ORDR: An Sports Club kit-ordering system

Thomas Moffat

Candidate Number: 4042

Centre Number: 51337

Report generated using LaTeX

CONTENTS

LIST OF FIGURES

LIST OF TABLES

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?

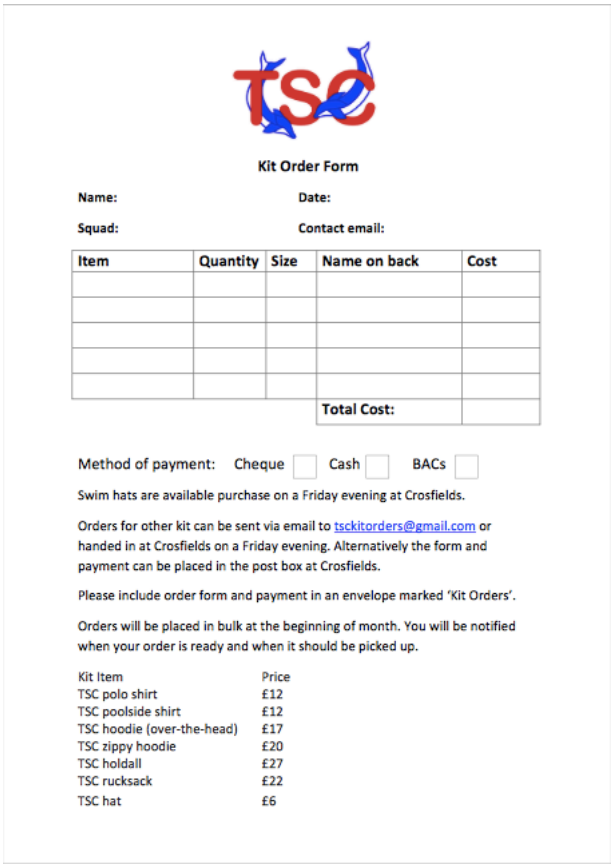
  Security for email addresses.

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:

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



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

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 an 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

