**System Overview**

This is the system maintenance document for the timetabler program. The timetabler program is a system which is designed to be used by a school to allocate staff and classrooms into pre-timetabled lessons, while trying to reduce the distance travelled by staff members. The system store data about the school: subjects, staff, sets, year groups, classes, lessons and layout, in a MariaDB relational database. Each to the described types of data is stored in their own table, and liked using primary and foreign keys. A copy of the initialization SQL script will be in this document in order to better show the layout of the database.

In order for a new entry to be added to any of the tables, first the data is extracted from a HTML5 Bootstrap modal using Javascript, where initial data type checks are carried out. This is then passed back through to Java using JavaFX’s JSObject, since the user interface is a HTML5 web page running in a JavaFX WebView. Once Java has the data, it is converted into the correct Java object, before being passed to the corresponding data access object, held by a data access object manager, where it is passed into a Java DataBase Connectivity (JDBC) PreparedStatement, which is sent to the MariaDB server. The server then returns the generated unique ID, which is passed back up through to Javascript, where it adds the data with the ID to the correct HTML table for the user to see.

Once the user specifies to location of the highest level map, and the lower maps, the system parses the highest map into a SchoolMap object, which wraps a 2D array of CellTypes. All maps are in the CSV format, where an empty cell is a Wall, a ‘0’ is a path, a building is just a name and a classroom is in the form C-*name*-*subject* where name is the name of the room and subject is the name of the subject taught in the room. The lower maps are loaded in after the highest map loads, and then each building in the collections of maps is initialised using the loaded maps. This is where the distances are calculated, by having a Walker object,walk recursively across the map to find the shortest distance between two rooms. Once is finds a path, it stores it length, then winds up the stack to find the next available path, and test if it is the shortest. The walker cannot detect beforehand if it will enter an infinite loop, so the user will be told to avoid possible loops in a map file. The shortest distance is then stored in both classroom objects, to be stored in the database after all the parsing and walking is done.

Due to the use of abstractions and interfaces, the majority of the system does not know it is using MariaDB as its data store. This allows for another developer to change the data store with no impact to system critical code. To enable the system to use a new data store, one must implement all interfaces in the me.timetabler.data.daos package, add a new case to the DaoManager’s getManager method’s switch statement and change the entry ‘*type*’ in the ‘*data\_type*’ map in the configuration file to your new data store. The rest of the system will continue to work as normal.

Algorithms

Map Walker

This method recusively transverses a map, by increasing the distance travelled, then checking if the ‘dest’ coordinate is adjacent to the ‘start’ cell, storing the distance in the ‘finalDistance’ variable and winding up the stack if it is. Then it checks north, south, each and west for a traversable cell and not the ‘last’ cell in that order, calling itself with the new start, last and distance variables. If no cell is applicable, it winds up the stack until it finds a new path to traverse. If all paths are traversed, the method returns, leaving the shortest distance stored in the ‘finalDistance’ variable.

finalDistance ← MAX\_INTEGER

PROCEDURE walk(currCoord, finalCoord, lastCoord, currDist)

currDist ← currDist + 1

IF currCoord.adjecentTo(lastCoord) THEN

finalDistance = 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

private void move(Coordinate start, Coordinate dest, Coordinate last, int distance) {

distance++;

CellType destination = schoolMap.getCell(dest);

CellType north, south, east, west;

north = schoolMap.getCell(start.x, start.y - 1);

south = schoolMap.getCell(start.x, start.y + 1);

east = schoolMap.getCell(start.x + 1, start.y);

west = schoolMap.getCell(start.x - 1, start.y);

if (north.equals(destination) || south.equals(destination) || east.equals(destination) || west.equals(destination)) {

if (finalDistance > distance) {

finalDistance = distance;

}

return;

}

if (!last.equals(new Coordinate(start.x, start.y - 1)) && north.isTraversable()) {

move(new Coordinate(start.x, start.y - 1), dest, start, distance);

}

if (!last.equals(new Coordinate(start.x, start.y + 1)) && south.isTraversable()) {

move(new Coordinate(start.x, start.y + 1), dest, start, distance);

}

if (!last.equals(new Coordinate(start.x + 1, start.y)) && east.isTraversable()) {

move(new Coordinate(start.x + 1, start.y), dest, start, distance);

}

if (!last.equals(new Coordinate(start.x - 1, start.y)) && west.isTraversable()) {

move(new Coordinate(start.x - 1, start.y), dest, start, distance);

}

}

**SQL**

The following SQL script is used to create the database, and can also be used to replace the existing database as removes the database if it already exists.

