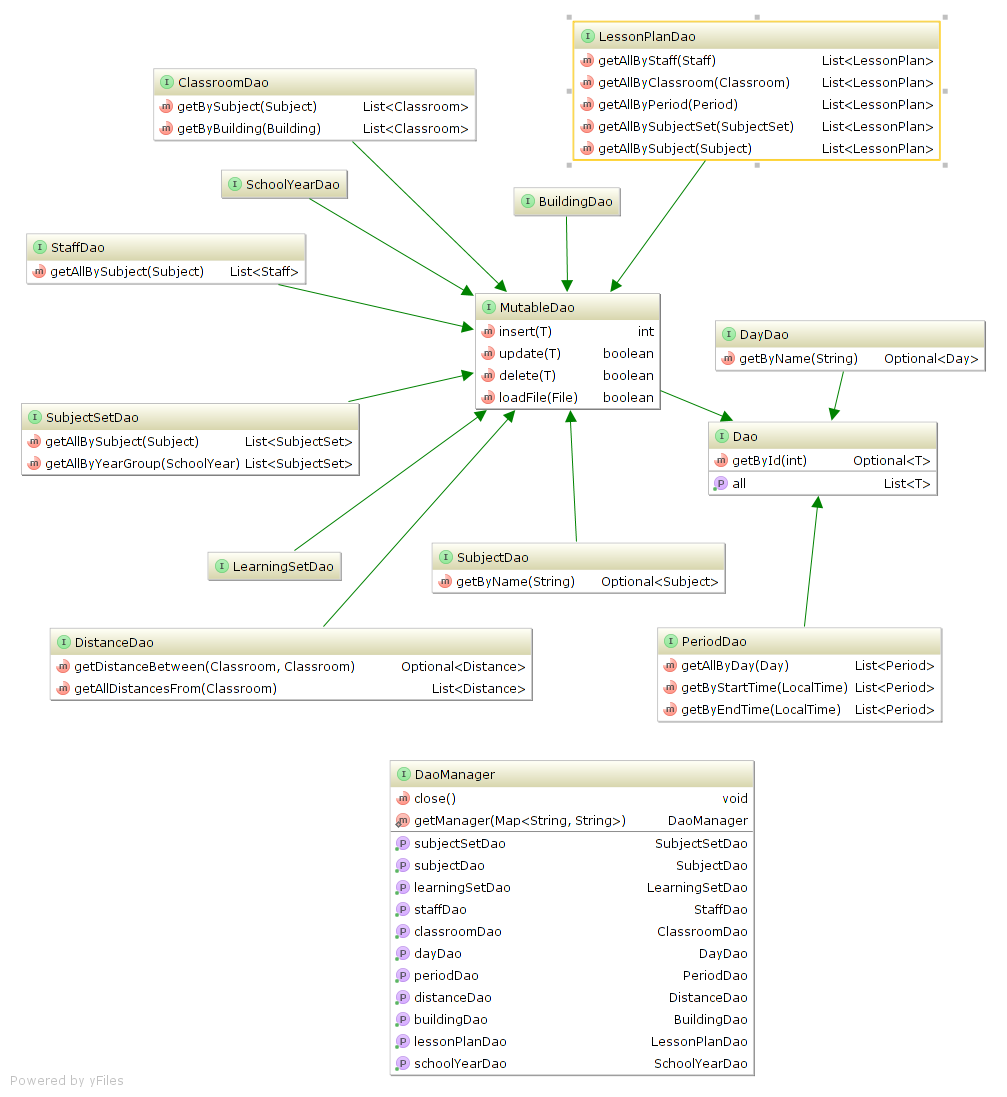


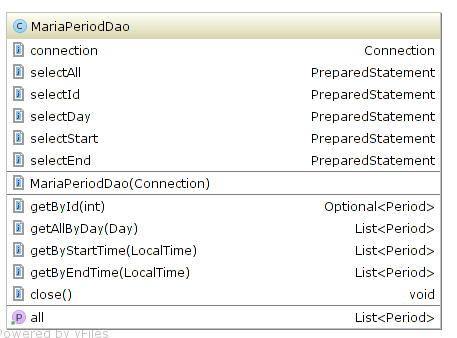
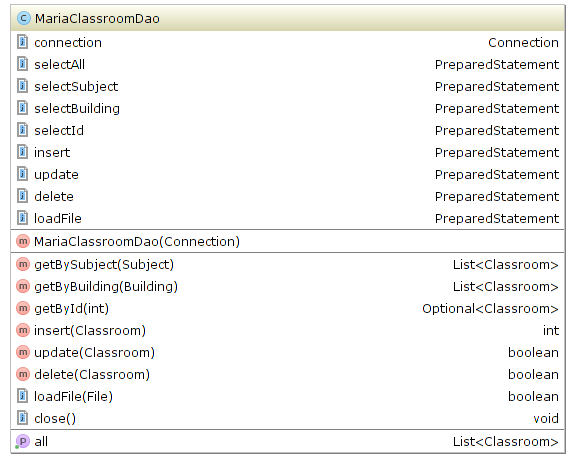
