The Computing Practical Project

School Timetabler

Analysis

Background to and Identification of Problem

My client for my timetable is Mrs Lansdown, Head of Sixth Form at Hitchin Boys’ School and constructs timetables for the whole school. Hitchin Boys’ School is a secondary educational establishment with a sixth form which is part of the Hitchin Sixth Form consortium 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. This is given by well trained staff with the use of projectors, interactive whiteboards and textbooks. Currently, I am studying form A2 at Hitchin Boys’ School sixth form and Mrs Lansdown is the head of the sixth form.

Description of the Current System

Currently, Mrs Lansdown uses Nova T6 to construct timetables for the whole school and, in conjunction with the other heads of sixth in the consortium, the sixth form. 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. Sadly, Nova T6’s automatic timetable construction functionality causes staff and students to travel unnecessary distances between lessons, even though classrooms are available within a small area. Currently, the timetable is initially constructed by Nova T6, then Mrs Lansdown manually adjusts the timetable to minimise distance travelled and other problems with the automatic system. In order to improve the automatic timetabling construction functionality, the new system would have to take in to account the distance between classrooms to reduce the distance travelled by staff over a day.

Identification of the Prospective User

Mrs Lansdown will be the immediate user of the system, but the system will be designed to allow for fast tutoring in the usage of the system, to allow all following heads of sixth form to continue to use the system. The IT department of school will not use the system directly, but will setup 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

After having an interview with the client, I have ascertained the client wants a system which generates a timetable using staff, student and subject data from the current database in the SIMS system. Unfortunately, the SIMS system and its database requires a licence to access the database and interface provides by SIMS, which requires a fee to be paid to Capita and a non-free training course in how to use the interface and database. The system must have simple and intuitive user interface which utilises blocks to represent the periods in a day and which lessons are running within the block. As each teacher must have at least a specific number of ‘contact’ hours with students, the system must be able to calculate and display the number of hours worked by each member of staff and specify what subjects the members of staff are qualified to teach. Since the new system will take into account the distances between the class rooms, each member of staff should have a preferred class room where the system will attempt to base as many lessons of their lessons as possible and classrooms should have a subject linked to them to avoid large distances between lessons for subject staff.

TODO: current docs

Data Sources and Destinations

The current system utilises a Microsoft SQL server to store all data for use within the system, which is updated using the interface provided by the system. Since the database requires a non-free licence to access, the source of the data for the new system will have to different to the current system. In order to make the data entry as easy as possible, the data about the staff, subjects, classes and map of the school will all be in the CSV format, each will be their own file. This allows the user to use an external program like Microsoft Office Excel to populate the files. A configuration file will be used to specify information about the data files, such as their locations and file types, which will be in the YAML format as it is easy to modify and process. The staff, subject and class data will be dynamic as it has to be updated at runtime and as the school changes throughout the program's life cycle. The map will be treated as static data, because the layout of the school will vary rarely. Also, the configuration file will be treated as static data as it will only be modified when the data about the school has been heavily modified.

TODO: output data and possible CSV export from SIMS, change to table form

Data Volumes

The volume of data to process will be large, as the school teaches many subjects, has a vast array of members of staff and individual classes, but this data will all be text data which is small and can be efficiently compressed. The is data will only have to accessed and processed roughly once per academic year, as the school will only need to generate timetables at the beginning of an academic year, and will only be accessed by one member of staff. The staff, subjects and class data will be read from individual CSV files into a database, but the CSVs will not be deleted at runtime to allow for the user to use the CSVs to modifiy the data outside of the program without needing to understand the database. There will be roughly 20 entries in the subjects table, roughly 70 entries in the staff table, and roughly 300 entries in the class table. To allow the user to change the location of the map of the school, the top map file and the file type of the school data, there will be a YAML configuration file.

