

ORDR: An Sports Club kit-ordering system

Thomas Moffat

Candidate Number: 4042

Centre Number: 51337

CONTENTS

1	Analysis	3
1.1	Background to and Identification of the Problem	3
1.2	Interview With Primary User	3
1.3	The Current System	5
1.4	Prospective Users	6
1.5	User Needs and Acceptable Limitations	6
1.6	Data Sources and Destinations	7
1.7	Data Volumes and Data Dictionaries	8
1.8	Data Flow Diagrams	9
1.9	Entity Relationship Models	11
1.10	Entity Description	11
1.11	Objectives for the Proposed System	12
1.12	Potential Solutions	12
1.13	Feasibility of Potential Solutions	13
1.14	Justification of Chosen Solution	14
2	Design	15
2.1	Overall System Design	15
2.2	Description of Modular System Structure	16
2.3	Design Data Dictionary	18
2.4	Database Design	19
2.5	Identification of Storage Material and Format	19
2.6	Identification of Processes and Algorithms for Data Transformation	19
2.7	User Interface Design and Rationale	19
2.8	Planned Data Capture and Entry	19
2.9	Planned Valid Output Designs	19
2.10	Measures Planned for Security and Integrity of Data	19
2.11	Measures Planned for System Security	19
2.12	Overall Test Strategy	20
	Appendices	20
A	Source Code Appendix	20

LIST OF FIGURES

Figure 1	The current order form	5
Figure 2	A level 0 data flow diagram of the current system	9
Figure 3	A level 1 data flow diagram of the current system	9
Figure 4	A level 0 data flow diagram of the proposed system	10
Figure 5	A level 1 data flow diagram of the proposed system	10

Figure 6	The proposed Entity Relationship diagram, exported from MySQL-Workbench	11
Figure 7	The part of the project facing the customer. This is all written in web-based technology	16
Figure 8	The part of the project facing the Kit Coordinator. This is all written in Java, except for the SQL script and the CSV	17

LIST OF TABLES

Table 1	Current Data Sources and Destinations	7
Table 2	Proposed Data Sources and Destinations	7
Table 3	Current Data Dictionary	8
Table 4	Proposed Data Dictionary	8
Table 5	Summary Table	15
Table 6	Part 1 of top down design	17
Table 7	Part 2 of top down design	17
Table 8	Customer Table Data Dictionary	18
Table 9	Order Data Dictionary	18
Table 10	Items Data Dictionary	18
Table 11	Connection XML Data Dictionary	19

1 ANALYSIS

1.1 Background to and Identification of the Problem

The client for this program is Kira, my mother, who volunteers as a kit orders organiser for Tilehurst Swimming Club, who asked me to develop a solution for her to be able to organise kit orders more effectively.

The current system is paper-based, so it is very slow and very inconvenient for her as it requires her to spend vast amounts of time filling in a spreadsheet in order to organise an order, and then she has to copy it all out of the spreadsheet into an email to send off to the kit suppliers. A computerised solution would alleviate some of the time required to do this and might even allow her to automate most of it.

1.2 Interview With Primary User

- What is the current system?

The current system is manual and uses either a paper form with a BACS (Bankers Automated Clearing Service), cheque or cash payment or an email of the same form with usually a BACS payment (sometimes a cheque delivered later) with all the relevant data being entered onto a spreadsheet for record keeping. The data is then transferred manually to an order sheet that is used to place the order at the printers.

The initial data required to be processed and kept track of by the club is - Name, Number, Garment Type, Size, Personalisation, Cost, Total Cost and Payment Made.

The data required for the order sheet that is given to the printers requires only Garment Type, Size and Personalisation and is categorised by Garment Type.

- What are the benefits of the existing system?

At the time of set up, this system did not require a lot of time to implement.

- What are the drawbacks?

Due to its simplicity, the current system is time-consuming. Each order requires a lot of manual entry and data processing, which could easily be achieved in a more automated way. Data can get lost in between the order spreadsheet and the order sheet that is sent to the printers.

- Which new features would be most useful?

- A usable interface to enter the order data by the parents or by the person with the responsibility for kit ordering in the club.
- Storage of this data in a usable way.
- Automatic creation of the order sheet from the initial data on a monthly basis
- Emailing the order sheet to the printer with a covering email.
- Ability to send an email to the parents to update them with the order status.

- Which existing features would you like kept?

The new system should be based on the old system but be a better version.

- Who would be using this system?

- On the front end?
The swimmers or swimmers' parents or the kit order person.
- On the back end?
The kit order person.

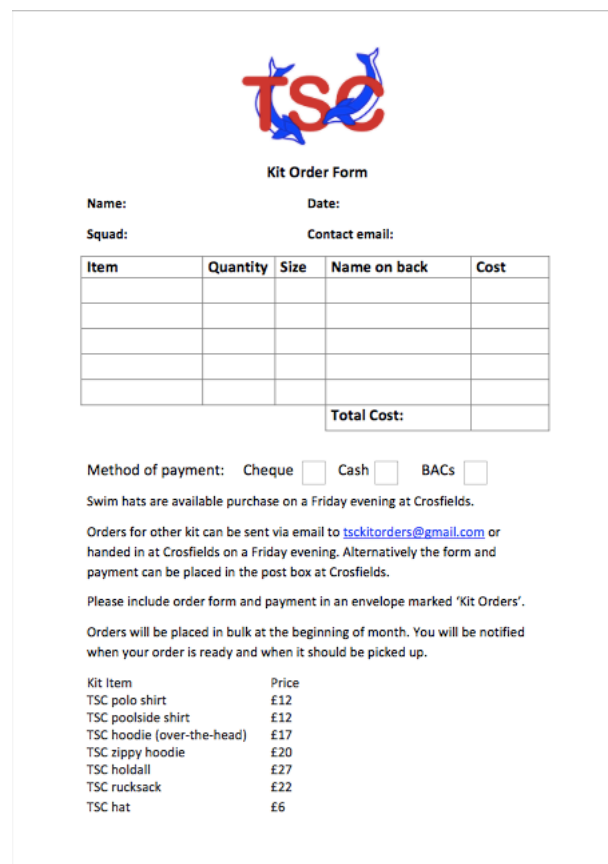
- How often would you expect to be using the system?

The system would be used monthly to create the order sheet for the printers.

- How often would you expect others will use the system?
Parents or swimmers could use the system daily to place orders.
- Will you need any security on it?
Security for email addresses.

1.3 The Current System

The current system is paper-based, so people wanting to order kit have to download a form from the club website then fill it in and either hand it in on one a Friday night or email it to the kit email address, which then requires Kira to collate all of these orders in to one before then sending it off to the manufacturers. The kit form is seen here:



TSC

Kit Order Form

Name: _____ Date: _____

Squad: _____ Contact email: _____

Item	Quantity	Size	Name on back	Cost
Total Cost:				

Method of payment: Cheque ☐ Cash ☐ BACs ☐

Swim hats are available purchase on a Friday evening at Crosfields.

Orders for other kit can be sent via email to tsckitorders@gmail.com or handed in at Crosfields on a Friday evening. Alternatively the form and payment can be placed in the post box at Crosfields.

Please include order form and payment in an envelope marked 'Kit Orders'.

Orders will be placed in bulk at the beginning of month. You will be notified when your order is ready and when it should be picked up.

Kit Item	Price
TSC polo shirt	£12
TSC poolside shirt	£12
TSC hoodie (over-the-head)	£17
TSC zippy hoodie	£20
TSC holdall	£27
TSC rucksack	£22
TSC hat	£6

Figure 1: The current order form

This is obviously a very slow system, especially as the orders only get placed once a month, so it can take up to a month to receive kit that has been ordered, and possibly longer if the orders have been forgotten, which has happened far too many times in the

past. This also has the problem of wasting a large amount of paper, as the forms are just collated on the computer and then discarded.

Yet another problem with this system is that it requires everyone to pay attention to the dates, as missing the deadline could mean a wait of another month, and if the kit has been ordered for a large competition then it could very easily mean that the swimmers don't get what was ordered in time for the competition, as it normally takes a couple of weeks for all of the ordered kit to be printed and then picked up again. After the orders have been placed and then received, it requires Kira to email out to all the parents that their kit has been received and then requires her to bring it to a Friday-evening session when the parents are also present, and not all of the members of the squads, especially in the lowest squad swim on Fridays, so it can be a few weeks or months before the swimmers actually get the kit that was ordered. The objective for this project then, is to make a way for parents to order kit and then for Kira to be able to collate this together without hours of data entry.

1.4 Prospective Users

The prospective main users of this system will be Kira and then whomever takes over from her when she steps down as Kit Organiser. For this reason, I will need to assume that the users of this system are not tech-savvy, due to the fact that, while I know that Kira knows how to use computers fairly well, I don't know who will be following her so I don't know what their capabilities are regarding computers. For this reason the solution will have to be very simple so that people of any ability can use the system.

The secondary users of this system will be the parents that are ordering kit using a form on the club website, so the forward-facing system will have to be very easy to use, as I don't know all of the parents so I have to assume that some of them will be tech-illiterate, or at the very least, uncomfortable with computers.

1.5 User Needs and Acceptable Limitations

Kira needs to have a way that she can have the parents order what they want and then have it in a searchable database so that she can just create an email to send to the kit manufacturers once a month. Then when she receives the kit she wants to be able to send out a mass email to the people who have ordered kit the previous month that tells them their kit is ready to be picked up.

Although the parents of the swimmers won't be the primary users, they will be affected quite a lot by the system, so it should be easy to navigate and similar in layout to the paper order form, to facilitate change-over. They will need a way to order kit, in a simple layout that then makes sure they know when their kit has been ordered and when it has come in.