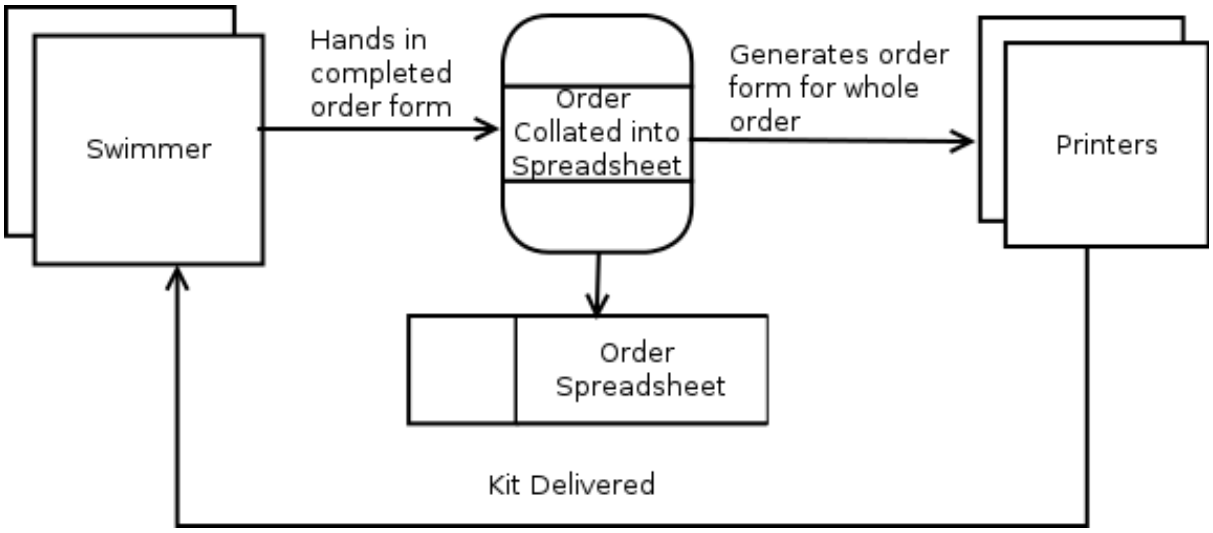


Figure 2: A level 0 data flow diagram of the current system

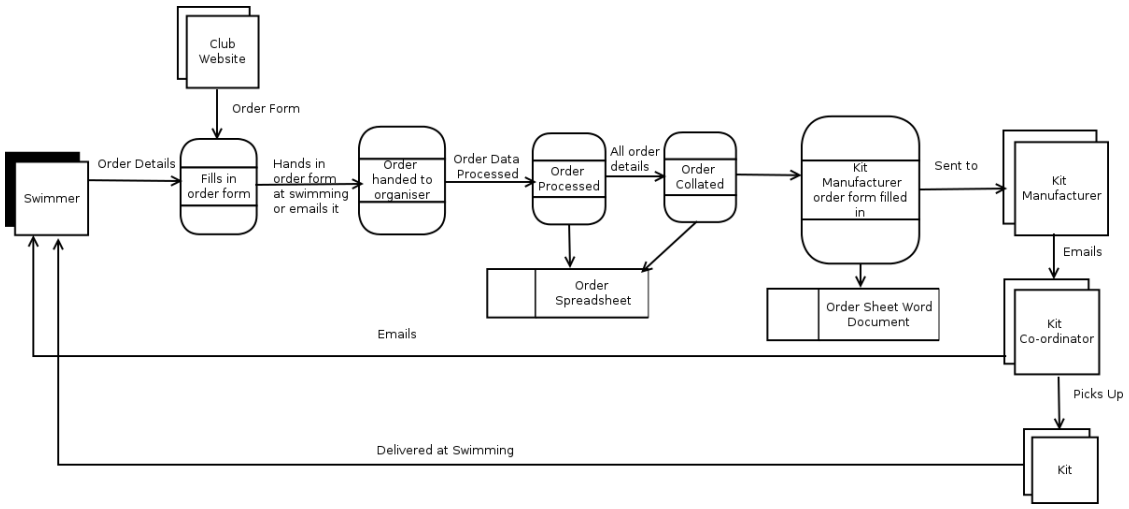


Figure 3: A level 1 data flow diagram of the current system

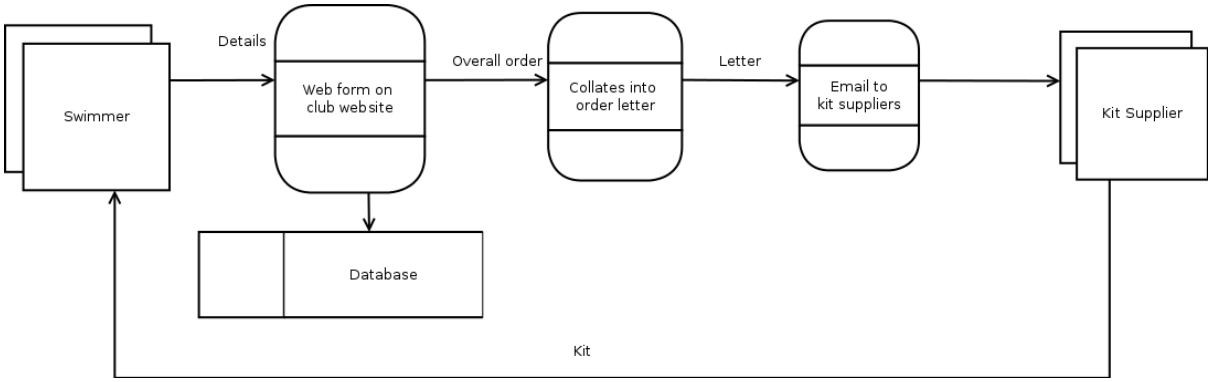


Figure 4: A level 0 data flow diagram of the proposed system

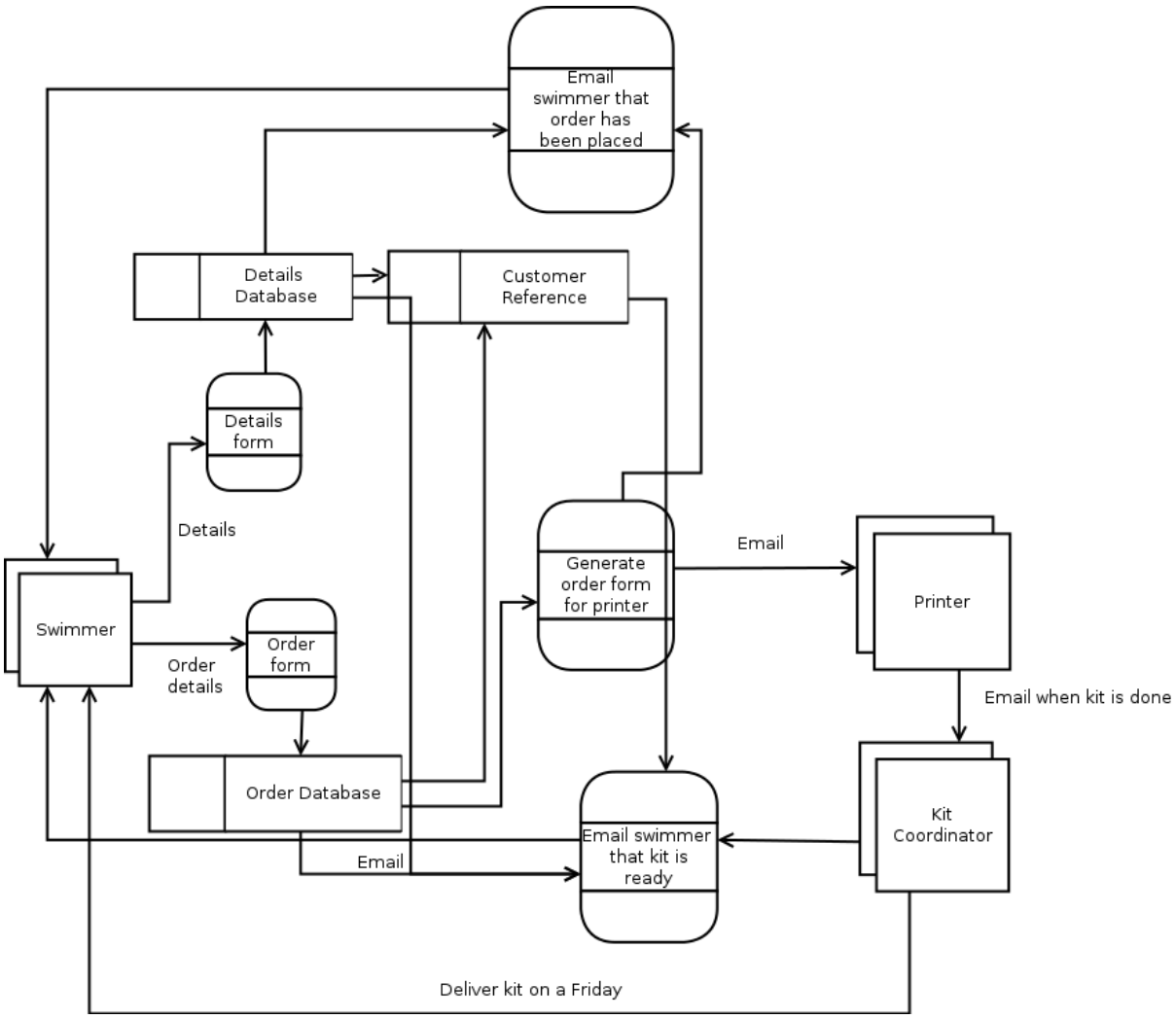


Figure 5: A level 1 data flow diagram of the proposed system

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

The current entity relationship model can be seen in Figure 6, and this is a rather abstract view of what requires a lot of paper and back and forth.

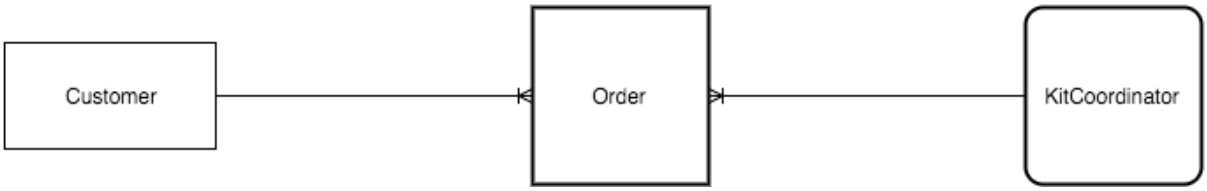


Figure 6: The current Entity Relationship Diagram

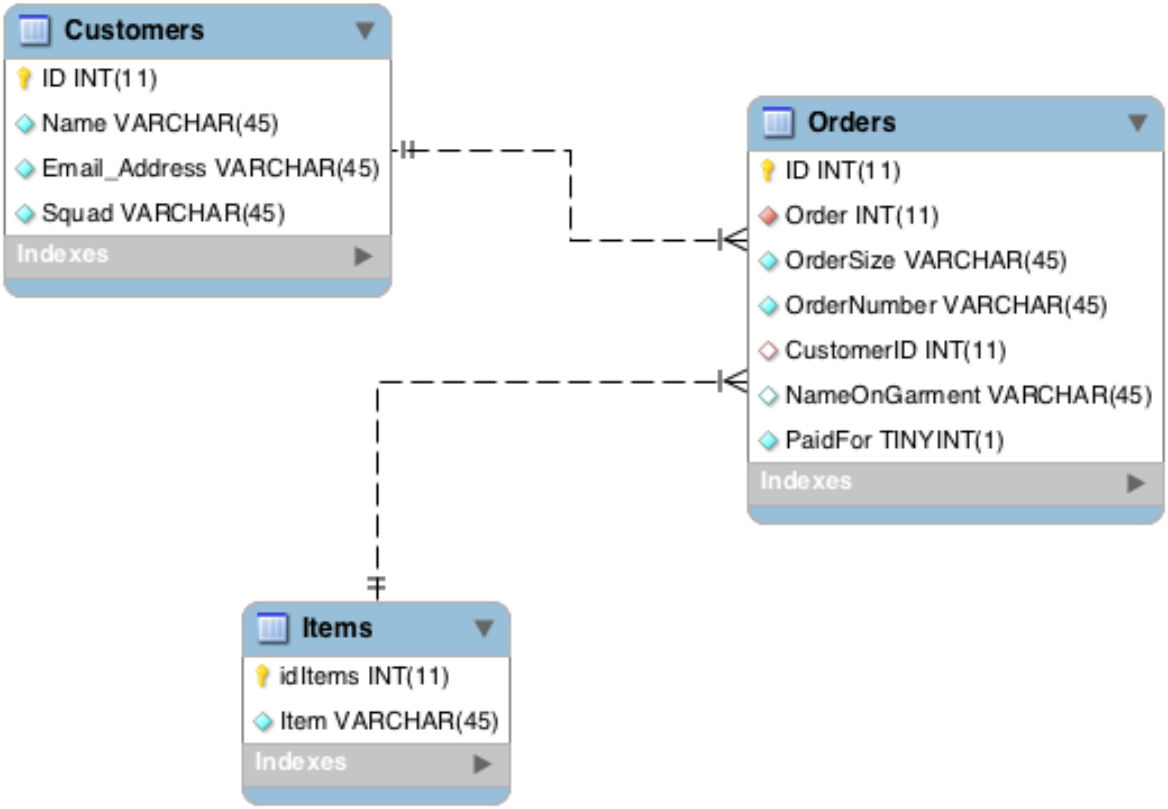


Figure 7: The proposed Entity Relationship diagram, exported from MySQLWorkbench

As can be seen in figure 7, 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 could 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 8 shows the process when the web page with the order form is filled out and submitted. This allows the program to remain streamlined in that the user just has to click submit and everything is hidden from them and put into the database automatically. This then makes it easier on the Kit Coordinator as they don't have to deal with any more paper, they can use the rest of the program to access the database and perform administrative tasks on it such as generating the contents of the database into a spreadsheet (more on that later)