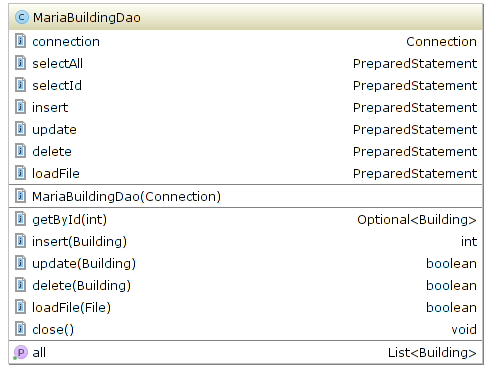


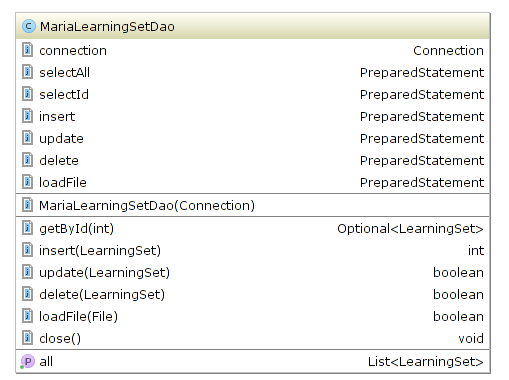
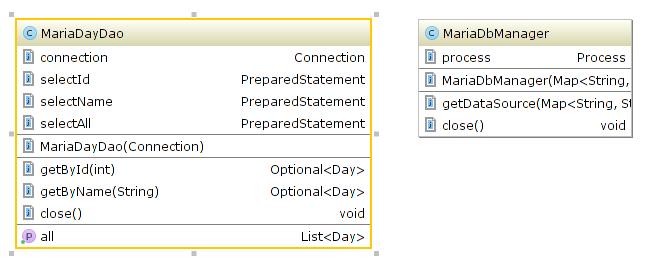
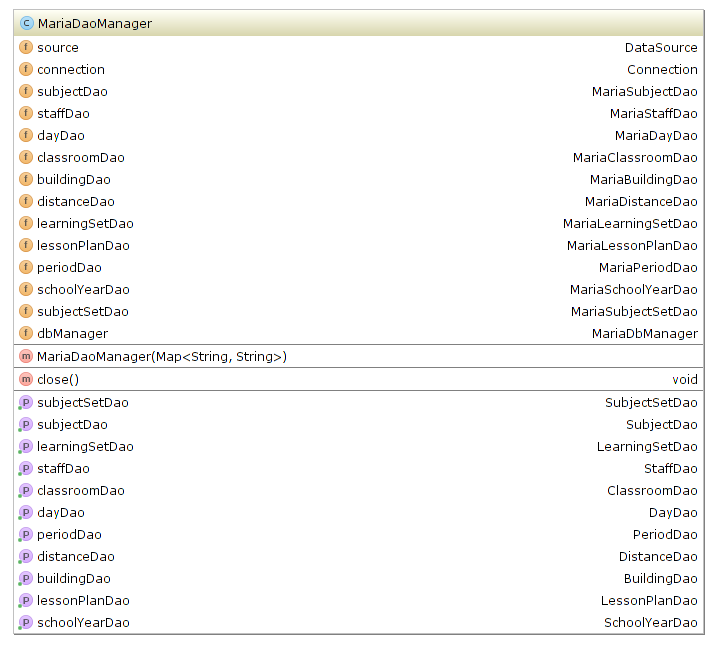