DROP DATABASE IF EXISTS school;

CREATE DATABASE school;

USE school;

CREATE TABLE subject

(id INT UNSIGNED NOT NULL AUTO\_INCREMENT,

subjectName VARCHAR(20) NOT NULL,

PRIMARY KEY (id));

CREATE TABLE staff

(id INT UNSIGNED NULL AUTO\_INCREMENT,

staffName VARCHAR(20) NOT NULL,

subjectId INT UNSIGNED NOT NULL,

hoursPerWeek INT UNSIGNED NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (subjectId) REFERENCES subject(id));

CREATE TABLE dayOfWeek

(id INT UNSIGNED NOT NULL,

dayOfWeek CHAR(9) NOT NULL,

PRIMARY KEY (id));

CREATE TABLE building

(id INT UNSIGNED NOT NULL AUTO\_INCREMENT,

buildingName VARCHAR(10) NOT NULL,

PRIMARY KEY (id));

CREATE TABLE classroom

(id INT UNSIGNED NOT NULL AUTO\_INCREMENT,

roomName VARCHAR(10) NOT NULL,

buildingId INT UNSIGNED NULL,

subjectId INT UNSIGNED NULL,

PRIMARY KEY (id),

FOREIGN KEY (buildingId) REFERENCES building(id),

FOREIGN KEY (subjectId) REFERENCES subject(id));

CREATE TABLE period

(id INT UNSIGNED NOT NULL AUTO\_INCREMENT,

dayId INT UNSIGNED NOT NULL,

startTime TIME NOT NULL,

endTime TIME NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (dayId) REFERENCES dayOfWeek(id));

CREATE TABLE learningSet

(id INT UNSIGNED NOT NULL AUTO\_INCREMENT,

setName VARCHAR(10) NOT NULL,

PRIMARY KEY (id));

CREATE TABLE schoolYear

(id INT UNSIGNED NOT NULL AUTO\_INCREMENT,

schoolYearName VARCHAR(8) NOT NULL,

PRIMARY KEY (id));

CREATE TABLE subjectSet

(id INT UNSIGNED NOT NULL AUTO\_INCREMENT,

subjectId INT UNSIGNED NOT NULL,

setId INT UNSIGNED NOT NULL,

schoolYearId INT UNSIGNED NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (subjectId) REFERENCES subject(id),

FOREIGN KEY (setId) REFERENCES learningSet(id),

FOREIGN KEY (schoolYearId) REFERENCES schoolYear(id));

CREATE TABLE distance

(id INT UNSIGNED NOT NULL AUTO\_INCREMENT,

startRoomId INT UNSIGNED NOT NULL,

endRoomId INT UNSIGNED NOT NULL,

distance INT NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (startRoomId) REFERENCES classroom(id),

FOREIGN KEY (endRoomId) REFERENCES classroom(id));

CREATE TABLE lessonPlan

(id INT UNSIGNED NOT NULL AUTO\_INCREMENT,

staffId INT UNSIGNED NULL,

classroomId INT UNSIGNED NULL,

periodId INT UNSIGNED NOT NULL,

subjectSetId INT UNSIGNED NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (staffId) REFERENCES staff(id),

FOREIGN KEY (classroomId) REFERENCES classroom(id),

FOREIGN KEY (subjectSetId) REFERENCES subjectSet(id),

FOREIGN KEY (periodId) REFERENCES period(id));

INSERT INTO dayOfWeek (id,dayOfWeek) VALUES (1,'Monday'),(2,'Tuesday'),(3,'Wednesday'),(4,'Thursday'),(5,'Friday');

INSERT INTO period (id,dayId,startTime,endTime) VALUES

(1,1,'9:10:00','10:10:00'),(2,1,'10:10:00','11:10:00'),(3,1,'11:30:00','12:30:00'),(4,1,'1:30 PM','2:30 PM'),(5,1,'2:30 PM','3:30 PM'),(6,1,'3:30 PM','4:30PM'),

(7,2,'9:10:00','10:10:00'),(8,2,'10:10:00','11:10:00'),(9,2,'11:30:00','12:30:00'),(10,2,'1:30 PM','2:30 PM'),(11,2,'2:30 PM','3:30 PM'),(12,2,'3:30 PM','4:30PM'),