TODO: Rough estimates of bytes

Data Dictionary

Staff Data

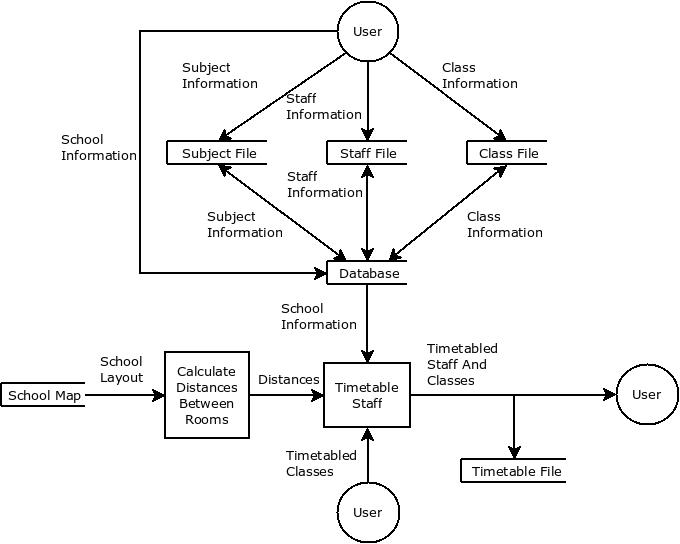
| Data | Data Type | Description | Data Size | Example Data | Validation |
| --- | --- | --- | --- | --- | --- |
| StaffID | String | The unique global identifier for the member of staff, it will be used for identification rather than their name | 3 characters | Ck | Not NULL |
| StaffName | String | The name of the member of staff. | 30 characters | Mr Cook | Not Null |
| SubjectIDs | List | The list subject IDs of the subjects this member of staff teaches. It should be in order of preference. | 5 elements | Biology, PE | Not NULL, length at least 1, valid SubjectID |

Subject Data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data | Data Type | Description | Data Size | Example Data | Validation |
| SubjectID | String | The unique global identifier for the subject taught at the school. This will be used to identify the subject rather than their name. | 3 Characters | Ma | Not NULL |
| SubjectName | String | The name of the subject taught at the school. | 30 Characters | Maths | Not NULL |

Class Data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data | Data Type | Description | Data Size | Example Data | Validation |
| ClassID | String | The globally unique identifier for the class at the school. | 3 Characters | 7Ma3 | Not NULL |
| ClassName | String | The name of the class to be taught. | 30 Characters | Year 7 Set 3 Maths | Not NULL |
| Subject | String | The subject ID of the subject taught in this class. All subject IDs must be defined in the subject data. | 3 Characters | Ma | Not NULL, valid SubjectID |
| Staff | String | The staff ID of the staff member who will teach this class. All staff IDs must be defined in the staff data. | 3 Characters | Ld | Not NULL, valid StaffID |

Dataflow Diagram

Entity Relationship Model

Object Orientation planning

| **Class Name** | **Inheritance** | **Overriding** | **Description** |
| --- | --- | --- | --- |
| Main | None | None | The entry point to the program, which binds the code base together. |
| Coordinate | None | None | A way of storing an x and y value within one object. |
| 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. |
| YamlConfigParser | ConfigParser | Parse | An implementation of ConfigParser which can parse a configuration file in the format of YAML |
| SchoolDataParser | None | None | A way of parsing all the data about the school into the database to be used at runtime. This should not be a concrete class, and should be used as a super class to all school data parsers. |
| CsvDataParser | SchoolDataParser | readStaff  readSubjects  readClasses  writeStaff  writeSubjects  writeClasses | An implementation of SchoolDataParser which can parse the data files in the CSV format. |
| 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 need 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. |
| Building | ImportantCell | getDistances  isTraversable | A cell which references another map. It should be able to calculate the distances between all ImportantCells within its map. |
| Entrance | ImportantCell | getDistances  isTraversable | A cell that represents an entrance to a building. There should be one per building in order to avoid conflicts. |
| ClassRoom | 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 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. |
| SchoolClass | None | None | A class which stores data about a single class at the school. |
| Subject | None | None | A class which stores data about a single subject taught at the school. |
| Staff | None | None | A class which stores data about a single member of staff at the school. |