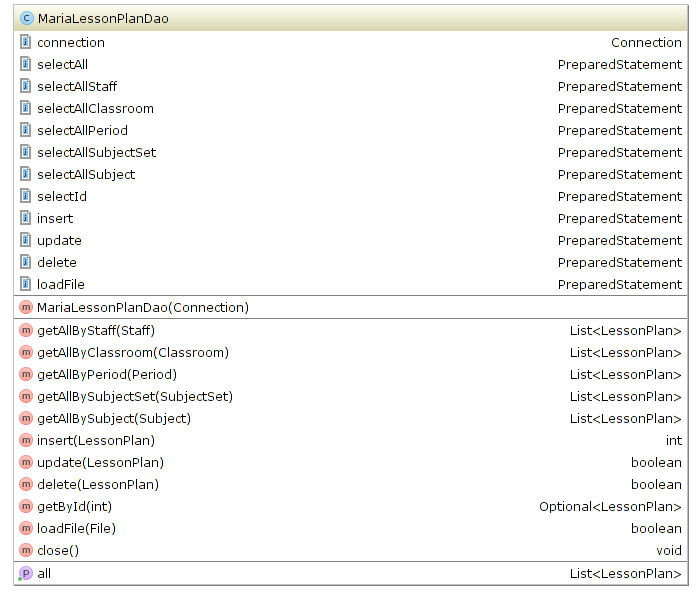


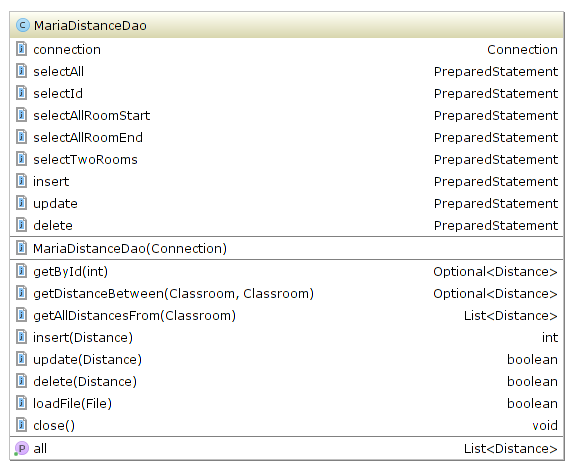


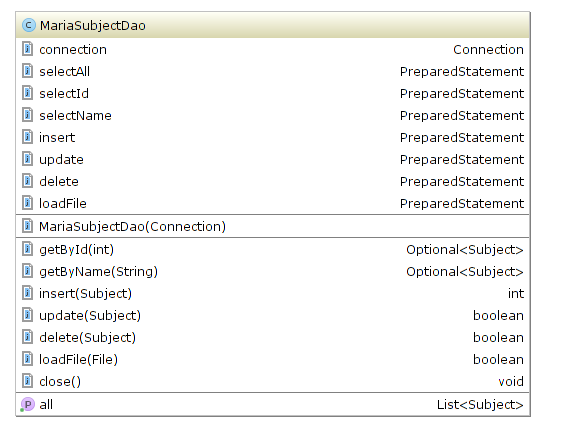
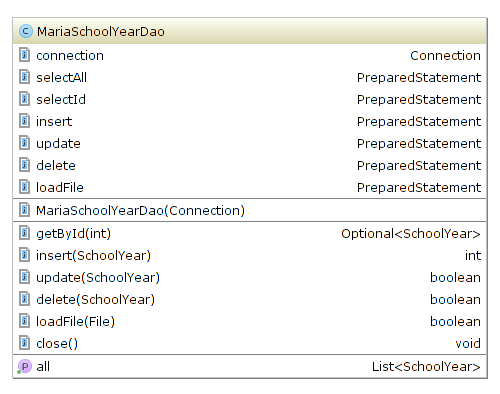