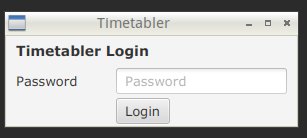
(13,3,'9:10:00','10:10:00'),(14,3,'10:10:00','11:10:00'),(15,3,'11:30:00','12:30:00'),(16,3,'1:30 PM','2:30 PM'),(17,3,'2:30 PM','3:30 PM'),(18,3,'3:30 PM','4:30PM'),

(19,4,'9:10:00','10:10:00'),(20,4,'10:10:00','11:10:00'),(21,4,'11:30:00','12:30:00'),(22,4,'1:30 PM','2:30 PM'),(23,4,'2:30 PM','3:30 PM'),(24,4,'3:30 PM','4:30PM'),

(25,5,'9:10:00','10:10:00'),(26,5,'10:10:00','11:10:00'),(27,5,'11:30:00','12:30:00'),(28,5,'1:30 PM','2:30 PM'),(29,5,'2:30 PM','3:30 PM'),(30,5,'3:30 PM','4:30PM');

**User Interface**

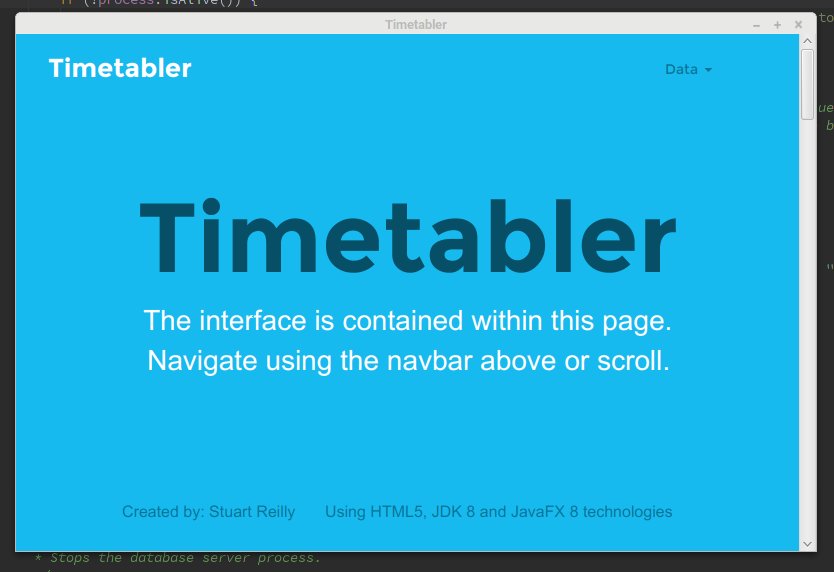
Login Screen



The password the user believes was entered at installation

Calls the Authenticator’s authenticate method, giving it the password given here, opening the main window if it successfully autheticates

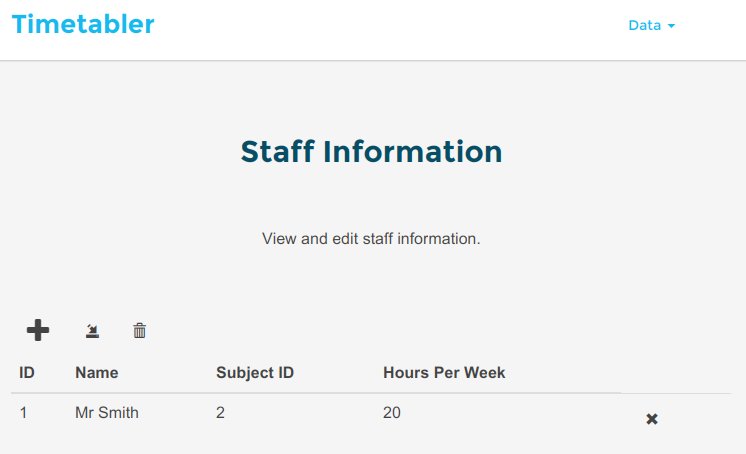
Main Section of Main Window



Navigation bar which is visible in all sections of the main window

Staff Section of Main Window

Calls the importFile Javascript method with the parameter ‘staff’