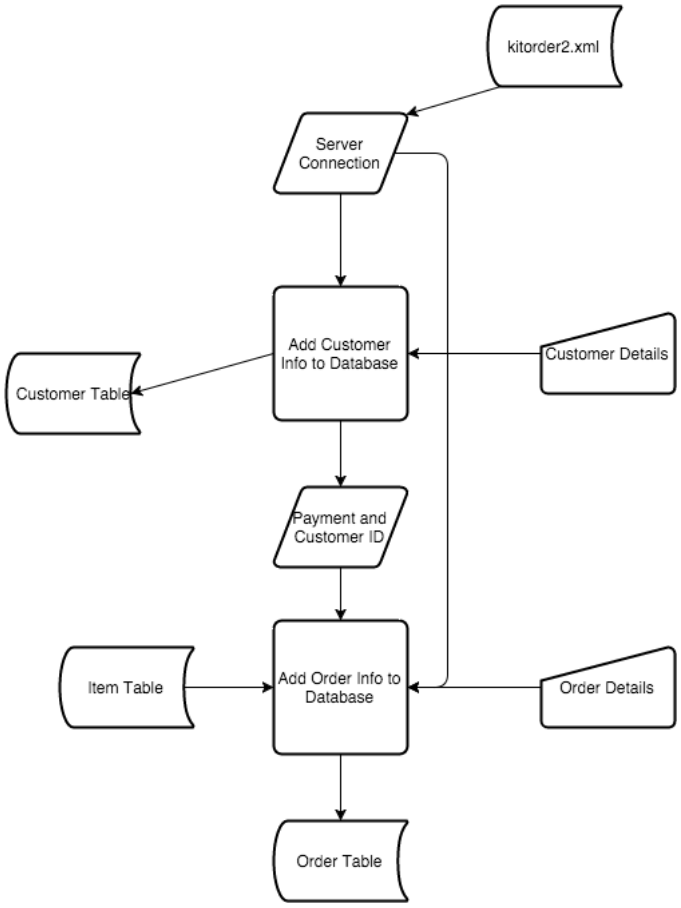


Figure 8: The part of the project facing the customer. This is all written in web-based technology

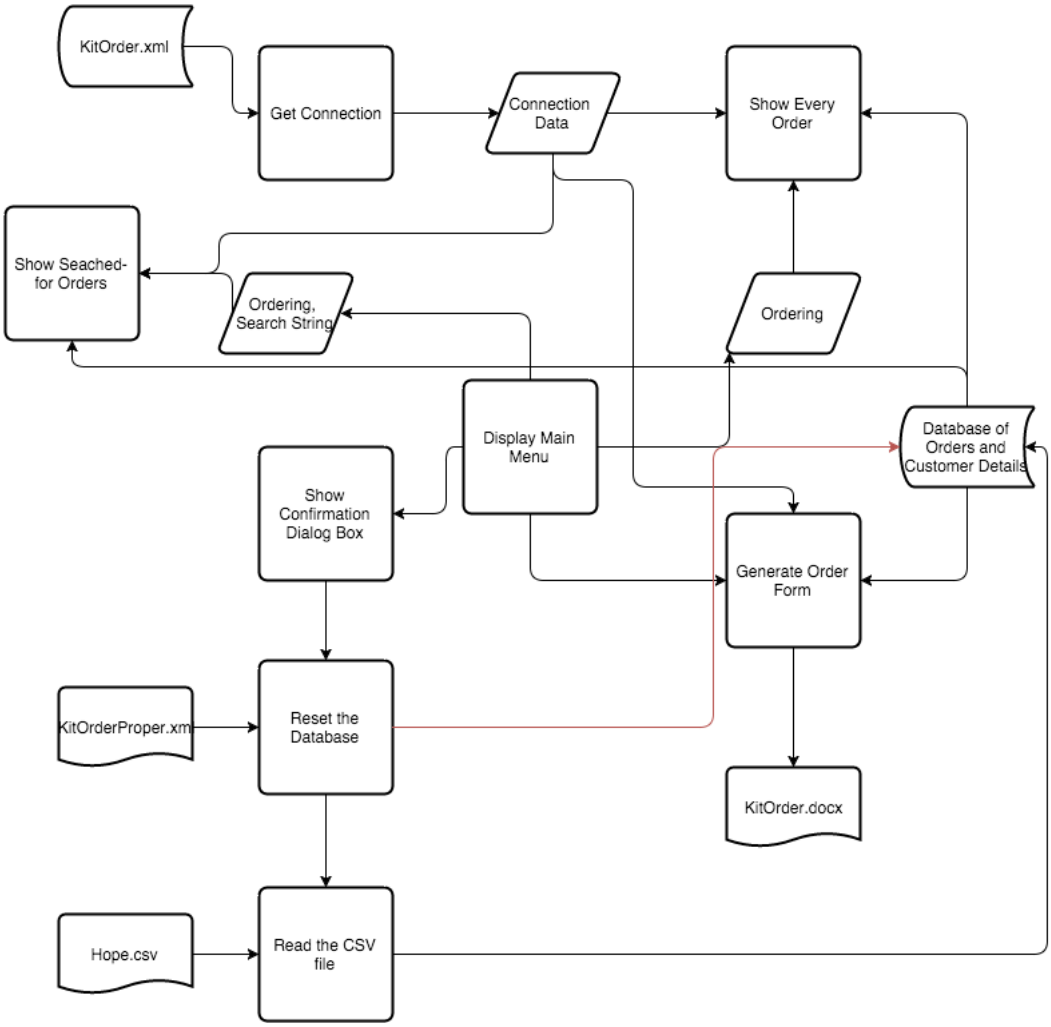


Figure 9: The part of the project facing the Kit Coordinator. This is all written in Java, except for the SQL script and the CSV

Due to the relative complexity of the Java end of the project, the functions are given in a slightly more easy to follow format in tables 6 and 7, which shows the main functions as you drill down into the program. This is marginally easier to follow than figure 9 due to the fact it follows a slightly more logical layout, although figure 9 gives a more complete view of what is going on in the program as a whole. These diagrams are most useful when used together for the design as figure 9 gives a better view of the inputs and outputs of the programs while tables 6 and 7 show the structure of the program better.

The main feature here is the function to generate an order form that outputs as a spreadsheet, as this allows the Kit Coordinator to click a button and have a correctly formatted order form to be generated and then sent off to the printers. This saves time and effort, as the Kit Coordinator doesn't have to spend large amounts of time transcribing the order forms by hand into an Excel spreadsheet and then sending that off to the printers. Instead the spreadsheet can be generated. The reset function can be useful if something goes horribly wrong with the

database, such as someone gains access to it and defaces it, the Kit Coordinator can just click a button and the database is remade completely fresh and blank.

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. The only validation method given for most of these is that they aren't null. They're just required to be present. This is because the validation will be performed using HTML forms, and the data types that it allows. So the email address box on the form will have an email type, etc. The items of kit for the order form will be in a drop-down menu so there isn't really any way that they can be altered, at least not in normal use. This is similarly true of the Squad and the Payment Method. The number of garments will be chosen with an upper limit of one and a lower limit of five, so there's no real way to have strange values, such as -1. The PaidFor option is just a radio on the webpage, so there shouldn't be any strange values. In this case MySQL won't accept the value anyway if it isn't 0 or 1, as it is a boolean.

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 8 shows the columns in the Customer table of the database, with all the datatypes etc. All the strings in this table are set to VARCHAR(45), i.e. if the strings are longer than 45 characters, they aren't accepted by the database and the query isn't updated. This length was chosen as it will be very unlikely to find a name or email address longer than that length in every day testing, and if a fringe case like that does come up then the database can be altered to allow longer email addresses to be used. As an example, Chargoggagoggmanchauggagoggchaubunagungamaugg is 45 characters, which is a very long place name and there will be almost no email addresses longer than this, unless said email address was chosen as a joke. This is why the typical length is set to 40 in the case of both the Name and the Email Address column, as this length is considered to be a fringe case so that data usage can be calculated

as if every single name and email address was that length and a worse case scenario can be found.