The acceptable limitations for this will be:

- The hardware this will be running off will be somewhat underpowered, as if it is hosted locally it will be running off a 2009 MacBook Pro, and otherwise, if it is web-based it will be running on the club's web server which is not configured for a large volume of data and a large number of users using it at once.
- My skills and knowledge - The system will have to not be too complex for me to create, as I have limited programming skills and limited resources. There are a few ways I could make this system, so I will have to be careful to not choose an overly simple solution, just because it is easy.
- Time constraints - This system will need to be finished by February half term.
- Features not able to be implemented due to complexity - Although this is an order system, it will have to work on a trust-based system as it would be far too complex for me to add in a payment solution, either involving BACS or something else. For this reason, the payments will still be processed manually, and people who haven't paid will be chased up in person.

1.6 Data Sources and Destinations

The sources of data are the parents ordering kit, by way of the order form and Kira entering the details into an Excel spreadsheet. This source will not change with the new system, however it won't be via Kira, it will be automatically added in to a database. In the current system the spreadsheet is printed out and then sent off to the kit manufacturer and then Kira receives an email when they are ready for the kit to be picked up. In the new system, the data would again be arranged into a spreadsheet and printed out. This is an unfortunate limitation of the kit suppliers, not a problem on the club's end.

Table 1: Current Data Sources and Destinations

What is it	Source	Destination
Customer Details	Parents filling in an order	Excel Spreadsheet
Order	Parents filling in an order	Spreadsheet
Order Details	Excel Spreadsheet	Word Document

Table 2: Proposed Data Sources and Destinations

What is it	Source	Destination
Customer Details	Parents filling in an order	customerDatabase
Order details	Parents ordering kit	orderDatabase
Admin Details	Kit Organiser	orderDatabase
Order Details	orderDatabase	Word Document or Email

1.7 Data Volumes and Data Dictionaries

The volume of data will be very low as the system will work entirely in text, only one person will be accessing the back-end of it, the volume of kit orders is fairly low, although high enough for this to be a problem. Also, this will only be accessed once or twice a month so the data volumes will be kept low.

A rough calculation (using the data in Table 4) would suggest that, per order, 1156.25 bytes will be produced (i.e. 1156 bytes, 2 bits). This would suggest that, at an average order size of about 10 orders per month, the system will produce about 10kB of data per month. This is an average, there will be more produced in the run up to the large competitions of the year, and less after said competitions.

Table 3: Current Data Dictionary

Data	Data Type	Description
Name	Text	Name of the customer
Order	Text	What the customer ordered
Order Quantity	Number	How many of each item was ordered
Paid	Text	Shows how the customer has paid
Email address	text	Customer's email address
Squad	text	What swimming squad the child is in
Name on back	text	what name the customer would like printed
Size	text	what size the clothes are
Cost	text/numbers	how much the overall cost will be

Table 4: Proposed Data Dictionary

Field Name	Purpose	Type	Typical length (B)	Example	Validation
Name	Stores name of customer	String	40 (320)	Alex	Not blank
Ordered Kit	Stores type of kit ordered	String	40 (320)	Polo shirt	Not blank
Quantity ordered	stores number of each item	Integer	2 (4)	1	>-1, <100
Paid?	Stores if order has been paid	boolean	1 (1/8)	True	true or false
Ordered	stores if order has been placed	bool	1 (1/8)	True	true or false
Email address	stores email address	String	47 (376)	abc@abc.com	Not blank
Name on back	stores name printed on the back	String	10 (80)	Pedro	Not blank
Squad	stores what squad swimmer is in	String	3 (24)	Top	Not blank
Size	size the items of clothing will be	String	4 (32)	M	Not blank