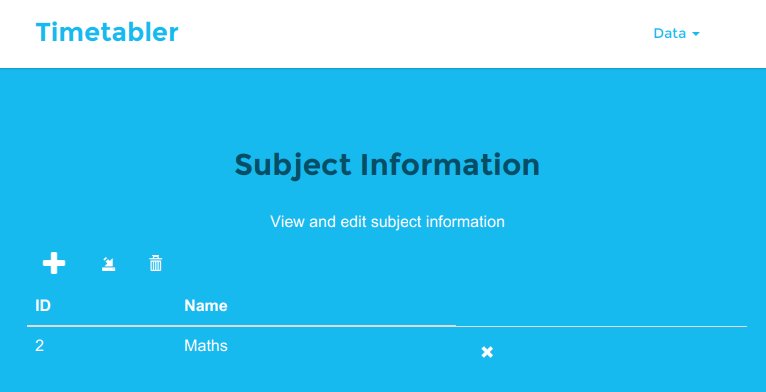
Adds a member of staff, by opening the ‘Add Staff’ modal

The table containing all the members of staff in the database

Calls the clearTable method, with the parameter ‘staffTable’

Subject Section of Main Window

Calls the Javascript method ‘clearTable’, with the parameter ‘subjectTable’



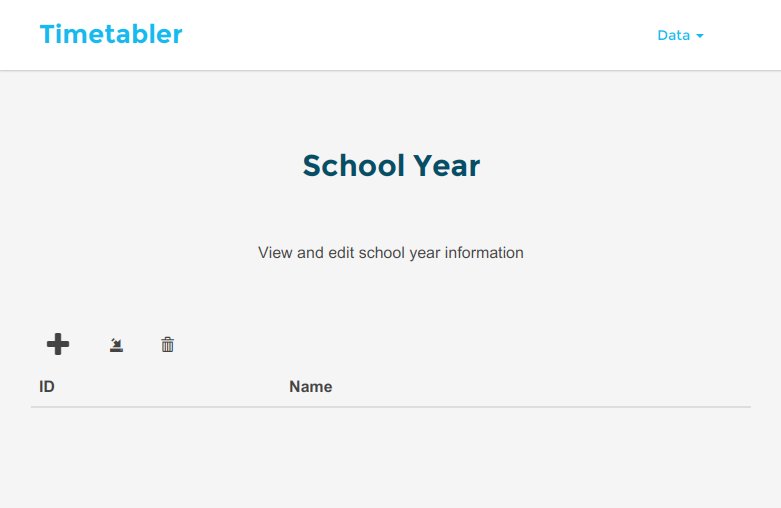
Adds a member of staff, by opening the ‘Add Staff’ modal

The table containing the subjects in the database

Calls the Javascript method ‘loadFile’, with the parameter ‘subject’

Year Group Section of Main Window

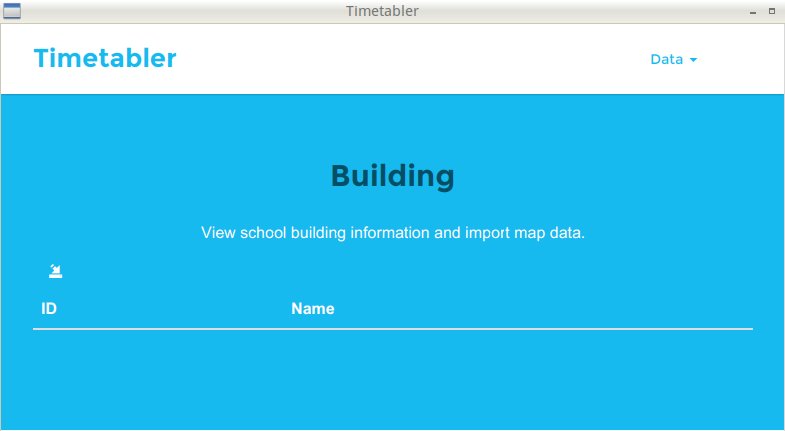
Calls the Javascript method ‘clearTable’, with the paramter ‘yearTable’



Adds a year group by opening the year modal

Calls the Javascript method ‘loadFile’, with the parameter ‘schoolYear’