**Main Class**

| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| --- | --- | --- | --- | --- |
| Private | school | School | School | A wrapper object to all data about the school. |
| Private | parser | SchoolDataParser | CsvDataParser | A parser of the school data, by default it parses CSV files. This can be changed at declaration without further modifications to usage, but does require a new implementation of SchoolDataParser. |
| Private | configType | ConfigType | YAML | The data type of the configuration file, by default it is YAML. This can be changed by changing this value without further modification to usage, but does require a new implementation of ConfigParser, and new value in ConfigType. |

| **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 | Initalises the progrmam. 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. |

**Coordinate Class**

| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| --- | --- | --- | --- | --- |
| Public | x | Integer | 0 | The x coordinate of this coordinate. |
| Public | y | Integer | 0 | The y coordinate of this coordinate. |

**ConfigParser Class**

| **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**

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

**YamlConfigParser**

| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| --- | --- | --- | --- | --- |
| Private | loc | String | assets/config.yaml | The path to the configuration file. |
| Private | mapRegex | String | .\*:\\s\* | A regular expression to determine if a line represents a YAML map. |
| Private | itemRegex | String | \\s{4}-.\* | A regular expression to determine if a line represents a YAML item. |

| **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. |
| Private | checkFile | File Path | Boolean | Checks if a file exists, is a file and can be read. |

**SchoolDataParser Class**

| **Access Type** | **Method Name** | **Parameters** | **Return Values** | **Description** |
| --- | --- | --- | --- | --- |
| Public | readStaff | None | Map of the staff data, where the key is the staff ID | Parses the class data from the staff file |
| Public | readSubject | None | Map of the subject data where the key is the subject ID | Parses the subject data from the subject file |
| Public | readClasses | None | Map of the class data where they key is the class ID | Parses the class data from the class data |
| Public | writeStaff | Map of staff, where the key is the staff ID | Boolean, only true if the data was successfully written | Writes the given staff data to the staff file |
| Public | writeSubjects | Map of subjects, where the key is the subject ID | Boolean, only true if the data was successfully written | Writes the given subject data to the subject file |
| Public | writeClasses | Map of classes, where the key is the class ID | Boolean, only true if the data was successfully written | Writes the given class data to the class file |
| Public | getParser | The data section of the configuration map | An implementation of SchoolDataParser | Gets the correct implementation of SchoolDataParser corresponding with the data type specified in the given configuration map |

**CsvDataParser**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Private | config | Configuration Map | Null | The configuration map containing information about the school data |

| **Access Type** | **Method Name** | **Parameters** | **Return Values** | **Description** |
| --- | --- | --- | --- | --- |
| Public | readStaff | None | Map of the staff data, where the key is the staff ID | Parses the class data from the staff CSV file |
| Public | readSubject | None | Map of the subject data where the key is the subject ID | Parses the subject data from the subject CSV file |
| Public | readClasses | None | Map of the class data where they key is the class ID | Parses the class data from the class CSV file |
| Public | writeStaff | Map of staff, where the key is the staff ID | Boolean, only true if the data was successfully written | Writes the given staff data to the staff CSV file |
| Public | writeSubjects | Map of subjects, where the key is the subject ID | Boolean, only true if the data was successfully written | Writes the given subject data to the subject CSV file |
| Public | writeClasses | Map of classes, where the key is the class ID | Boolean, only true if the data was successfully written | Writes the given class data to the class CSV file |

**CellType Class**

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

**ImportantCell**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 | isTraverable | None | Boolean | Returns true if a cell is traversable |

**Building Class**

| **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 |

| **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 | isTraverable | None | Boolean | Returns true as is a traversable cell |

**ClassRoom Class**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 | isTraverable | None | Boolean | Returns true as is a traversable cell |

**Entrance**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 within the same map |
| Public | isTraverable | None | Boolean | Returns true as is a traversable cell |