1.8 Data Flow Diagrams

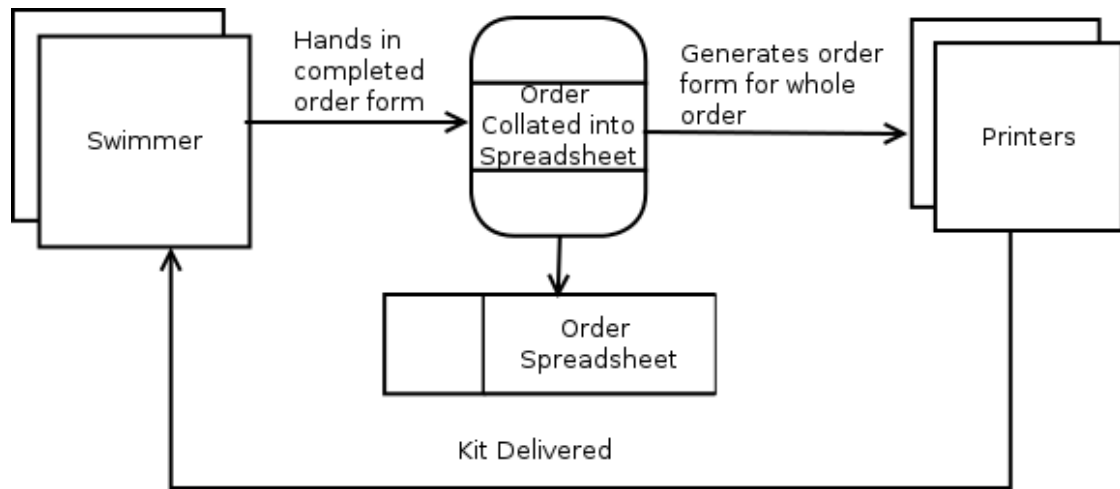


Figure 2: A level 0 data flow diagram of the current system

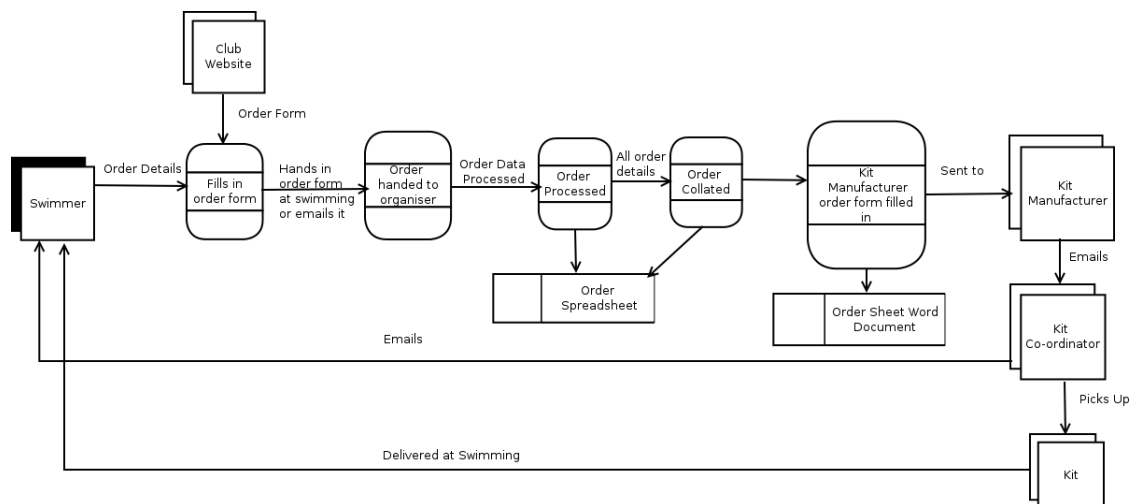


Figure 3: A level 1 data flow diagram of the current system

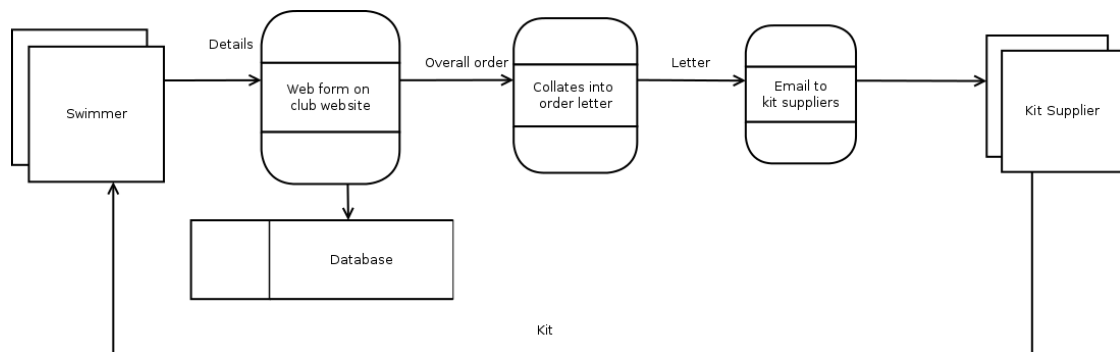


Figure 4: A level 0 data flow diagram of the proposed system

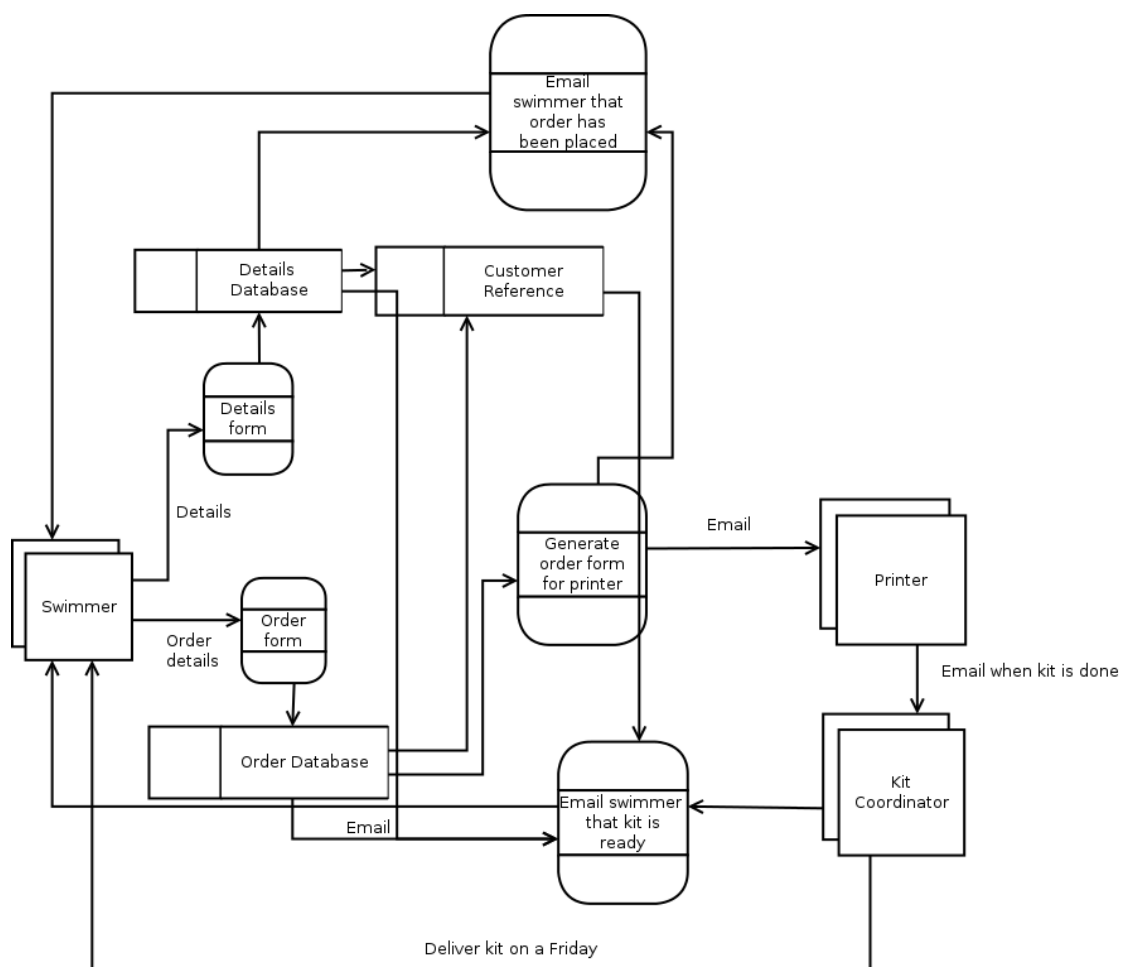


Figure 5: A level 1 data flow diagram of the proposed system

The proposed system as seen in figure 5 is rather more complicated than the current system seen in figure 3 which unfortunately means that there are more things that will need to be kept track of and so more things that could potentially break, however this will be traded off with a massive increase in convenience for everyone, not to mention that most of the system can be automated which will reduce the kit coordinator's workload. This will also mean that the system will be much faster, as orders will be entered into the system immediately and so will not be forgotten, unlike in the current system.

1.9 Entity Relationship Models

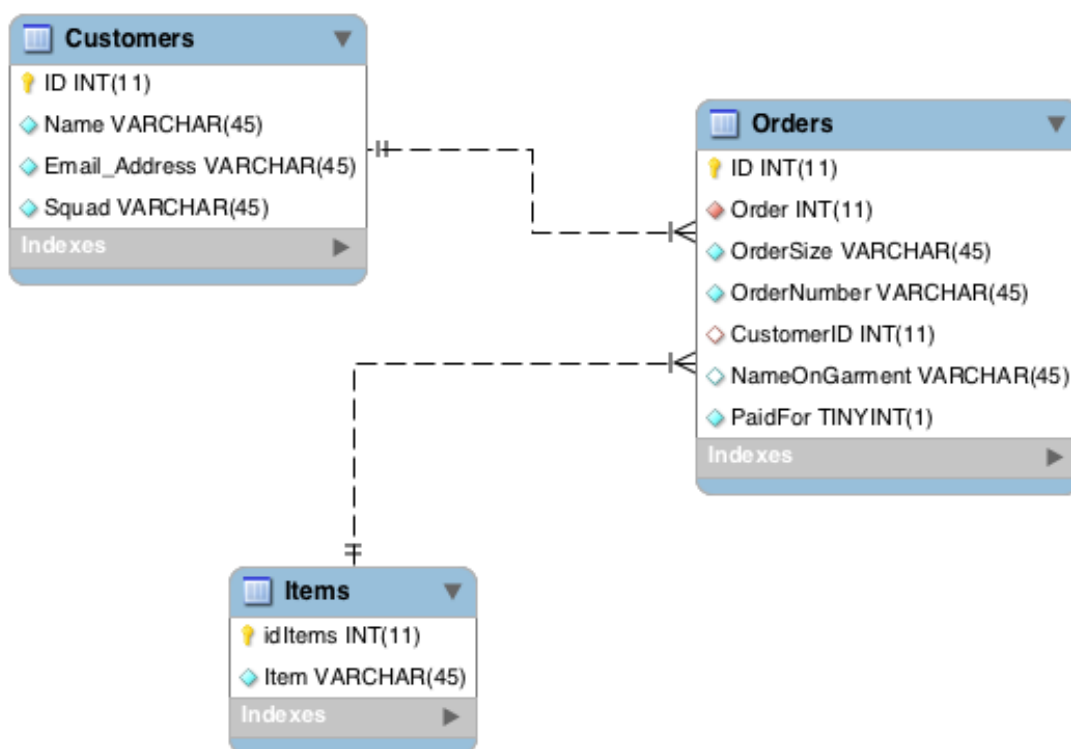


Figure 6: The proposed Entity Relationship diagram, exported from MySQLWorkbench

As can be seen in figure 6, the proposed EER will be fairly simple, with some foreign keys.

1.10 Entity Description

Customer(ID, Name, Email_Address, Squad)

Order(ID, Order (Foreign key from Items), OrderSize, OrderNumber, CustomerID (Foreign

key from Customer), NameOnGarment)
Items(idItems, Item)

1.11 Objectives for the Proposed System

The objectives of this system are to have an easy to use system that will run on minimal hardware. More specifically

1. It must have a well-structured (1NF or 2NF) database system that can be easily accessed.
2. It must have some sort of user interface allowing a person to enter the data then have the program handle the sorting.
3. It must store the data in a usable way (Most likely plain text).
4. It must have a search function, so the kit coordinator can search the database for certain attributes, like, for example, people who have and haven't paid for their order.
5. It must be lightweight enough to be run on a web server, such as the one hosting the club website.
6. It should have a way to automate the kit ordering procedure, by generating the order form that is sent off to the kit printers.
7. It should have a web-based front end so the orders for kit can be placed via the club website.
8. It should be able to send a mass email to all of the people who have ordered kit.
9. It could have an ability to monitor the email inbox of the kit email address so when the printer emails that the kit is ready it will send out a mass email automatically to everyone that has ordered.
10. It would be nice to have an integrated payments solution so everything could be done through the web interface, although due to time and complexity constraints this probably won't happen.

1.12 Potential Solutions

The potential solutions for this project are:

- An Excel spreadsheet using a VBA front-end application
- A VB.NET front-end application and a Microsoft Access database on the back-end

- A fully bespoke system programmed in Java and HTML
- A fully bespoke system programmed in C++

The respective advantages and disadvantages are seen in the next section.

1.13 Feasibility of Potential Solutions

- An Excel spreadsheet using a VBA front-end. Advantages:
 - It would be exceedingly easy to implement, as it just requires a spreadsheet and a small amount of programming to get up and running.
 - User already has experience using Excel and so would be able to use the system with minimal help.

Disadvantages:

- This solution would be offline only, so one of the main features of the proposed solutions, the online ordering system, would have to be left out, or radically changed.
 - Excel is a flat file database, which doesn't lead to good database design practices and also won't let me handle links between data.
 - The current system involves Excel so the new system would be too similar to the current system, and so wouldn't follow what the client wanted.
- A VB.NET front-end application and a Microsoft Access database back-end. Advantages:
 - Most of the system is pre-implemented so it would require a minimal amount of work to get up and running.
 - Unlike the Excel spreadsheet, data can be linked to and the database won't be just flat.

Disadvantages:

- I have no experience using VB.NET, which would add an unnecessary level of complexity to the project, as I would need to learn VB.NET as I worked.
 - I don't have convenient access to a copy of Microsoft Access, which would mean that I would need to purchase a copy.
 - The solution will be offline only, which will mean that orders wouldn't be able to be placed through the website.
- A bespoke coded system in Java. Advantages:
 - It would allow me to control and integrate everything from the beginning, as I would be creating most of the system from scratch.

- There would be no outlay, as I can use a free IDE to develop in, and Java which is itself free.
- The system won't be a flat file database, so I will be able to have links between data.
- Thanks to JDBC, the database will be searchable using SQL statements.
- The solution will be able to be hosted online and allow the kit coordinator to log in to the back end to view the database.

Disadvantages:

- Java can be unnecessarily complex to write a program in.
 - Java can be a massive resource hog, which means that it would be tricky to run on a low-budget web server.
 - I have minimal experience programming in HTML.
- A bespoke coded solution in C++. Advantages:
 - C++ is a fairly low-level language, so it's quite powerful
 - It has a large community, so if I get stuck with a problem then I can research solutions with little time wasted.

Disadvantages:

- I have absolutely no knowledge of programming using C++ so I would have to spend a lot of time learning how to code using it, which could be better spent programming the solution.
- Is apparently not very good for cross-platform applications, as a library is normally chosen which is platform specific, although due to my lack of knowledge of this language I don't know if this is true or not.

1.14 Justification of Chosen Solution

I will be making a bespoke solution in Java for this project, as I have far more experience in this language than any of the other solutions that I proposed. I also feel this will be the best as Java is incredibly versatile, and so can be run on any platform with minimal amounts of set-up. It does not require any purchase to be made, unlike Microsoft products which means that it can be made on a shoestring budget.

Also due to Java's ubiquity it has a vast number of resources that will be very helpful for referring to, if I get stuck with a certain section of this project. It has very good integration with SQL thanks to JDBC, which will mean that everything can be integrated into one complete package, rather than relying on solutions that could break if there is an update to the commercial software package being used. This integration will also mean that I will be able to make the database searchable via SQL queries, which will

definitely improve workflow as the kit organiser will be able to search for, for instance, people who haven't paid.

Although Java is normally seen as quite a resource intensive language, I think that, if the program stays small it will be relatively lightweight to run on the server. I will be able to deal with the relative complexity of Java as a language by ensuring that my design is logical and methodical.