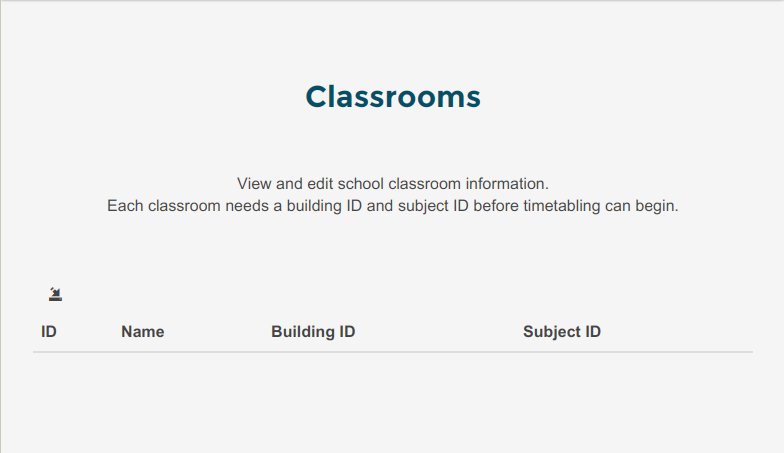
Building Section of Main Window



Calls the Javascript method ‘loadMap’

The table containing all the buildings in the database

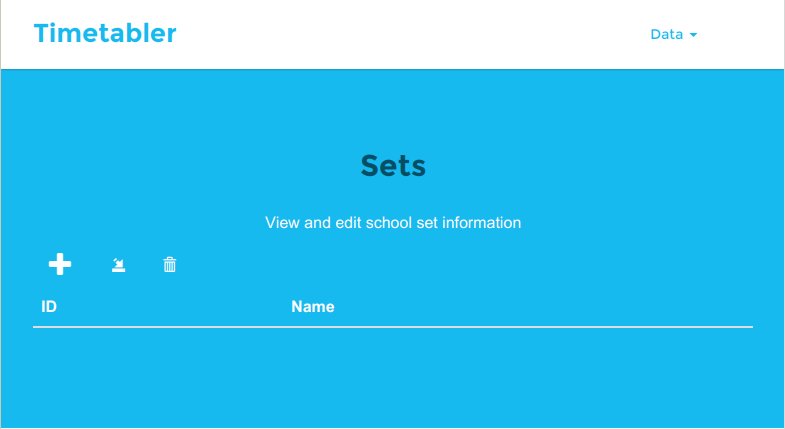
Classroom Section of Main Window



Calls the Javascript method ‘loadMap’

The table containing all the classrooms in the database

Set Section of Main Window



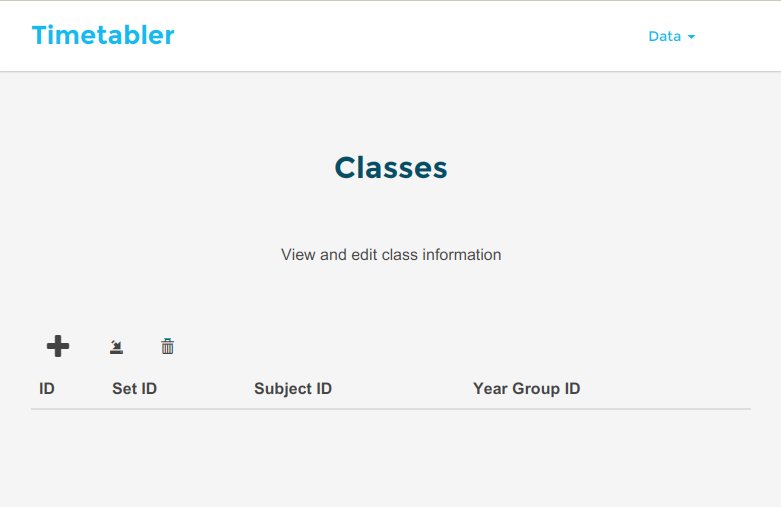
Calls the Javascript method ‘clearTable’, with the parameter ‘setTable’

Adds a new set by opening the set modal

Calls the Javascript method ‘importFile’ with the parameter ‘learningSet’

The table containing all the sets in the database

Class Section of Main Window



Calls the Javascript method ‘clearTable’ with the parameter ‘classTable’

Adds a new class by opening the class modal

Calls the Javascript method ‘importFile’ with the parameter ‘subjectSet’

The table containing all the classes in the database

Lesson Section of Main Window



Table containing all the stored in the database

Adds a new lesson, by opening the lesson modal

Removes all lessons, by calling the Javascript method ‘clearTable’

Timetables the lessons, by calling the Javascript method ‘timetable’