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	Bool	1 (1)	0	Not null
PaymentMethod	Stores payment method	String	4 (64)	BACS	Not null

Table 9 shows the columns within the Orders table of the database. While the strings are all set to VARCHAR(45) again, most of the fields will be changed using drop down boxes, so the length of the field is overkill. This is to facilitate easy changing in the future if the length of some option suddenly became far longer. The only string in this table that isn't chosen from a drop-down menu is the NameOnGarment column, which has a 'social limit' in that most people have nicknames on the backs of their T-Shirts or Hoodies that don't extend to any more than 8 characters, and normally fewer, and people have their initials on their bags, which is normally four or fewer characters. So the VARCHAR(45) field here is still overkill, however it is still there if the customer wants the option to have a fabulously long name on the back of their T-Shirt (the bags don't have enough room to have anything other than initials printed on them).

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 **??** shows the two columns from the Items table in the database. This is never directly altered except for when the remake button on the program is clicked, and the program accesses a CSV file with the values to be inserted into the table, and then populates this table. This table could be set to read-only to prevent vandalism or mistakes, however this could raise problems further down the road so will not be implemented during the initial stages of this program.

Table 11: Connection XML Data Dictionary

Field Name	Purpose	Type	Typical length (B)	Example	Validation
dbms	Show database manager	String	5 (80)	mysql	Not null
database_name	shows database name	String	4 (64)	mydb	Not null
user_name	stores the user name	String	5 (80)	root	Not null
password	stores password	String	8 (128)	cheetah	Not null
port_number	stores the port number	int	4 (64)	3306	Not null
server_name	stores the server IP	String	15 (240)	localhost	Not null

The last table, table 11 is each of the nodes in the XML files kitorder.xml and kitorder2.xml. These are, I feel, self-explanatory, in that they do exactly what the node names say. They are retrieved when they are needed by a program. Note that there are two files that read exactly the same thing. The idea is that one will be kept locally with the java program (kitorder.xml) and the other (kitorder2.xml) will be loaded onto the web server, so they bother know how to find the database at all times. This came about during testing of the XML parsers in Java and PHP, and the XML parser in Java required the elements to be declared, while the XML parser in PHP kicked out an error when this was altered. This is very useful however as it allows the connection settings to be altered independently.

2.4 Database Design

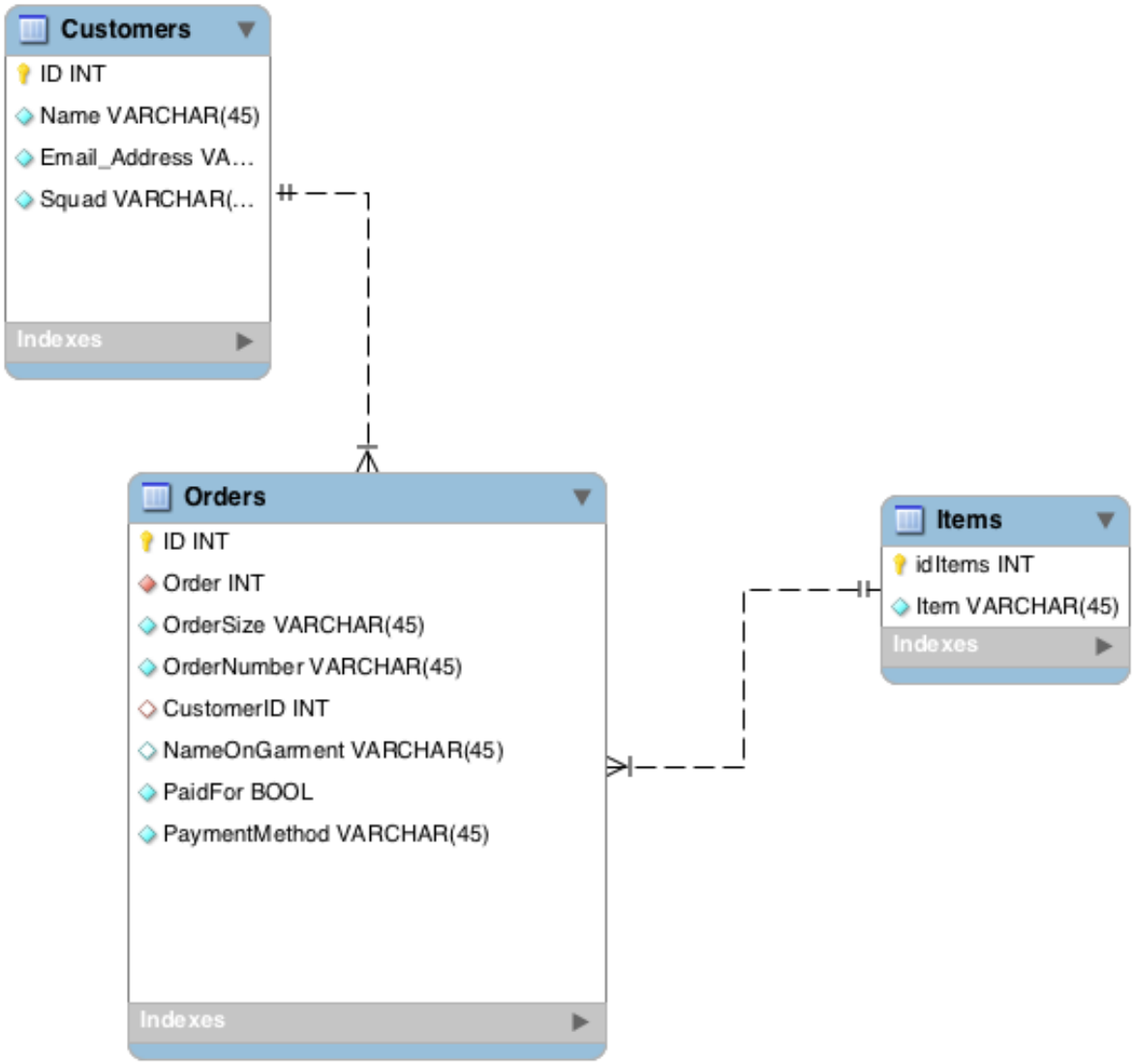


Figure 10: The EER for the designed system

Figure 10 is the database diagram for the system as it is designed, this essentially takes the place of the order form in the original EER diagram seen in Figure 6, except it requires far less paper to be wasted during the order process.

2.5 Identification of Storage Material and Format

The Java portion of this program will be stored on the user's computer, for them to connect to the database, which will be stored on the server that the swimming club's website is hosted

on, in order to make the web form passing far easier, as this will obviously be hosted on the club's website as well, so just requires a localhost as opposed to a static IP or DNS lookup, which, while feasible, would be far harder to implement than having the website and server in the same place. This will be needed for the Java portion of the program, however this is going to be used far less often than the order form will, so it saves more time having the MySQL database on the website server.

Most of the data will be written into and read from the database using SQL queries, as this is the only real way to input data into a table. When the database initialises, it will read in a SQL script and execute that using Mybatis, so a FileInputStream within Java; when the data from the database is output to an Excel spreadsheet then it will use a FileOutputStream within Java; and the connection settings, both in the Java end and the web end will use an XML parser from a FileInputStream on the Java end and a simplexml_load_file in PHP.

The finished project will end up less than 100 MB, and most likely closer to 60, which would be including the Excel file as an output and all of the required libraries for the compiled .jar file. The raw source code will probably be less than 1MB.

The backup system for the program will involve backing up the project when the website is backed up. The program itself will have a copy on the user's computer, then also on my computer and so in any backups I make using Backblaze, as well as a copy on Dropbox, and then the source code will be available on GitHub, so if everything goes wrong it will be feasible to retrieve the code and recompile it. If all three of GitHub, Dropbox and Backblaze go down then there's probably problems that are slightly more important than obtaining a copy of source code, but even then it will be stored on a physical USB stick kept with my computer.