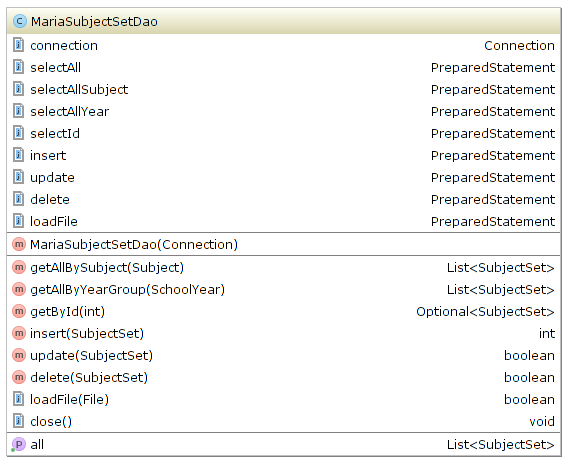






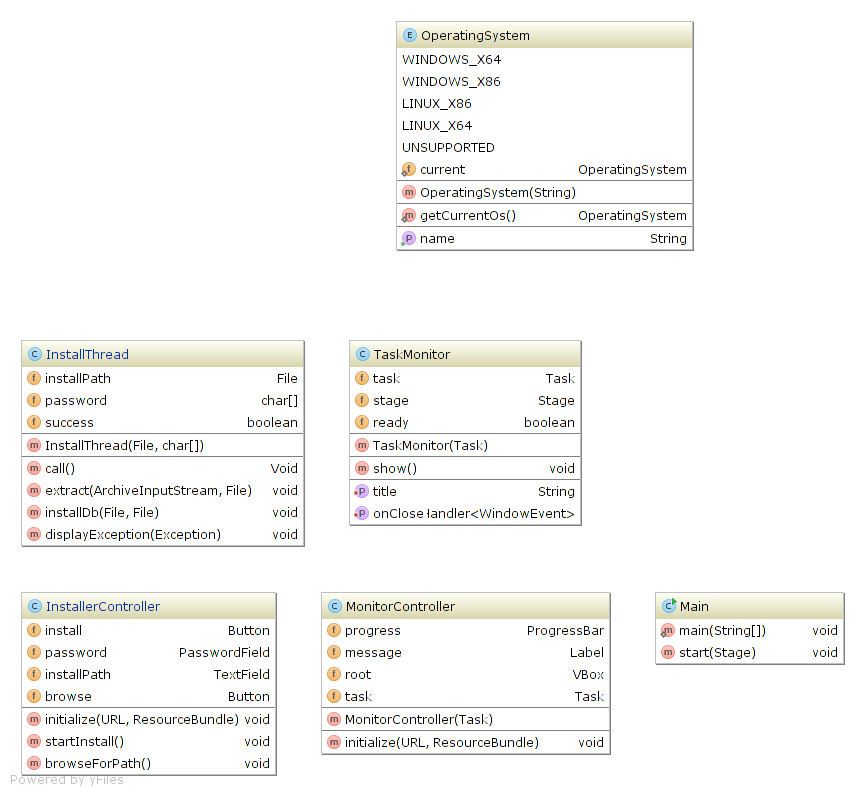




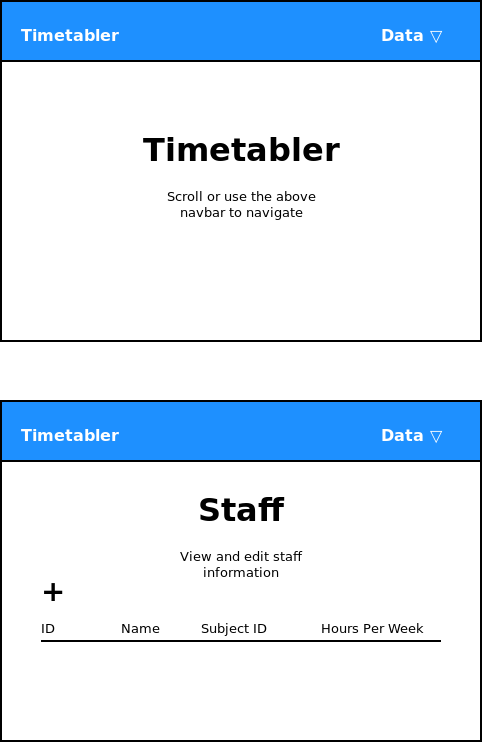


Installer

The following are the class definitions for the installation program which installs the timetabler.