While I do have minimal experience using HTML, I feel that this will be the easiest way to create a web form, as opposed to coding an applet in Java. This is due to the fact that I experimented with making a JApplet and decided that it was too convoluted for what I wanted to do and would have bogged me down, trying to get it to work.

2 DESIGN

2.1 Overall System Design

Table 5: Summary Table

Inputs	Processes
Swimmer's name Email Address Squad Name on Garments Order Size of the Garment Number of Garments ordered Whether the order has been paid for	Add new Order Add new customer Compile order to send off Edit order Show all orders to Kit Co-ordinator
Tables Storing Data	Outputs
Order Details Customer Details Items Available	Compiled Order Form Certain Orders via Search Function All orders in table to kit co-ordinator

2.2 Description of Modular System Structure

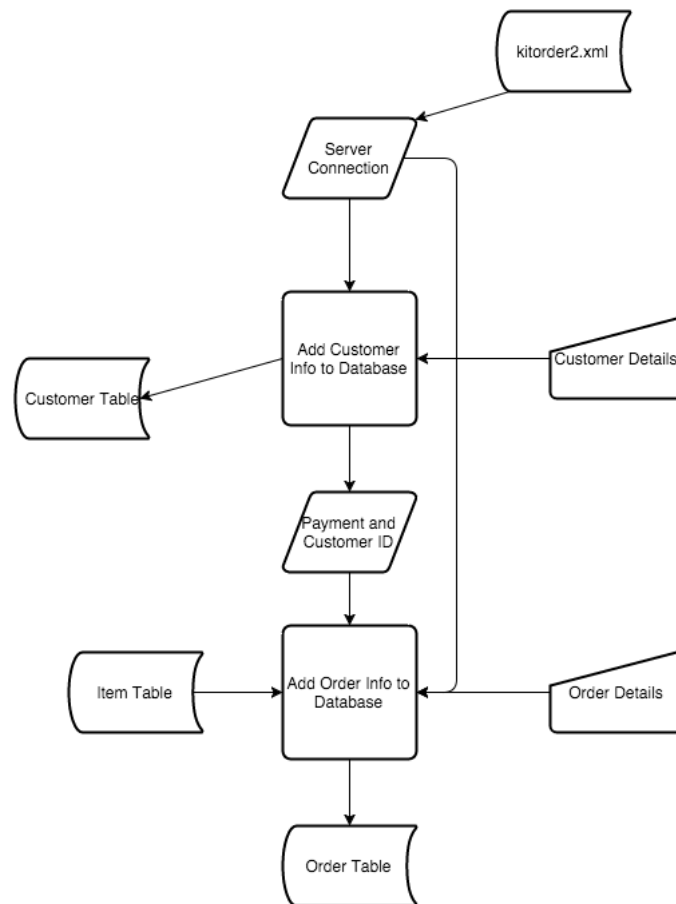


Figure 7: The part of the project facing the customer. This is all written in web-based technology

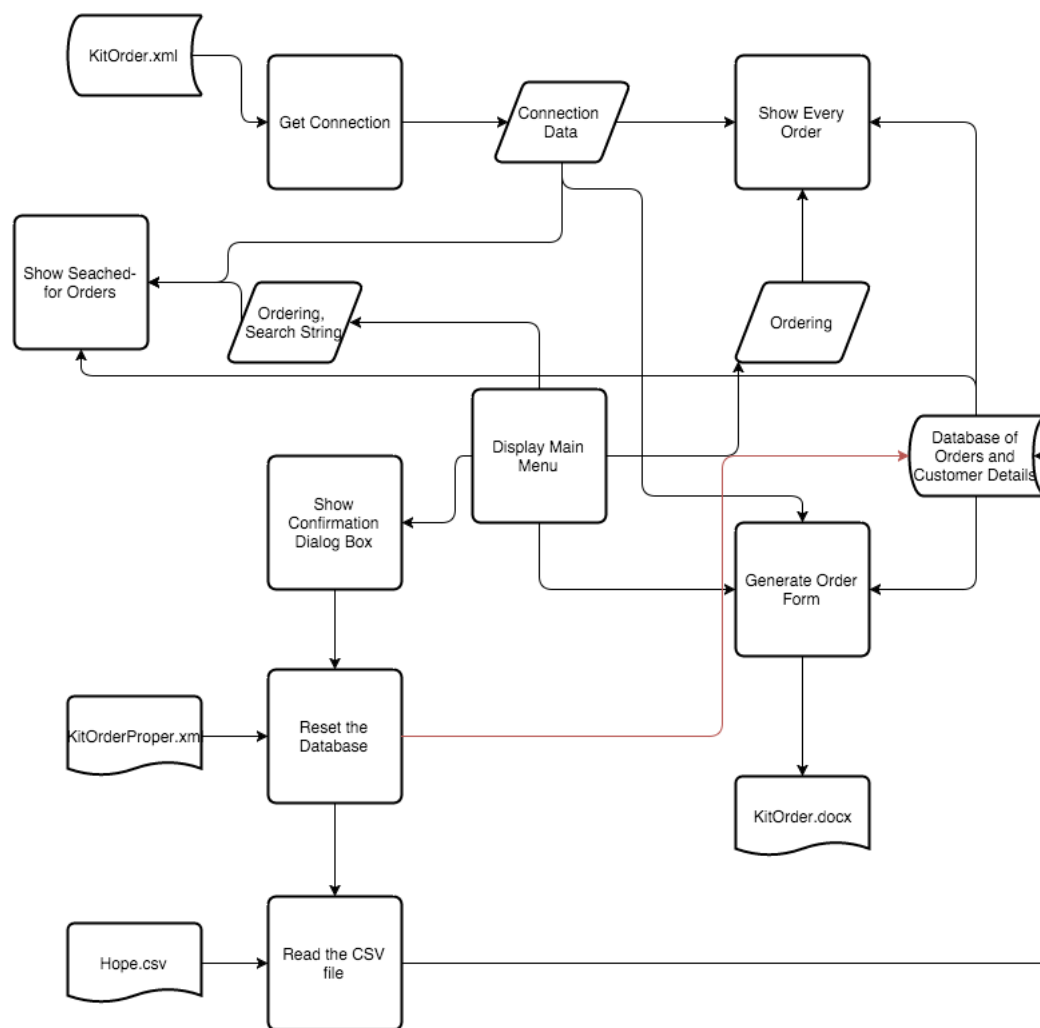


Figure 8: The part of the project facing the Kit Coordinator. This is all written in Java, except for the SQL script and the CSV

Ordr									
MainMenu			Get Connection			Remake Database			
Display	Enter Sort	Enter Search String	Open XML File	Get Connection Details	Return	Open SQL File	Run Script	Open CSV	Execute

Table 6: Part 1 of top down design

Ordr						
Show Contents with search				Show Contents		
Get Search	Get Ordering	Execute SQL Query	Show Contents	Get Ordering	Execute SQL Query	Show Contents

Table 7: Part 2 of top down design

2.3 Design Data Dictionary

For the purposes of this, B is the length of the field in bits. This is all assuming that the text is encoded in UTF-8, which allows for internationalisation.

Table 8: Customer Table Data Dictionary

Field Name	Purpose	Type	Typical length (B)	Example	Validation
ID	Differentiate Customer	Int	3 digits max (16)	2	It's generated by MySQL
Name	Stores Customer Name	String	40 (640)	Joe Bloggs	Not null
Email	Stores Email address	String	40 (640)	abc@abc.com	Not null, is an email address
Squad	Stores Swimmer's squad	String	3 (48)	Top	Not null

Table 9: Order Data Dictionary

Field Name	Purpose	Type	Typical length (B)	Example	Validation
ID	Stores the Order ID	Int	3 digits max (16)	3	Auto-generated by MySQL
Orders	Stores the Item ID	Int	1 (16)	4	Retrieved from Items
OrderSize	Shows the size of the item	String	5 characters max (80)	M	Not null
OrderNumber	Number of the item ordered	Int	1 (16)	1	Not null
CustomerID	Stores the ID of the customer	Int	3 digits max (16)	4	Retrieved from Customers
NameOnGarment	Stores string put on item	String	5 (80)	Pedro	Not null
PaidFor	Whether item has been paid for	TinyInt	1 (16)	0	Not null
PaymentMethod	Stores payment method	String	4 (64)	BACS	Not null

Table 10: Items Data Dictionary

Field Name	Purpose	Type	Typical length (B)	Example	Validation
idItems	Stores item ID	int	1 (16)	5	Auto-generated by MySQL
Items	Stores item name	String	7 (112)	Polo Shirt	Input on initialisation

Table 11: Connection XML Data Dictionary

dbms	Show database manager	String	5 (80)	mysql	Not null
------	-----------------------	--------	--------	-------	----------

2.4 Database Design

2.5 Identification of Storage Material and Format

2.6 Identification of Processes and Algorithms for Data Transformation

2.7 User Interface Design and Rationale

2.8 Planned Data Capture and Entry

2.9 Planned Valid Output Designs

2.10 Measures Planned for Security and Integrity of Data

To make sure the data input is valid, the program will make sure that the data entered has the expected hallmarks of the entered data. So, for instance, if an email address is entered, the program will check to see if the string has an @ symbol in it (done via HTML form and specifically the email field within it), and it will make sure that there is a domain name, although due to the manual element of the program, where everything is checked by eye before it is sent off, whether the domain is valid or not won't matter. The back-up strategy will involve a back-up whenever the website of the swimming club is backed up. This will ensure that the database backup is always as up to date as the website itself is.

2.11 Measures Planned for System Security

The database will be password protected, and this will be entered when someone attempts to connect to it. I am planning to do this using SSL, however I am not sure what this will be like to implement so it may have to be scaled back somewhat or possibly abandoned, depending on the complexity and whether I can obtain a signed certificate, which should be possible using Let'sEncrypt. No matter what happens the password will never be sent via plaintext, it will be salted and hashed on the client machine then sent to the server. To prevent SQL Injections, I will be making all of the SQL queries that the user can influence (for instance, the search query) Prepared Statements, which escape any special characters, hence foiling their dastardly plots to ruin my database.