2.6 Identification of Processes and Algorithms for Data Transformation

In this there will be a database search function, however this project will be using the ORDER BY function within MySQL as opposed to an algorithm due to how the data is handled when it is retrieved from the database, as it is retrieved as a ResultSet, and then converted into a map before being inserted into the TableView. So a bubble sort or shuttle sort would be unable to run on the data. The relevant section of code can be seen on lines 96-109 of Appendix A.3, or lines 105-118 of Appendix A.5. This is feasible as the volumes of data will be too small to justify a Bubble or Shuttle sort on the data and the MySQL ORDER BY will be fast enough for what is needed for this project. The SQL statements that will be used in this project are given here:

```
select o.ID, o.CustomerID, c.Name, c.Email_Address, c.Squad,
    o.Orders, o.OrderSize, o.OrderNumber, o.NameOnGarment, o
    .PaidFor, o.PaymentMethod, i.Item from Orders o INNER
    JOIN Customers c ON o.CustomerID = c.ID INNER JOIN Items
    i ON i.idItems=o.Order where o.ID like ? or o.CustomerID
    like ? or c.Name like ? or c.Email_Address like ? or c.
    Squad like ? or o.Orders like ? or o.OrderSize like ? or
    o.OrderNumber like ? or  o.NameOnGarment like ? or o.
    PaymentMethod like ? or i.Item  like ? ORDER BY;
```

This first SQL query is the search query. This selects everything in the database where one of the columns matches the search query. The question marks are from the prepared statement in Java, and there would normally be a variable after the ORDER BY however this was cut off for readability in this report. This is the longest and most complex single SQL query in the whole project.

```
select o.ID, o.CustomerID, c.Name, c.Email_Address, c.Squad,
    o.Orders, o.OrderSize, o.OrderNumber, o.NameOnGarment, o
    .PaidFor, o.PaymentMethod, i.Item from Orders o INNER
    JOIN Customers c ON o.CustomerID = c.ID INNER JOIN Items
    i ON i.idItems=o.Orders ORDER BY;
```

This SQL query is very similar to the previous one, the difference being that it doesn't have to be a prepared statement as the only user input is from a dropdown box, so the statement doesn't have to be protected in any way, due to the lack of attack vectors. Once again there would be a variable after the ORDER BY but this was cut out to improve readability.

```
SELECT * FROM Customers INTO OUTFILE /Users/tsmoffat/
    kitordersystem/kitordersystem/kitordersystem/Customers.
    csv FIELDS ENCLOSED BY '"' TERMINATED BY ';' ESCAPED BY '
    "' LINES TERMINATED BY '\r\n'; SELECT * FROM Orders INTO
    OUTFILE /Users/tsmoffat/kitordersystem/kitordersystem/
    kitordersystem/Orders.csv FIELDS ENCLOSED BY '"'
    TERMINATED BY ';' ESCAPED BY '"' LINES TERMINATED BY '\r\
    n';
```

This SQL script is run to dump the contents of the Orders and Customers tables to CSV files so that they could be loaded back in to the database if need be. This script is run just before the database gets recreated by Mybatis in DBReset.jave, which can be found in Appendix A.7.

The last SQL query is actually a SQL script, and can be seen in full in Appendix A.8. This is the SQL script that is called when the entire database is rebuilt from the ground up. This is actually the most complex bit of SQL code, although it can't be called a query, as it is in fact a series of them, although Java does execute them all in the same update.

2.7 User Interface Design and Rationale

The user interface will be constructed in JavaFX, as this is both easier to work with and better looking than swing. It also allows me to do some preliminary design work on the GUI in the program Scenebuilder, although, thanks to some more preliminary testing, this won't end up being the way I make this GUI. Figure 11 shows the design for the Main Menu of the Java end of the program. This is a very simple UI and so it is easy to understand, even if the user is unfamiliar with computers. This is the only screen with any real input. There is a text box once one clicks on the Remake button, but that is very linear, in that there are only two choice: input the phrase and click ok or close the box. This can be seen in figure 12. The final screen will

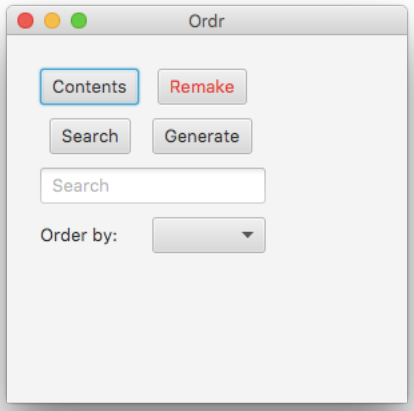


Figure 11: The design for the Main Menu

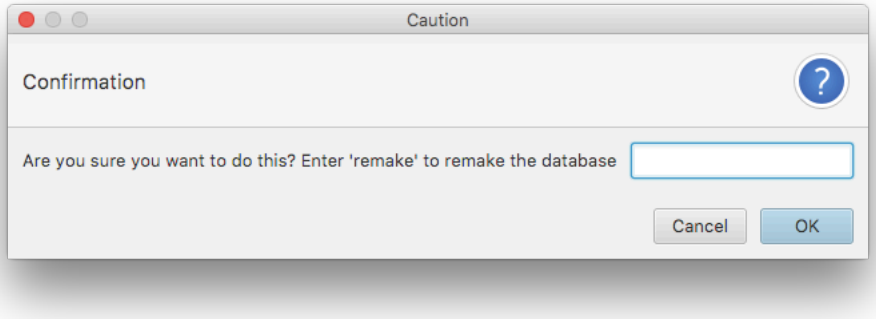
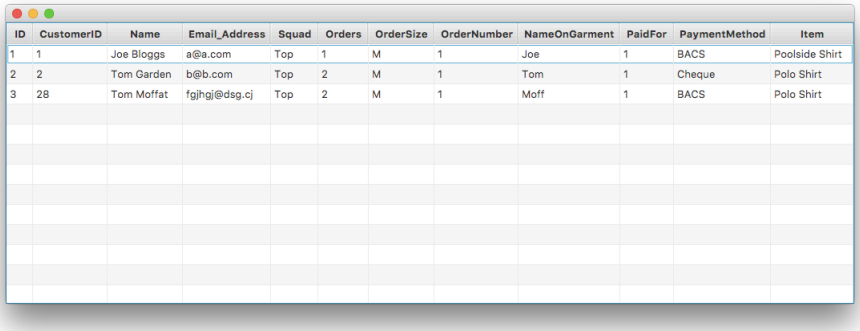


Figure 12: The design for the confirm text box. This will just use a JavaFX dialog box



ID	CustomerID	Name	Email_Address	Squad	Orders	OrderSize	OrderNumber	NameOnGarment	PaidFor	PaymentMethod	Item
1	1	Joe Bloggs	a@a.com	Top	1	M	1	Joe	1	BACS	Poolside Shirt
2	2	Tom Garden	b@b.com	Top	2	M	1	Tom	1	Cheque	Polo Shirt
3	28	Tom Moffat	fgjhgj@dsg.cj	Top	2	M	1	Moff	1	BACS	Polo Shirt

Figure 13: The display of the contents and the search display. These will be generated dynamically

show the contents and the results of the search query. As shown in Section 2.6, this will vary depending on the ordering and the search query in the case of the search display.

This UI is designed solely for people who have little experience using SQL. If they have prior experience using this then they either use this GUI if they want or they can bypass the GUI entirely and connect to the database using a terminal/command prompt and enter the commands directly, although this will mean that they won't be able to use some of the features of the program, such as the dump the relevant contents of the database to an Excel spreadsheet to be sent off to the manufacturers. Most of the functions to add a user or an order will be done by the users themselves on the webpage. The designs for these are shown by figure 14. These are quite simple to follow, with large buttons and text boxes to make it obvious what goes where. This facilitates ease of use for the customers,as some of them may not be particularly used to using computers.