User Interface Design Rational



Short introduction to reduce impact on screen space, allowing more space for the table

Use of simple iconography to increase readability and reduce space usage

Simple title page for minimal visual impact on the use of the system

Table to display all staff information in a clearly defined format

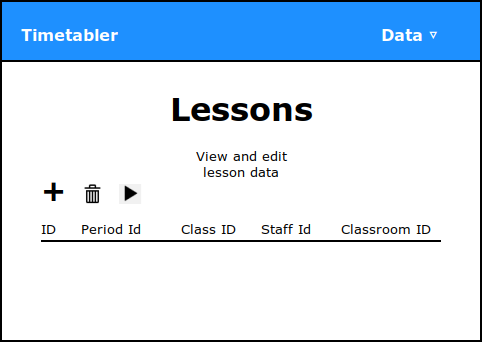
Permanent ‘navbar’ for simple, universal navigation

Dropdown menu for easy access to all sections of the interface

The above design for the data entry and display will be used as a basic template for all other data entry section. They will utilise the table layout, but will have their corresponding headings, and the data entry dialogue, but the with their corresponding data requirements. For example, the subject data entry section will have the headings, ID and Name, for the table and will prompt the user for a name for the subject they are creating using the data entry dialogue.

Lessons

The lesson data entry section will be slighly different in appearance to the rest, by having a ‘Start Timetabling’ button. This button will make the program timetable the currently stored staff and classroom into the lessons added to the timetabler system. Here is where the output data will be displayed, as the staff and classroom ids will be -1 before timetabling, and have a meaning full value after.



Simple icons to show where to add and clear data, also how to start timetabling

Sample Of Output

If the timetabler system was given some data, which could include the data below, it would be able to create an output such as the second table, but with more data than given currently.

As shown, the staff and classroom ids are -1, to show there is no specific member of staff and classroom for each lesson.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Period ID | Class ID | Staff ID | Classroom ID |
| 1 | 1 | 1 | -1 | -1 |
| 2 | 1 | 4 | -1 | -1 |
| 3 | 1 | 2 | -1 | -1 |

After running the timetabler, the table would change into a table containing data such as the data given below. Since the staff and classroom ids are no longer -1, each lesson now has a specific member of staff and classroom for the lesson to take place. The data is now in a form for the user to transcribe back into Nova T6 to be stored in the SIMS database.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Period ID | Class ID | Staff ID | Classroom ID |
| 1 | 1 | 1 | 4 | 9 |
| 2 | 1 | 4 | 3 | 11 |
| 3 | 1 | 2 | 2 | 14 |

Description of Measures Planned for Security and Integrity of the Data

In order to protect access to the data and system, at installation time, the user will specify a password for the database and will be required to enter the password at each launch of the program. This will allow the program to utilise the data security of MariaDB. At runtime, the database will active, the user will be promoted for a password. The given password will be sent to the server with the root username for authentication. Until the user provides the correct password, the system will not allow access to the data manipulation portion of the system. In order, to protect the database from external connections other than the system, it will be restricted to only one connection.

Most data entered by the user is a string, which has no required format, but has maximum lengths. The map files have references to subjects when defining a classroom, therefore the classroom will only be added to the database if the subject is already in the database. Also, due to the database being normalised, there are types of data which rely on other data, such as staff data requires correct subject data. In this case, upon data entry, the data dependencies will be checked before being added to the database, and upon data removal, the data dependencies will also be checked to ensure no data which relies on the data being removed still exists within the database. Upon entry of number data, such as hours per week for a member of staff, the system will check if the entered data is an unsigned integer before allowing it to be stored in the database.

Error messages will take the following form:



Error title

Error message and information on how to fix the error

Overall Testing Strategy

In order to fully test the system in its current capacity, I will follow the following testing strategy as it covers all the important sections of how the system works. The testing strategy does not need to test how well the system can handle a new data source other than MariaDB, as the majority of the system is unaware of the data source at all and only interacts through data access objects. This means the system will be resilient enough to handle a change in data source as long as the new implementation of the data access objects are correct.

|  |  |
| --- | --- |
| Test Series | Purpose of Test Series |
| 1 | Test the data entry sections of the user interface for adding a single entry to each section though each add entry dialogue. This will involve testing if data can be added which is correct, boundary and erroneous. |
| 2 | Test the map loading functionality, ensuring it can parse all maps correctly and calculate the necessary distances. |
| 3 | Test the system’s ability to timetable staff and classrooms into lessons with a predetermined set of test data. |