**Path**

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

**Wall**

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

**SchoolMap**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

| **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**

| **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' over 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 |

| **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, where it checks if finalDistance is higher than the current distance, and sets finalDistance if finalDistance is larger than distance, then winds up the stack to find another route. If the destination cell is not adjacent to the current cell, then 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 last cell. Once walker begins to wind up the stack, it will stop and try an alternative route at every point available, check if it is a shorter rout or even a possible route. |

**SchoolClass**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| Public | id | String | Null | The unique identifier for this class |
| Public | name | String | Null | The name of this class |
| Public | subjectId | String | Null | The unique identifier for the subject taught in this class |

**Subject**

| **Access Type** | **Field Name** | **Field Type** | **Initial Value** | **Description** |
| --- | --- | --- | --- | --- |
| Public | name | String | Null | The name of this subject |
| Public | id | String | Null | The unique identifier for this subject |

**Staff**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 | subjects | Array of Strings | Null | The list of subjects taught by this member of staff in order of preference |

Objectives

|  |  |
| --- | --- |
| **Objective Number** | **Details** |
| **1** | The system must be able to parse a map of the school and calculate the distances between every 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 grid-based interface, allowing the user to specify colours for subjects within the grid. |
| **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, using a configuration file. |

<https://www.mindmup.com/#m:a17ff69ab0ad720133e69d2110f3315c98>

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 was designed to 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 to learn VB.NET before being able to undertake the project as I did not learn VB.NET in my AS course. The system would be confined to Microsoft Windows and Microsoft Office. This would limit the life span of the system, as if the user migrates from Windows or Office, the system would no longer be functional. Also, the system would 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 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 the world renowned MySQL, and adjusted to have increased scalability and performance in system critical situations. By using MariaDB, I will have access to a relational database through the use of SQL using the official JDBC (Java Database Connectivity) driver. Also, Java allows me to design a system without being confined to any operating system. In order to transfer the system, one would only have to change the version of MariaDB used, as it is a natively compiled system. | By using a language which is run within a virtual machine, the system will have increased runtime requirements compared to a natively compiled language. The system would 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, for example the language does not provide a Pair class, therefore one must design one. |
| 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 me to directly modify 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 allow the system to be scalable, 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. I would have to learn PHP before undertaking the project, which will be difficult with my background in Java and its object orientated programming. The 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.

Design

Overall System Design

| **Input** | **Processes** | **Storage** | **Output** |
| --- | --- | --- | --- |
| Staff Data  Subject Data  Class Data  School Map  Timetable without staff members | Calculate distances between all classrooms to all other classrooms  Create optimal path between lessons reducing distance travelled | Database Tables:  Staff  Subjects  Classes | Timetable containing staff members in their lessons |

Modular Design

Definition of Record Structure

Validation Required

**Subject Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Validation Check** | **Description** | **Error Message** |
| id | Presence and Datatype(Integer) | The id must be not be NULL and integer | Please specify a valid integer identification number. |
| subjectName | Presence and Datatype(Varchar) | The name of the subject must not be NULL and a string of arbitrary length | Please specify a valid name of this subject. |

**Staff Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Validation Check** | **Description** | **Error Message** |
| id | Presence and Datatype(Integer) | The id must be not be NULL and integer | Please specify a valid integer identification number. |
| staffName | Presence and Datatype(Varchar) | The name of the subject must not be NULL and a string of arbitrary length | Please specify a valid name of this member of staff. |
| subjectID | Presence, Datatype(Integer) and Lookup(subject.id) | The subject id of the subject taught by this meber of staff must not be NULL, an integer, and present in the id list of the subject table. | Please specify a valid integer identification number from the current list of subjects. |

File Organisation and Processing

Database Design including Normalised Diagrams

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 is 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 runtime through a form in the user interface. In order to add a new member of staff at runtime, 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 class SET subjectID=? WHERE id=?;

Identification of Storage Media

Class Definitions and Detail of Object Behaviours and Methods

User Interface Design Rational

Description of Measures Planned for Security and Integrity of the Data

Description of Measures Planned for System Security

Overall Testing Strategy