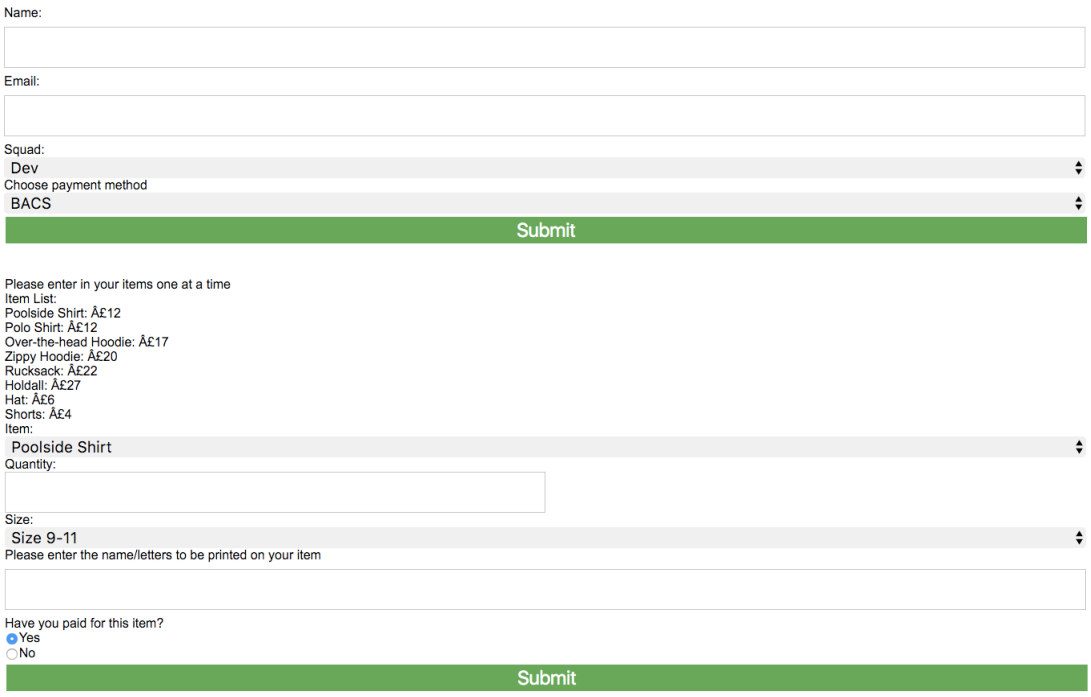


Figure 14: The input forms on the internet

2.8 Planned Data Capture and Entry

The planned data entry forms can be seen in Figure 14. The design was chosen to be easy to read, in nice large fonts with a good contrast. This is useful as a variety of ages will be using this web page and the aim is for it to be easy for everyone to read and to use. This is achieved through the minimal use of attention grabbing features, really the only one is the large, green submit button. It is also very easy to tell what goes where, either due to the large font on the drop-down menu or the label just above it which is easy to read. The text boxes are large, so that they can be selected easily, and to show that the length of string available is very long. In this case, the string that the customer can input is limited to 45 characters although as discussed earlier the customer is unlikely to reach this limit as this length would accept a name as long as the third longest place-name in the world, which, suffice to say, is unlikely during normal usage. If a user runs into problems with the length of their name then this can always be increased.

2.9 Planned Valid Output Designs

The major output of this program is an Excel spreadsheet with all the relevant values from the table in it. Each item will be split into its own sheet, with the name of the customer, the size they want and what they want to be printed on the item input in columns on the respective sheet. This also includes hats and shorts as, even though the options for printing names on the hats and shorts isn't available at this moment in time, this allows for easy expansion in the near

future. This output is slightly changed from the Analysis as, from some preliminary testing, the Java libraries that allow access to Word and Excel documents work half the time and then the other half, even with unchanged code, they don't. This level of reliability was unacceptable so the method chosen after this preliminary testing was to write a Python script that handles the output to the spreadsheet and then using Java to call and execute the Python script. This has proven to have a 100% success rate and so will be the method that will be used in the final code.

This program also outputs two CSV files when the remake function is called, which dumps the contents of the database so that it can be repopulated so no orders are lost if the database needs to be recreated for some reason. This is just a backup backup function, just in case the backups taken of the database can't be reloaded.

2.10 Measures Planned for Security and Integrity of Data

Most of the validation techniques are performed by MySQL, checking whether the values are null or whether they're over the character limit, etc. The email and the number of items are defined in the HTML source, the email as an email datatype and the number of items has a range defined between 1 and 5 so if the input string isn't an email address or the number of items is over 5 then the web browser will kick out an error and make them change it. SQL Injections are prevented by the use of Prepared Statements, which reduces this attack vector considerably. This means that there is no way that people should be able to gain a threshold into the database using this method.

Error messages will be output in a dialog box with the error message and the stack trace for if the user is slightly more experienced with computers, or for troubleshooting purposes. These dialog boxes are the ones built in to JavaFX, with an expanded text box built in for displaying the stack trace. An example of this is available in Appendix C.2.2, Figure 56.

If data input is incorrect then it goes into the database and requires a human to check it and alter it when it is output into an Excel spreadsheet, or the customer can submit a request to the Kit Coordinator to alter the record directly in the database, if the Kit Coordinator has the knowledge to do so. If not then the data will have to be modified manually in the Excel spreadsheet. The customer could also submit another form with the updated details and this will be taken as the more recent one and so used when the order sheet is being used, as long as there are no orders tied to that customer. If the order is messed up then they can submit another order and ask for the incorrect order to be taken off the order sheet.

The backup solution has already been detailed, however it will be covered in more detail here. For this program, it will follow the 3-2-1 backup system, i.e. At least three backups, on at least two different storage mediums with at least one stored off site. In this case, the program source code is stored on my computer, on a flash drive and on GitHub. When the program goes into production, a backup will be taken of the database when the website itself is backed up. This will be stored with the web host, and a task will be set-up where the most recent back-up is downloaded whenever the database is backed up, then synchronised onto a NAS using an incremental backup job. This is to ensure that even if one or more of these systems goes down there is always a copy of the source code available.

2.11 Measures Planned for System Security

The database will be password protected, and this will be administered through MySQLWorkbench, as this allows for multiple accounts to be created with different permissions. So when the program goes into production a user account just for the web end will be created that allows one to insert records and read them but do nothing else. This will help prevent vandalising the database, as the vandals will have no useful access. In production this connection will be protected by SSL, with a certificate signed by Let'sEncrypt, as it is free and so within the budget of a small swimming club.

As hard as it is to say it, from a bad security standpoint, the best protection on the front end will probably be through obscurity and through not being a very interesting target. The swimming club isn't particularly big or important, and so isn't particularly hard to gain access to and so won't gain anyone any bragging rights for being able to crack (criminally hack) into it. Most of those types of people, except for the Script Kiddies who are probably the biggest threat, go for the big targets that are locked down, for the fun of it and for the challenge or because they want to deliver a message. This club is about as inoffensive as it is possible to get, and isn't hard at all to gain access to so will just end up blending in among the thousands of other websites that are very similar. It would be folly, and a rather large waste of time to assume anything different so for this the built in access controls in MySQL will be more than sufficient.

2.12 Overall Test Strategy

First, the program will undergo black box testing, to check that every item entered in is output in the manner it should be in the database. This is more of a problem on the HTML end of the program, as there is very little data entry on the Java side of the program. This will be tested by entering in semi-fictional data and then checking to see if the output is the same as the input. In this, normal data will be tested, followed by an extremely long name (for the purposes of testing I have chosen Llanfairpwllgwyngyllgogerychwyrndroblllllantysiliogogogoch, as I feel this is a sufficiently long name to test responses), and then a SQL query, to test whether the prepared statements are working as intended.

Following this I will conduct white box testing which will mostly consist of editing the connection XML files and then retrieving the response to the change. This will be achieved through dialog boxes from the web browser and the stack trace outputs from the Java code.

Integration testing will allow me to test if some of the functions in the PHP code work, which will be achieved by testing the whole order process as if I were a customer then reading the output of the order in the database. This is because some of the code in the second PHP script called will require session variables from the first PHP script and so it will test to see if the session variables are transferred correctly.

Finally, the system will undergo System Testing, where the client will be brought in and asked to test the code, to see if the system is in fact simple enough to use and if the finished system meets their requirements and needs. This will require some time before the system is put into production in order to make any suggested changes and to make sure that the system will work

on any system, as opposed to just mine, as there are hard-coded file links in the program that will have to be removed and replaced with relative links.