2.12 Overall Test Strategy

To begin with I will be performing black box testing. In this stage of the testing I will be creating fake orders and seeing what happens when they are fed into the system. The expected output should be exactly the same as what is input, unless what is input is too long for the SQL field, or the input is invalid (for instance trying to input a SQL command into the text-boxes and so trying to perform a SQL injection. This should be prevented due to the use of prepared statements but I will test these just to make sure). For the web based part of this project this will involve testing in Google Chrome and Safari to see if the CSS works and it passes to the database correctly in both browsers.

This will be followed by white box testing where I will test every part of the system individually, and make sure I get the outputs that I expect. To achieve this I will insert print statements in parts of my code and then trigger the classes that they are in in order to trace their progress through the execution of the program.

Appendices

A SOURCE CODE APPENDIX

```

1 package kitordersystem;
2
3 import javafx.application.Application;
4 import javafx.application.Platform;
5 import javafx.event.ActionEvent;
6 import javafx.event.EventHandler;
7 import javafx.geometry.HPos;
8 import javafx.geometry.Insets;
9 import javafx.scene.Scene;
10 import javafx.scene.control.*;
11 import javafx.scene.layout.GridPane;
12 import javafx.scene.paint.Color;
13 import javafx.stage.Stage;
14 import org.xml.sax.SAXException;
15
16 import javax.xml.parsers.ParserConfigurationException;
17 import java.io.IOException;
18 import java.sql.SQLException;
19 import java.util.Optional;
20
21 /**
22  * This is, as the name suggests, the main menu for my programme. Where all
23  * the magic happens.
24  */
25 public class MainMenu extends Application {
26
27     public static String token;
28     public static String ordering;
29
30
31     public static void main(String[] args) throws SQLException, IOException {
32         launch(args);
33     }
34
35     /**
36      * Starts the stage, fairly self-evident
37      */
38     @Override
39     public void start(Stage primaryStage) {
40         primaryStage.setTitle("Ordr");
41         Button remakeButton = new Button("Remake");

```

```

42 Button dbViewButton = new Button("Contents");
43 Button emailButton = new Button("Generate");
44 Button searchButton = new Button("Search");
45 TextField searchField = new TextField();
46 ComboBox orderComboBox = new ComboBox();
47 Label label = new Label();
48 searchField.setPromptText("Search");
49 orderComboBox.getItems().addAll("Name A-Z", "Name Z-A", "Item",
50     "Order ID", "Squad", "Customer ID", "Payment Method");
51 remakeButton.setOnAction(new EventHandler<ActionEvent>() {
52     /**
53     *
54     * @param event
55     * This controls what happens when the reset button is clicked,
56     * calls the DBReset class followed by the CSVReader class, as
57     * long as the string input into the dialog box is "remake"
58     */
59     @Override
60     public void handle(ActionEvent event) {
61         TextInputDialog dialog = new TextInputDialog();
62         dialog.setTitle("Caution");
63         dialog.setContentText("Are you sure you want to do this? " +
64             "Enter 'remake' to remake the database");
65         Optional<String> result = dialog.showAndWait();
66         System.out.println(result);
67         String result1 = result.toString();
68         result1 = result1.replace("Optional[", "");
69         result1 = result1.replace("]", "");
70         System.out.println(result1);
71         if (result.isPresent() && result1.equals("remake")) {
72             System.out.println("Resetting");
73             DBReset reset = new DBReset();
74
75             try {
76                 CSVReader reader = new CSVReader();
77             } catch (IOException e) {
78
79                 e.printStackTrace();
80             } catch (SQLException e) {
81
82                 e.printStackTrace();

```

```

83         } catch (SAXException e) {
84
85             e.printStackTrace();
86         } catch (ParserConfigurationException e) {
87
88             e.printStackTrace();
89         }
90
91     } else {
92
93     }
94
95 }
96
97 });
98 dbViewButton.setOnAction(new EventHandler<ActionEvent>() {
99     /**
100      *
101      * @param event
102      * This controls what happens when the button to view everything
103      * in the database at that time is called, takes the value from the
104      * combo box for the order and passes it through to the class,
105      * which then runs a SQL statement with an order by
106      */
107     @Override
108     public void handle(ActionEvent event) {
109         ordering = orderComboBox.getValue().toString();
110         Platform.runLater(new Runnable() {
111             public void run() {
112                 try {
113                     new TableViewTest().start(new Stage());
114                 } catch (Exception e) {
115
116                     e.printStackTrace();
117                 }
118             }
119         });
120     }
121 }
122
123 });

```

```

124 emailButton.setOnAction(new EventHandler<ActionEvent>() {
125     /**
126     *
127     * @param event
128     * Calls the document writer class, which then outputs everything
129     * to a spreadsheet
130     */
131
132     @Override
133     public void handle(ActionEvent event) {
134         try {
135             DocWriter writer = new DocWriter();
136         } catch (Exception e) {
137             e.printStackTrace();
138         }
139     }
140 });
141 searchButton.setOnAction(new EventHandler<ActionEvent>() {
142     @Override
143     public void handle(ActionEvent event) {
144         token = searchField.getText();
145         ordering = orderComboBox.getValue().toString();
146         try {
147             new DBSearch().start(new Stage());
148         } catch (Exception e) {
149             e.printStackTrace();
150         }
151     }
152 });
153
154 GridPane grid = new GridPane();
155
156 grid.setHgap(10);
157 grid.setVgap(10);
158 grid.setPadding(new Insets(25, 25, 25, 25));
159
160 Scene scene = new Scene(grid, 300, 275);
161 primaryStage.setScene(scene);
162 grid.add(dbViewButton, 0, 0, 1, 1);
163 grid.add(remakeButton, 1, 0, 1, 1);
164 remakeButton.setTextFill(Color.RED);

```



```

165     grid.add(searchButton, 0, 1, 1, 1);
166     grid.add(emailButton, 1, 1, 1, 1);
167     grid.add(searchField, 0, 2, 3, 1);
168     grid.add(new Label("Order by: "), 0, 3, 1, 1);
169     grid.add(orderComboBox, 1, 3, 2, 1);
170     GridPane.setHalignment(remakeButton, HPos.CENTER);
171     GridPane.setHalignment(emailButton, HPos.CENTER);
172     GridPane.setHalignment(dbViewButton, HPos.CENTER);
173     GridPane.setHalignment(searchButton, HPos.CENTER);
174     GridPane.setHalignment(searchField, HPos.CENTER);
175     GridPane.setHalignment(orderComboBox, HPos.CENTER);
176
177
178     primaryStage.show();
179 }
180
181 }

1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools / Templates
4   * and open the template in the editor.
5   */
6  package kitordersystem;
7
8  import org.w3c.dom.Document;
9  import org.w3c.dom.Element;
10 import org.w3c.dom.NodeList;
11 import org.xml.sax.SAXException;
12
13 import javax.xml.parsers.DocumentBuilder;
14 import javax.xml.parsers.DocumentBuilderFactory;
15 import javax.xml.parsers.ParserConfigurationException;
16 import java.io.FileNotFoundException;
17 import java.io.IOException;
18 import java.sql.Connection;
19 import java.sql.DriverManager;
20 import java.sql.SQLException;
21 import java.util.InvalidPropertiesFormatException;
22 import java.util.Properties;
23
24 /**

```

```

25  * This class gets all the connection properties by opening an XML file (set
26  * as kitorder.xml) and then use the nodes in that to connect to the
27  * database itself
28  */
29  public class getConnection {
30
31      public String dbms;
32      public String dbName;
33      public String userName;
34      public String password;
35      private String serverName;
36      private int portNumber;
37
38      /**
39       * @return
40       * @throws SQLException
41       * @throws IOException
42       * @throws SAXException
43       * @throws ParserConfigurationException
44       */
45      public Connection getConnection() throws SQLException, IOException,
46          SAXException, ParserConfigurationException {
47          Connection conn = null;
48          this.setProperties("/Users/tsmoffat/kitordersystem/kitordersystem/" +
49              "kitordersystem/kitorder.xml");
50          String JDBC_URL = "jdbc:" + dbms + "://" + serverName + ":" + "3306"
51              + "/";
52          System.out.println("Connecting to: " + dbms + "://" + serverName +
53              ":3306");
54          conn = DriverManager.getConnection(JDBC_URL, userName, password);
55          return conn;
56      }
57
58      /**
59       * @param fileName - The file where the connection details can be found,
60       *                  it's easier to pass it through from elsewhere
61       * @throws FileNotFoundException
62       * @throws IOException
63       * @throws InvalidPropertiesFormatException
64       * @throws SAXException
65       * @throws ParserConfigurationException

```

```

66  */
67  public void setProperties(String fileName) throws FileNotFoundException,
68      IOException, InvalidPropertiesFormatException, SAXException,
69      ParserConfigurationException {
70      DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
71      DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
72      Document doc = dBuilder.parse(fileName);
73      doc.getDocumentElement().normalize();
74
75      NodeList nList = doc.getElementsByTagName("database");
76      Element database = (Element) nList.item(0);
77
78
79      Element connection = (Element) database.getElementsByTagName
80          ("connection").item(0);
81
82
83      this.dbms = connection.getElementsByTagName("dbms").item(0).
84          gettextContent();
85      this.dbName = connection.getElementsByTagName("database_name").item
86          (0).gettextContent();
87      this.userName = connection.getElementsByTagName("user_name").item(0).
88          gettextContent();
89      this.password = connection.getElementsByTagName("password").item(0).
90          gettextContent();
91      this.serverName = connection.getElementsByTagName("server_name").
92          item(0).gettextContent();
93      this.portNumber = Integer.parseInt(connection.getElementsByTagName
94          ("port_number").item(0).gettextContent());
95  }
96
97  }

```

```

1  package kitordersystem;
2
3  import javafx.application.Application;
4  import javafx.beans.property.SimpleStringProperty;
5
6  import javafx.beans.value.ObservableValue;
7  import javafx.collections.FXCollections;
8  import javafx.collections.ObservableList;
9  import javafx.scene.Scene;

```

```

10 import javafx.scene.control.TableColumn;
11
12 import javafx.scene.control.TableView;
13 import javafx.stage.Stage;
14 import javafx.util.Callback;
15
16
17 import java.io.IOException;
18 import java.sql.Connection;
19 import java.sql.ResultSet;
20 import java.sql.SQLException;
21
22 /**
23  * The class that shows everything in the database all at once. Excuse the name,
24  * I put it in as a test to see if it would work and then it worked so well that
25  * I just never changed it. This takes a global variable from the main menu for
26  * the ordering.
27  */
28 public class TableViewTest extends Application {
29
30     private TableView tableview = new TableView();
31
32     // MAIN EXECUTOR
33     public static void main(String[] args) {
34         launch(args);
35     }
36
37     // CONNECTION DATABASE
38
39     /**
40      * @throws SQLException
41      * @throws IOException
42      */
43     public void buildData() throws SQLException, IOException {
44         Connection c;
45         ObservableList<ObservableList> data = FXCollections.observableArrayList();
46         try {
47             c = new getConnection().getConnection();
48             String SQL = "USE mydb";
49             c.createStatement().executeQuery(SQL);
50             // SQL FOR SELECTING TABLES

```

```

51 String ordering = MainMenu.ordering;
52 String order;
53 switch (ordering){
54     case "Name A-Z":    order = "Name ASC";
55         break;
56     case "Name Z-A":    order = "Name DESC";
57         break;
58     case "Item":        order = "Orders";
59         break;
60     case "Order ID":    order = "ID";
61         break;
62     case "Squad":       order = "Squad";
63         break;
64     case "Customer ID": order = "CustomerID";
65         break;
66     case "Payment Method": order = "PaymentMethod";
67         break;
68     default:            order = "ID";
69 }
70
71 SQL = "select o.ID, o.CustomerID, c.Name, c.Email_Address, c.Squad," +
72       " o.Orders, o.OrderSize, o.OrderNumber, o.NameOnGarment," +
73       " o.PaidFor, o.PaymentMethod, i.Item from Orders o INNER" +
74       " JOIN Customers c ON o.CustomerID = c.ID INNER JOIN " +
75       "Items i ON i.idItems=o.Orders ORDER BY " + order +";";
76
77
78 // ResultSet
79
80 ResultSet rs = c.createStatement().executeQuery(SQL);
81
82 /**
83  * Gets column headings and adds them to the TableView dynamically
84  */
85
86 for (int i = 0; i < rs.getMetaData().getColumnCount(); i++) {
87
88     // We are using non property style for making dynamic table
89
90     final int j = i;
91

```

```

92     TableColumn col = new TableColumn(rs.getMetaData()
93         .getColumnName (i + 1));
94     col.setCellValueFactory(
95         new Callback<TableColumn
96             .CellDataFeatures<ObservableList , String>,
97             ObservableValue<String>>() {
98             public ObservableValue<String> call(TableColumn
99
↪     .CellDataFeatures<ObservableList, String> param) {
100         return new SimpleStringProperty(param.
101             getValue().get(j).toString());
102     }
103     });
104     tableview.getColumns().addAll(col);
105     System.out.println("Column [" + i + "] ");
106
107 }
108 /*
109  *
110  * Data added to ObservableList *
111  *
112  *****/
113 while (rs.next()) {
114     // Iterate Row
115     ObservableList<String> row = FXCollections.observableArrayList();
116     for (int i = 1; i <= rs.getMetaData().getColumnCount(); i++) {
117         // Iterate Column
118         row.add(rs.getString(i));
119     }
120     System.out.println("Row [1] added " + row);
121     data.add(row);
122 }
123 // FINALLY ADDED TO TableView
124 tableview.setItems(data);
125 } catch (Exception e) {
126     e.printStackTrace();
127     System.out.println("Error on Building Data");
128 }
129 }
130
131 /**

```

```

132     * Makes the whole
133     */
134     @Override
135     public void start(Stage stage) throws Exception {
136         // TableView
137         buildData();
138         // Main Scene
139         Scene scene = new Scene(tableview);
140         stage.setScene(scene);
141         stage.show();
142     }
143 }

1 package kitordersystem;
2
3
4 import org.apache.poi.ss.usermodel.Sheet;
5 import org.apache.poi.ss.usermodel.Workbook;
6 import org.apache.poi.xssf.usermodel.XSSFRow;
7 import org.apache.poi.xssf.usermodel.XSSFWorkbook;
8 import org.apache.poi.xwpf.usermodel.*;
9
10 import java.io.FileOutputStream;
11 import java.sql.Connection;
12 import java.sql.ResultSet;
13 import java.text.DateFormat;
14 import java.text.SimpleDateFormat;
15 import java.util.Date;
16 import java.util.Map;
17 import java.util.TreeMap;
18
19 /**
20  * Created by tsmoffat on 21/02/2016.
21  * A class to essentially dump the contents of the database to a spreadsheet
22  * in order to facilitate easy order sheet creation. This can be done as Word
23  * has an import from Excel option, although there is a possibility that the
24  * spreadsheet itself will be used as the order sheet
25  */
26 public class DocWriter {
27     public DocWriter() throws Exception{
28         XWPFDocument document = new XWPFDocument();
29         DateFormat df = new SimpleDateFormat("yyyy/MM/dd_HH.mm.ss");

```

Thomas Moffat, 51337, 4042

```

30     Date dateobj = new Date();
31
32
33     Connection conn = new getConnection().getConnection();
34     ResultSet rs;
35     String query;
36
37     Workbook wb = new XSSFWorkbook();
38
39     FileOutputStream fileOut = new FileOutputStream
40         ("Tilehurst_Order_Raw_" + df.format(dateobj) + ".xlsx");
41     for (int i=1; i<9; i++){
42         String stmt = "USE mydb";
43         conn.createStatement().executeQuery(stmt);
44         stmt="SELECT Item from Items where idItems = " + i;
45         rs = conn.createStatement().executeQuery(stmt);
46         rs.next();
47         String item = rs.getString("Item");
48         Sheet sheet1 = wb.createSheet(item);
49         XSSFRow row;
50         Map< String, Object[] > orderInfo =
51             new TreeMap< String, Object[] >();
52         orderInfo.put("1", new Object[] {item});
53         orderInfo.put("2", new Object[] {"Name", "Size", "Printed On"});
54         stmt = "SELECT c.Name, o.OrderSize, o.NameOnGarment FROM Orders o" +
55             " INNER JOIN Customers c on o.CustomerID " +
56             "= c.ID where o.Orders = " + i;
57         wb.write(fileOut);
58         while(rs.next()){
59             int j = 3;
60             String size = rs.getString("OrderSize");
61             String name = rs.getString("Name");
62             String nameOnBack = rs.getString("NameOnGarment");
63             String rowNum = Integer.toString(j);
64             orderInfo.put(rowNum, new Object[] {name, size, nameOnBack});
65             wb.write(fileOut);
66         }
67         wb.write(fileOut);
68     }
69     wb.write(fileOut);
70     fileOut.close();

```



```

71     wb.close();
72
73
74
75
76     XWPFTable table1 = document.createTable();
77     XWPFPParagraph paragraph1 = document.createParagraph();
78     String stmt = "USE mydb;";
79     conn.createStatement().executeQuery(stmt);
80     query = "SELECT Item from Items where idItems = 1";
81     rs = conn.createStatement().executeQuery(query);
82     rs.next();
83     String item1 = rs.getString("Item");
84     XWPFRun paraRun1 = paragraph1.createRun();
85     paraRun1.setBold(true);
86     paraRun1.setUnderline(UnderlinePatterns.SINGLE);
87     paragraph1.setAlignment(ParagraphAlignment.CENTER);
88     paraRun1.setText(item1);
89     System.out.println(1);
90     XWPFTableRow trOne1 = table1.getRow(0);
91     trOne1.getCell(0).setText("Name");
92     trOne1.addNewTableCell().setText("Size");
93     trOne1.addNewTableCell().setText("Name on Back");
94     query = "SELECT c.Name, o.OrderSize, o.NameOnGarment FROM Orders " +
95             "o INNER JOIN Customers c on o.CustomerID" +
96             " = c.ID where o.Orders = 1";
97     rs = conn.createStatement().executeQuery(query);
98
99     while(rs.next()) {
100         String size = rs.getString("OrderSize");
101         String name = rs.getString("Name");
102         String nameOnBack = rs.getString("NameOnGarment");
103         XWPFTableRow tRow = table1.createRow();
104         tRow.getCell(0).setText(name);
105         tRow.getCell(1).setText(size);
106         tRow.getCell(2).setText(nameOnBack);
107
108     }
109     return;
110
111

```

```

112     }
113
114 }

1  package kitordersystem;
2
3  import javafx.application.Application;
4  import javafx.beans.property.SimpleStringProperty;
5  import javafx.beans.value.ObservableValue;
6  import javafx.collections.FXCollections;
7  import javafx.collections.ObservableList;
8  import javafx.scene.Scene;
9  import javafx.scene.control.TableColumn;
10 import javafx.scene.control.TableView;
11 import javafx.stage.Stage;
12 import javafx.util.Callback;
13
14 import java.sql.Connection;
15 import java.sql.PreparedStatement;
16 import java.sql.ResultSet;
17
18 /**
19  * Created by tsmoffat on 10/02/2016.
20  * This is a class to search the database for a specific string. Works in
21  * much the same way as the connection part in TableViewTest but uses a user
22  * inputted search string as opposed to returning everything. Uses
23  * PreparedStatement to sanitise inputs. Yay.ø
24  */
25 public class DBSearch extends Application {
26
27     private ObservableList<ObservableList> data;
28     private javafx.scene.control.TableView tableview;
29
30     // MAIN EXECUTOR
31     public static void main(String[] args) {
32         launch(args);
33     }
34
35     public void Search() {
36         Connection c;
37         data = FXCollections.observableArrayList();
38         try {