3 TESTING

Please note: All screenshots are available in Appendix C, references will be made in the tables

3.1 Testing Web Forms

3.1.1 Testing Customer Details Form

Screenshots for this table are available in Appendix C.1.1

Table 12: Customer Form Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
1	Testing Name	To test response when normal data submitted	Thomas Moffat	Thomas Moffat will be appended to the end of the database	PASS	Figure 15
2	Testing Name	To test what will happen when a SQL query is submitted to the database through the form	SELECT * FROM Customers;	Query should be added in as a name	PASS	Figure 16
3	Testing Name	To test response when long names are inserted	The full name of Llanfair PG	Database should reject the name as it is too long	PASS	Figure 17
4	Testing Name	To test response when field is blank		Database should accept input	PASS	Figure 18
5	Testing Email	To test response when email is passed	a@a.com	Database will accept the input	PASS	Figure 19
6	Testing Email	To test what happens when a string that is not an email address input	a	Web browser will reject the input due to input type being set as email	PASS	Figure 20
7	Testing Email	To test response when TLD is non-existent	a@a.shad	Email will be accepted	PASS	Figure 21

Table 12: Customer Form Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
8	Testing Squad	Testing to see if the Squad is correctly inserted into the database when chosen from a drop-down menu	Top	Top will be inserted into the Squad column of the table (the furthest right column)	PASS	Figure 22
9	Testing Squad	Testing to see if the Squad is correctly inserted into the database when chosen from a drop-down menu	Dev	Dev will be inserted into the Squad column of the table (the furthest right column)	PASS	Figure 23
10	Testing Squad	Testing to see if the Squad is correctly inserted into the database when chosen from a drop-down menu	Jag	Jag will be inserted into the Squad column of the table (the furthest right column)	PASS	Figure 24
11	Testing Payment	Testing to see if payment method is correctly inserted into the table when chosen from a drop-down menu	Cheque	Cheque will be inserted into the table in the furthest right column	PASS	Figure 25
12	Testing Payment	Testing to see if payment method is correctly inserted into the table when chosen from a drop-down menu	BACS	BACS will be inserted into the table in the furthest right column	PASS	Figure 26

Table 12: Customer Form Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
13	Testing Payment	Testing to see if payment method is correctly inserted into the table when chosen from a drop-down menu	Cash	Cash will be inserted into the table in the furthest right column	PASS	Figure 27

3.1.2 Testing Order Details Form

Screenshots for this table are available in Appendix C.1.2. A reference for the layout of the items table for the items testing is as follows:

Table 13: Representation of Items table

idItems	Items
1	Poolside Shirt
2	Polo Shirt
3	Hoodie
4	Zippy Hoodie
5	Hat
6	Shorts
7	Backpack
8	Holdall

Table 14: Order Form Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
1	Testing Customer ID	Testing to see if the Customer ID is correctly retrieved from the database when the previous form is submitted	NA	(As of testing) Customer ID 35 should be inserted into the table	PASS	Figure 28

Table 14: Order Form Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
2	Testing Item	Testing to see if an item is correctly inserted into the database when chosen from a drop-down menu (for the Testing Item tests, the column of interest is column 2)	Shorts	The item Shorts would be inserted into the table	PASS	Figure 29
3	Testing Item	Testing to see if an item is correctly inserted into the database when chosen from a drop-down menu	Poolside Shirt	The item Poolside Shirt would be inserted into the database	PASS	Figure 30
4	Testing Item	Testing to see if an item is correctly inserted into the database when chosen from a drop-down menu	Polo Shirt	The item Polo Shirt would be inserted into the database	PASS	Figure 31
5	Testing Item	Testing to see if an item is correctly inserted into the database when chosen from a drop-down menu	Hoodie	The item Hoodie would be inserted into the database	PASS	Figure 32
6	Testing Item	Testing to see if an item is correctly inserted into the database when chosen from a drop-down menu	Zippy Hoodie	The item Zippy Hoodie would be inserted into the database	PASS	Figure 33

Table 14: Order Form Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
7	Testing Item	Testing to see if an item is correctly inserted into the database when chosen from a drop-down menu	Hat	The item Hat would be inserted into the database	PASS	Figure 34
8	Testing Item	Testing to see if an item is correctly inserted into the database when chosen from a drop-down menu	Backpack	The item Backpack would be inserted into the database	PASS	Figure 35
9	Testing Item	Testing to see if an item is correctly inserted into the database when chosen from a drop-down menu	Holdall	The item Holdall would be inserted into the database	PASS	Figure 36
10	Testing Quantity	Testing to see if a quantity of an item is correctly inserted into the database	2	The quantity 2 should be added into the OrderNumber column (the fourth column along) of the table	PASS	Figure 37
11	Testing Quantity	Testing to see what the response is when the quantity is over the number of items allowed	7	The web browser should reject the quantity without the contents being added into the table	PASS	Figure 38

Table 14: Order Form Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
12	Testing Size	Testing to see what the response is when the size is input from the dropdown menu	Medium	M should be in the OrderSize column (the third column along) in the most recent entry to the table	PASS	Figure 39
13	Testing Size	Testing to see what the response is when the size is input from the dropdown menu	9-11	9-11 should be in the OrderSize column in the most recent entry to the table	PASS	Figure 40
14	Testing Size	Testing to see what the response is when the size is input from the dropdown menu	12-13	12-13 should be in the OrderSize column in the most recent entry to the table	PASS	Figure 41
15	Testing Size	Testing to see what the response is when the size is input from the dropdown menu	Small	S should be in the OrderSize column in the most recent entry to the table	PASS	Figure 42
16	Testing Size	Testing to see what the response is when the size is input from the dropdown menu	Large	L should be in the OrderSize column in the most recent entry to the table	PASS	Figure 43
17	Testing Size	Testing to see what the response is when the size is input from the dropdown menu	Miscellaneous (for bags, hats and shorts)	Misc should be in the OrderSize column in the most recent entry to the table	PASS	Figure 44

Table 14: Order Form Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
18	Testing Name	Testing to see what the response is when a normal name is input	Moff	The input String should be displayed in the bottom row under the NameOnGarment column	PASS	Figure 45
19	Testing Name	Testing to see what the response is when a SQL query is input into the database	SELECT * FROM Customers;	The input String should be inserted into the table as-is without the command being executed	PASS	Figure 46
20	Testing Name	Testing to see what the response is when the name is too long for the SQL Table	The long form of Llanfair PG	The table should not accept it	PASS	Figure 47
21	Testing Paid	Testing to see what the response is when the paid radio is set to yes	Yes	1 should appear in the PaidFor column of the table (the seventh column)	PASS	Figure 48
22	Testing Paid	Testing to see what the response is when the paid radio is set to no	No	0 should appear in the PaidFor column of the table (the seventh column)	PASS	Figure 49

3.1.3 Miscellaneous Testing

This section is for testing the things that don't fit into the other two tests, so what happens if the connection details are missing, or different to what was expected, etc. Screenshots can be found in Appendix C.1.3. For the following tests, there will only be one screenshot as the response from both web pages will be exactly the same.

Table 15: Miscellaneous Web Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
1	Connection	Testing the result when kitorder2.xml is missing	The XML file will be removed from the directory	PHP script will not be able to connect and return an error	PASS	Figure 50
2	Connection	Testing the result when the DBMS is altered	PostgreSQL	An error will be returned	FAIL (On further inspection it turns out the PHP script doesn't use the DBMS so this test is of no import)	
3	Connection	Testing the result when the database name is altered in the XML	my	An error will be returned	PASS	Figure 51
4	Connection	Testing the result when the user name is altered in the XML	roo	An error will be returned	PASS	Figure 52
5	Connection	Testing the result when the password in the XML file is altered	***	Access will be denied / An error will be returned	PASS	Figure 53

Table 15: Miscellaneous Web Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
6	Connection	Testing the result when the port in the XML file is altered	330	The script will return an error as it can't find the MySQL server	FAIL (The query completed successfully and the details were inserted into the table as they should have been)	
7	Connection	Testing the result when the server address in the XML file is altered	localhos	The script will return an error as it can't find the server	PASS	Figure 54
8	Styling	Testing the result when the CSS is missing from the website directory	styling.css will be moved from the working directory	The website will revert to default styling	PASS	Figure 55

3.2 Java Testing

3.2.1 Java General Testing

This section is to test the program holistically, to see what the result is when a button is clicked, etc. Screenshots will be available in Appendix C.2.1. Please note that since this testing was concluded an edit was made to the code for displaying the stack trace so the stack trace now outputs into a dialog box. This gives exactly the same results as the code did originally, however for brevity updated screenshots are not provided. An example of this updated error message is given in Appendix C.2.1, Figure 56

Table 16: Java Program Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
1	All Contents	Testing the result when the order is set to Name A-Z	Order by is set to Name A-Z	The output will show the names in alphabetical order	PASS	Figure 57