```

```

39     c = new getConnection().getConnection();
40     String SQL = "USE mydb";
41     c.createStatement().executeQuery(SQL);
42     // SQL FOR SELECTING TABLES
43     String token = MainMenu.token;
44     String ordering = MainMenu.ordering;
45     String order;
46     switch (ordering){
47         case "Name A-Z":    order = "Name ASC";
48                             break;
49         case "Name Z-A":    order = "Name DESC";
50                             break;
51         case "Item":        order = "Order";
52                             break;
53         case "Order ID":    order = "ID";
54                             break;
55         case "Squad":       order = "Squad";
56                             break;
57         case "Customer ID": order = "CustomerID";
58                             break;
59         case "Payment Method": order = "PaymentMethod";
60                             break;
61         default:            order = "ID";
62     }
63     PreparedStatement statement = c.prepareStatement("select o.ID, o" +
64         ".CustomerID, c.Name, c.Email_Address, c.Squad, o.Orders," +
65         " o.OrderSize, o.OrderNumber, o.NameOnGarment, o.PaidFor, " +
66         "o.PaymentMethod, i.Item from Orders o INNER JOIN " +
67         "Customers c ON o.CustomerID = c.ID INNER JOIN Items i ON " +
68         " i.idItems=o.Order where o.ID like " +
69         "? or o.CustomerID like ? or c.Name like ? or c" +
70         ".Email_Address like ? or c.Squad like ? or o.Orders like" +
71         " ? or o.OrderSize like ? or o.OrderNumber like ? or o" +
72         ".NameOnGarment like ? or o.PaymentMethod like ? or i" +
73         ".Item like ? ORDER BY "+ order + ";");
74
75     statement.setString(1, "%" + token + "%");
76     statement.setString(2, "%" + token + "%");
77     statement.setString(3, "%" + token + "%");
78     statement.setString(4, "%" + token + "%");
79     statement.setString(5, "%" + token + "%");

```

```

80 statement.setString(6, "%" + token + "%");
81 statement.setString(7, "%" + token + "%");
82 statement.setString(8, "%" + token + "%");
83 statement.setString(9, "%" + token + "%");
84 statement.setString(10, "%" + token + "%");
85 statement.setString(11, "%" + token + "%");
86 System.out.println("Executing: " + statement);
87 ResultSet rs = statement.executeQuery();
88
89 /*
90  *
91  * TABLE COLUMN ADDED DYNAMICALLY *
92  *
93  * */
94
95 for (int i = 0; i < rs.getMetaData().getColumnCount(); i++) {
96
97     // We are using non property style for making dynamic table
98
99     final int j = i;
100
101     TableColumn col = new TableColumn(rs.getMetaData()
102         .getColumnName(i + 1));
103     col.setCellValueFactory(
104         new Callback<TableColumn.CellDataFeatures
105             <ObservableList, String>, ObservableValue
106             <String>>() {
107             public ObservableValue<String> call(TableColumn.
108
↪ CellDataFeatures<ObservableList, String> param) {
109                 return new SimpleStringProperty(param
110                     .getValue().get(j).toString());
111             }
112         });
113     tableview.getColumns().addAll(col);
114     System.out.println("Column [" + i + "] ");
115
116 }
117 /*
118  *
119  * Data added to ObservableList *

```

```

120         *
121         *****/
122         while (rs.next()) {
123             // Iterate Row
124             ObservableList<String> row = FXCollections.observableArrayList();
125             for (int i = 1; i <= rs.getMetaData().getColumnCount(); i++) {
126                 // Iterate Column
127                 row.add(rs.getString(i));
128             }
129             System.out.println("Row [1] added " + row);
130             data.add(row);
131         }
132         // FINALLY ADDED TO TableView
133         tableview.setItems(data);
134     } catch (Exception e) {
135         e.printStackTrace();
136         System.out.println("Error on Building Data");
137     }
138 }
139
140 /**
141  *
142  */
143 @Override
144 public void start(Stage stage) throws Exception {
145     // TableView
146     tableview = new TableView();
147     Search();
148     // Main Scene
149     Scene scene = new Scene(tableview);
150     stage.setScene(scene);
151     stage.show();
152 }
153 }

```

```

1 package kitordersystem;
2
3 import org.xml.sax.SAXException;
4
5 import javax.xml.parsers.ParserConfigurationException;
6 import java.io.BufferedReader;
7 import java.io.FileReader;

```

```

8 import java.io.IOException;
9 import java.sql.Connection;
10 import java.sql.SQLException;
11 import java.sql.Statement;
12
13 /**
14  * This is called when the database is reset, from the main menu, its only
15  * function is to repopulate the items database, which it does from a CSV
16  * file so that the contents of the table can be edited if the items ever
17  * change without having to recompile the whole program.
18  */
19 public class CSVReader {
20     /**
21      * @throws IOException
22      * @throws SQLException
23      * @throws SAXException
24      * @throws ParserConfigurationException
25      */
26
27     public CSVReader() throws IOException, SQLException, SAXException,
28         ParserConfigurationException {
29         BufferedReader reader = new BufferedReader(new FileReader("/Users/" +
30             "tsmoffat/kitordersystem/kitordersystem/kitordersystem/Hope" +
31             ".csv"));
32         Connection c = new getConnection().getConnection();
33         Statement st = c.createStatement();
34         st.executeUpdate("use mydb");
35         String line = "";
36         while ((line = reader.readLine()) != null) {
37             String[] item = line.trim().split(",");
38             // if you want to check either it contains some name
39             // index 0 is first name, index 1 is last name, index 2 is ID
40             st.executeUpdate("insert into Items (Item) values (\\"" + item[1]
41                 + "\\")");
42         }
43     }
44 }
45
46 }
47
48 package kitordersystem;
49
50

```

```

3 import org.apache.ibatis.jdbc.ScriptRunner;
4 import org.xml.sax.SAXException;
5
6 import javax.xml.parsers.ParserConfigurationException;
7 import java.io.BufferedReader;
8 import java.io.FileReader;
9 import java.io.IOException;
10 import java.io.Reader;
11 import java.sql.Connection;
12 import java.sql.SQLException;
13
14 /**
15  * How the program resets the database if everything goes wrong with it, or
16  * it gets too cluttered. Most of the heavy lifting is done through Mybatis,
17  * which is far more powerful than this project needs but it works very well.
18  */
19 public class DBReset {
20
21     public DBReset(){
22         String SQLFilePath = "/Users/tsmoffat/kitordersystem/kitordersystem/" +
23             "kitordersystem/KitOrderProper.sql";
24         try{
25             System.out.println("RESETTING");
26             Connection conn = new getConnection().getConnection();
27             ScriptRunner sr = new ScriptRunner(conn);
28             Reader reader = new BufferedReader(new FileReader(SQLFilePath));
29             sr.runScript(reader);
30         } catch (SAXException | SQLException | ParserConfigurationException |
31             IOException e) {
32             e.printStackTrace();
33         } catch (Exception e){
34             System.err.println("Failed to execute " + SQLFilePath + ". The " +
35                 "error is " + e.getMessage());
36         }
37     }
38 }
39
40 }
41
42 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
43 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
44 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

```

```

4 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT;
5 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS;
6 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION;
7 SET NAMES utf8;
8 SET @OLD_TIME_ZONE=@@TIME_ZONE;
9 SET TIME_ZONE='+00:00';
10 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
11 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
12 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO';
13 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0;
14 CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8;
15 USE `mydb`;
16 DROP TABLE IF EXISTS `mydb`.`Customers`;
17 CREATE TABLE IF NOT EXISTS `mydb`.`Customers` (
18   `ID` INT NOT NULL AUTO_INCREMENT,
19   `Name` VARCHAR(45) NOT NULL,
20   `Email_Address` VARCHAR(45) NOT NULL,
21   `Squad` VARCHAR(45) NOT NULL,
22   PRIMARY KEY (`ID`),
23   UNIQUE INDEX `ID_UNIQUE` (`ID` ASC))
24 ENGINE = InnoDB;
25 DROP TABLE IF EXISTS `mydb`.`Items`;
26 CREATE TABLE IF NOT EXISTS `mydb`.`Items` (
27   `idItems` INT NOT NULL AUTO_INCREMENT,
28   `Item` VARCHAR(45) NOT NULL,
29   PRIMARY KEY (`idItems`),
30   UNIQUE INDEX `idItems_UNIQUE` (`idItems` ASC))
31 ENGINE = InnoDB;
32 DROP TABLE IF EXISTS `mydb`.`Orders`;
33 CREATE TABLE IF NOT EXISTS `mydb`.`Orders` (
34   `ID` INT NOT NULL AUTO_INCREMENT,
35   `Orders` INT NOT NULL,
36   `OrderSize` VARCHAR(45) NOT NULL,
37   `OrderNumber` VARCHAR(45) NOT NULL,
38   `CustomerID` INT NULL,
39   `NameOnGarment` VARCHAR(45) NULL,
40   `PaidFor` TINYINT(1) NOT NULL,
41   `PaymentMethod` VARCHAR(45) NOT NULL,
42   PRIMARY KEY (`ID`),
43   INDEX `Order1_idx` (`Orders` ASC),
44   INDEX `CustomerID_idx` (`CustomerID` ASC),

```



```

45     CONSTRAINT `Order1`
46     FOREIGN KEY (`Orders`)
47     REFERENCES `mydb`.`Items` (`idItems`)
48     ON DELETE NO ACTION
49     ON UPDATE NO ACTION,
50     CONSTRAINT `CustomerID`
51     FOREIGN KEY (`CustomerID`)
52     REFERENCES `mydb`.`Customers` (`ID`)
53     ON DELETE NO ACTION
54     ON UPDATE CASCADE)
55 ENGINE = InnoDB;
56 SET SQL_MODE=@OLD_SQL_MODE;
57 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
58 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
5      <link rel="stylesheet" type="text/css" href="styling.css" />
6      <meta char-set="utf-8"/>
7      <title>Customer Input</title>
8    </head>
9    <body>
10   <div>
11     <form id="customerform" method="post" >
12       Name:<br>
13       <input type="text" name="name"><br>
14       Email:<br/>
15       <input type="email" name="email"/><br/>
16       Squad:<br/>
17       <select name="squad">
18         <option value="Dev">Dev</option>
19         <option value="Jag">Jag</option>
20         <option value="Top">Top</option>
21       </select><br/>
22       Choose payment method <br />
23       <select name="paymethod">
24         <option value="BACS">
25           BACS
26         </option>
27         <option value="Cheque">

```

```

28         Cheque
29     </option>
30     <option value="Cash">
31         Cash
32     </option>
33 </select><br/>
34 <input type="submit" value="Submit"/><br/>
35
36 </form>
37 </div>
38 <script type="text/javascript">
39     function passtophp(e) {
40         e.preventDefault()
41
42         var str = $(this).serialize();
43         $.ajax('dbcustomerupdate.php', str, function (result) {
44             alert(result)
45         })
46
47         return false;
48     }
49
50     $('#customerform').submit(function (e) {
51         e.preventDefault()
52
53         var str = $(this).serialize();
54
55         $.post('dbcustomerupdate.php', str, function (result) {
56             alert(result)
57         })
58         window.location="./WebInputOrder.html"
59         return false;
60     })
61 </script>
62 </body>
63 </html>
64
65 <!DOCTYPE html>
66 <html>
67 <head>
68     <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
69     <link rel="stylesheet" type="text/css" href="styling.css" />

```

```

6     <meta char-set="utf-8" />
7     <title>Order Input</title>
8 </head>
9 <body>
10    <div>
11        Please enter in your items one at a time<br />
12        Item List:<br />
13        Poolside Shirt: £12<br />
14        Polo Shirt: £12<br />
15        Over-the-head Hoodie: £17<br />
16        Zippy Hoodie: £20<br />
17        Rucksack: £22<br />
18        Holdall: £27<br />
19        Hat: £6<br />
20        Shorts: £4<br />
21        <form method="post" id="orderform">
22            Item:<br />
23            <select name="item">
24                <option value="Poolside Shirt">
25                    Poolside Shirt
26                </option>
27                <option value="Polo Shirt">Polo Shirt</option>
28
29                <option value="Hoodie">
30                    Hoodie
31                </option>
32                <option value="Zippy Hoodie">
33                    Zippy Hoodie
34                </option>
35                <option vaue="Hat">
36                    Hat
37                </option>
38                <option vale="Shorts">
39                    Shorts
40                </option>
41                <option value="Backpack">
42                    Backpack
43                </option>
44                <option value="Holdall">
45                    Holdall
46                </option>

```

```

47 </select><br />
48 Quantity:<br />
49 <input type="number" name="quantity" min="1" max = "5"/><br />
50 Size:<br/>
51 <select name="size">
52   <option value="9-11"><!--Inside job-->
53     Size 9-11
54   </option>
55   <option value="12-13">
56     Size 12-13
57   </option>1
58   <option value="S">
59     Small
60   </option>
61   <option value="M">
62     Medium
63   </option>
64   <option value="L">
65     Large
66   </option>
67   <option value="Misc">
68     Misc (for bags, hats and shorts)
69   </option>
70
71 </select><br />
72 Please enter the name/letters to be printed on your item<br />
73 <input type="text" name="NameOnBack" /><br />
74 Have you paid for this item?<br />
75 <input type="radio" name="haspaid" value="1" checked />Yes<br />
76 <input type="radio" name="haspaid" value="0" />No<br />
77 <input type="submit" value="Submit"/><br/>
78 </form>
79 </div>
80 <script type="text/javascript">
81   function passtophp(e) {
82     e.preventDefault()
83
84     var str = $(this).serialize();
85     $.ajax('dborderupdate.php', str, function (result) {
86       alert(result)
87     })

```

```

88
89     return false;
90 }
91
92 $('#orderform').submit(function (e) {
93     e.preventDefault()
94
95     var str = $(this).serialize();
96
97     $.post('dborderupdate.php', str, function (result) {
98         alert(result)
99     })
100
101     return false;
102 })
103 </script>
104 </body>
105 </html>

```



```

1  input[type=text] {
2      padding: 12px 20px;
3      margin: 8px 0;
4      box-sizing: border-box;
5      font-size: 1.2em;
6      width: 100%
7  }
8  input[type=email] {
9      padding: 12px 20px;
10     margin: 8px 0;
11     box-sizing: border-box;
12     font-size: 1.2em;
13     width: 100%
14 }
15 input[type=text]:focus {
16     border: 3px solid #555;
17     outline: none;
18 }
19 input[type=email]:focus {
20     border: 3px solid #555;
21     outline: none;
22 }
23 select {

```

```

24     padding: 16px 20px;
25     border: none;
26     border-radius: 4px;
27     background-color: #f1f1f1;
28     font-size: 1.2em;
29     width: 100%;
30 }
31 input[type=submit] {
32     background-color: #4CAF50;
33     border: none;
34     color: white;
35     padding: 32px, 64px;
36     text-decoration: none;
37     margin: 4px 2px;
38     cursor: pointer;
39     font-size: 1.5em;
40     width: 100%;
41 }
42 div {
43     font-family: "Arial", Helvetica, sans-serif;
44     font-size: 1em;
45
46
47 }
48 input[type=number]{
49     padding: 12px 20px;
50     margin: auto;
51     box-sizing: border-box;
52     font-size: 1.2em;
53     width: 50%
54 }
55 input[type=number]:focus {
56     border: 3px solid #555;
57     outline: none;
58 }

```

```

1  <?php
2  session_start();
3  if (file_exists('kitorder2.xml')) {
4      $xml = simplexml_load_file('kitorder2.xml');
5      $dbms = $xml->connection->dbms;
6      $dbname = $xml->connection->database_name;

```

Thomas Moffat, 51337, 4042

```

7  $user_name = $xml->connection->user_name;
8  $password = $xml->connection->password;
9  $port_numberraw = $xml->connection->port_number;
10 $port_number = (int) $port_numberraw;
11 $servername = $xml->connection->server_name;
12 $conn = new mysqli($servername, $user_name, $password, $dbname, $port_number);
13 if ($conn->connect_error){
14     die("Connection failed: " . $conn->connect_error);
15 }
16
17
18 $name = test_input($_POST['name']);
19 $email = test_input($_POST['email']);
20 $squad = test_input($_POST['squad']);
21
22 $paymethod = test_input($_POST['paymethod']);
23
24 $_SESSION["paymethod"]=$paymethod;
25 $stmt = $conn->prepare("INSERT INTO Customers (Name, Email_Address, Squad) VALUES (?,
↪  ?, ?);");
26 $stmt->bind_param("sss", $name, $email, $squad);
27 $stmt->execute();
28
29 $sql = $conn->prepare("SELECT MAX(ID) FROM Customers WHERE Name = ?;");
30 $sql->bind_param("s", $name);
31 $sql->execute();
32 $result = $sql->get_result()->fetch_row()[0];
33 $_SESSION["ID"] = $result;
34
35
36 } else {
37     exit('Failed to open kitorder2.xml.');
38 }
39 function test_input($data){
40     $data=trim($data);
41     $data= stripslashes($data);
42     $data=htmlspecialchars($data);
43     return $data;
44 }
45 ?>

```

```

1  <?php
2  // The file test.xml contains an XML document with a root element
3  // and at least an element [/root]/title.
4  session_start();
5  error_reporting(E_ALL);
6  ini_set('display_errors', 1);
7  if (file_exists('kitorder2.xml')) {
8      $xml = simplexml_load_file('kitorder2.xml');
9      $dbms = $xml->connection->dbms;
10     $dbname = $xml->connection->database_name;
11     $user_name = $xml->connection->user_name;
12     $password = $xml->connection->password;
13     $port_numbreraw = $xml->connection->port_number;
14     $port_number = (int) $port_numbreraw;
15     $servername = $xml->connection->server_name;
16     $conn = new mysqli($servername, $user_name, $password, $dbname, $port_number);
17     if ($conn->connect_error){
18         die("Connection failed: " . $conn->connect_error);
19     }
20
21     if ($_SERVER["REQUEST_METHOD"] == "POST") {
22
23         $item = $_POST['item'];
24         $quantityraw = $_POST['quantity'];
25         $quantity = $quantityraw;
26         $size = $_POST['size'];
27         $nameonback = $_POST['NameOnBack'];
28         $haspaidraw = $_POST['haspaid'];
29         $haspaid = $haspaidraw;
30         $paymethod = $_SESSION['paymethod'];
31         $stmt = $conn->prepare("SELECT idItems from Items where Item = ?;");
32         $stmt->bind_param("s", $item);
33         $stmt->execute();
34         $idraw=$_SESSION['ID'];
35         $id = $idraw;
36         $itemidraw = $stmt->get_result()->fetch_row()[0];
37         $itemid = $itemidraw;
38         var_dump($item, $quantity, $size, $nameonback, $haspaid, $paymethod, $itemid, $id);
39         $stmt = $conn->prepare("INSERT INTO Orders (`Orders`, `OrderSize`, `OrderNumber`,
↪ `CustomerID`, `NameOnGarment`, `PaidFor`, `PaymentMethod`) values (?,?,?,?,?,?,?);");

```



```
40      $stmt->bind_param('isiisis', $itemid, $size, $quantity, $id, $nameonback, $haspaid,  
↪    $paymethod);  
41      $stmt->execute();  
42  
43  }  
44  
45  } else {  
46      exit('Failed to open kitorder2.xml.');
```

47 }

```
48  function test_input($data){  
49      $data=trim($data);  
50      $data= stripslashes($data);  
51      $data=htmlspecialchars($data);  
52      return $data;  
53  }  
54  ?>
```

REFERENCES

Thomas Moffat, 51337, 4042