Table 16: Java Program Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
2	All Contents	Testing the result when the order is set to Name Z-A	Order by is set to Name Z-A	The output will show the names in reverse alphabetical order	PASS	Figure 58
3	All Contents	Testing the result when the order is set to Item	Order by is set to Item	The output will show the names in order of the item they bought	PASS	Figure 59
4	All Contents	Testing the result when the order is set to OrderID	Order by is set to OrderID	The output will show the names in order of the Order ID	PASS	Figure 60
5	All Contents	Testing the result when the order is set to Squad	Order by is set to Squad	The output will show the names in alphabetical order by Squad	PASS	Figure 61
6	All Contents	Testing the result when the order is set to Customer ID	Order by is set to CustomerID	The output will show the names in order by Customer ID	PASS	Figure 62
7	All Contents	Testing the result when the order is set to Payment Method	Order by is set to Payment Method	The output will show the names in alphabetical order by Payment Method	PASS	Figure 62
8	All Contents	Testing the result when the order is set to null	Order by is set to null	The program will kick a stack trace into the terminal (note that this can only occur when the program has just been started up, after this the option for null is unselectable)	PASS	Figure 62

Table 16: Java Program Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
9	Generate	Testing what happens when the generate button is pressed	Generate button is clicked	The program will call a function that generates the contents of the database as a spreadsheet	PASS	Firgure 65
10	Database Search	Testing what happens if both the order and the search string are null	Both will be left blank	The program will output a stack trace and nothing else will happen	PASS	Figure 66
11	Database Search	Testing what happens when searching for something with ordering left blank	Order will be left blank, search string will be Tom	The program will throw an error in the console and nothing else will happen	PASS	Figure 67
12	Database Search	Testing what happens when the search string is left blank and the order is set to Name A-Z[1]	Search left blank	The output will be the same as clicking on contents	PASS	Figure 68
13	Database Search	Testing what happens when a string that is present in the database is searched for	Tom	Will return every instance of the string 'Tom' from the database	PASS	Figure 69
14	Database Search	Testing what happens when a string that is not present in the database is searched for	Jim	Will return an empty window	PASS	Figure 70

1 NB: As the code for the Contents and the search are very similar with how they deal with ordering and displaying said ordering, this will not be shown to be retested in this report. Suffice to say that they have been tested and all work

Table 16: Java Program Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
15	Testing Remake	Testing to see what happens when the close button on the remake dialog box is clicked	Close button will be clicked	The window should exit without error	PASS	Figure 71
16	Testing Remake	Testing to see what will happen when the cancel button on the remake dialog box is clicked	Cancel button will be clicked	The window should exit without error	PASS	Figure 72
17	Testing Remake	Testing to see what will happen when the OK button on the dialog box is clicked and the string 'remake' isn't present in the text box	The text box will be left blank	There should be an output, but nothing will be done to the database	PASS	Figure 73
18	Testing Remake	Testing to see what will happen when the OK button on the dialog box is clicked with a string other than 'remake' present	The string Tom will be entered	The string will be output to the console (a testing message) and nothing will happen to the database	PASS	Figure 74
19	Testing Remake	Testing to see if the remake function works when the remake button is clicked and the string 'remake' is present	The string remake will be present in the text box	The entire database should be dropped then the Items table should be remade correctly from a CSV file	PASS	Figure 75

Table 16: Java Program Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
20	Generate Testing	Testing to see what the output is when the database has no contents with the exception of the items table	The database is empty and the Generate button will be clicked	A spreadsheet with only headings will be generated	PASS	Figure 76
21	Close Testing	Testing to see what the output will be when the program is closed	The close button will be clicked	The program will exit with an exit code 0	PASS	Figure 77

3.2.2 Java Connection Testing

This section is to see what happens when the connection settings are altered in kitorder.xml. Please note that, as the responses will be very similar for each function of the program, screenshots will only be provided for the response of the Contents function, although the other responses have been tested and have been shown to have similar responses.

Table 17: Java Program Connection Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
1	Connection Testing	To test what the output is when the XML file with the connection settings is missing	kitorder.xml missing	Program will ouput a stacktrace to the console	PASS	Figure 78
2	Connection Testing	To test what the output is when the DBMS node in the XML file is altered	dbms node changed to mysq	Program will kick out a stacktrace	PASS	Figure 79

Table 17: Java Program Connection Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
3	Connection Testing	To test what the output is when the database name is changed within kitorder.xml	dbname changed to myd	The program will connect to the database as normal due to redundant SQL statements in the code to use mydb	PASS	Figure 80
4	Connection Testing	To test what the output is when an invalid user name is inserted into the kitorder.xml file	root changed to roo	Access will be denied by the MySQL server and the program will output a stack trace to this end	PASS	Figure 81
5	Connection Testing	To test what the output is when an invalid password is inserted into the kitorder.xml file	**** changed to ***	Access will be denied by the MySQL server and the program will output a stack trace about the issue	PASS	Figure 82
6	Connection Testing	To test what the output will be when the Port Number within the kitorder.xml is changed	3306 changed to 330	Program should error with a StackTrace	FAIL (On further inspection, the code was set to always use port 3306, upon chang- ing this it became PASS)	Figure 83
7	Connection Testing	To test what the output will be when the Server Address within the kitorder.xml is changed	localhost changed to localhos	Program should error with a stack trace	PASS	Figure 84

Table 17: Java Program Connection Testing

Test	Purpose	Description	Test Data	Expected Result	Pass/Fail	Evidence Ref.
8	Connection Testing	To test what the output is when the MySQL server is not present (so missing from the system or turned off)	Server turned off	The program will error with a connection failed and a stacktrace	PASS	

4 APPRAISAL

4.1 Appraisal

1. It must have a well-structured(1NF or 2NF)database system that can be easily accessed. This has been achieved through the GUI system on the private end of the program, which is very easy to use.

2. It must have some sort of user interface allowing a person to enter the data then have the program handle the sorting. This has been achieved, the user interface allowing the person to enter data is web-based and then the sorting is handled by the SQL database and the Java program.

3. It must store the data in a usable way (Most likely plain text). This is achieved, as the database itself is useful, then the output form is an Excel spreadsheet, which can have data copied to and from it, and then when the database is being remade it dumps the contents of the database to two CSV files, which can then be used to repopulate the database or just for future reference.

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. This has been achieved through the use of SQL ORDER BY and prepared statements with wildcard operators.

5. It must be lightweight enough to be run on a web server, such as the one hosting the club website. This has been achieved, as the whole project, including libraries, is under 100 MB. This is mostly Java, the web technologies are under 10MB.

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. This has been achieved, as the program is able to output the relevant information to an Excel spreadsheet which is formatted well enough to be used as an order form.

7. It should have a web-based front end so the orders for kit can be placed via the club website. This has been achieved, as it is the main way of entering data into the database. It is easy enough to understand thanks to the large scaling.

8. It could be able to send a mass email to all of the people who have ordered kit. This objective was unfortunately not achieved due to a lack of time after the other objectives were met and a lack of satisfactory java libraries for interacting with email. Although now that this program is using python as well then this might be something to revisit further down the line.

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. This was not achieved for the same reasons as the previous item, as

well as the fact that it isn't particularly secure. It requires the program to be open all the time, which eats up processing power, and then someone who knows what to write could send the program an email with instructions to pass on a malicious payload to the entire mailing list of people. So for the time being it is far safer to have a manual email system, although this may be something to add in the future.

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. This was the furthest from being implemented, which was unfortunate, however it would have required a far greater level of system security than is currently present, and would have required forcing the user to use their debit card or PayPal or Amazon or an equivalent payment processor, which not everyone wants to do, or in some cases is even able to do. This would be on the list for updates in the far future when my skills in web development and specifically staying secure on the web are improved massively.

4.2 User Feedback

- How well would you say the system has met:
    1. The requirement to have a simple user interface? This ordering system has a simple and intuitive user interface that achieves the objective requested of it. It would make it easy for any user to place an order for kit on the club website.
    2. The requirement to store the data in a usable way? The order data is stored in a tabular form and in its raw form might be confusing, but the user manual includes a key to help with this. Unless there is a problem with an order the data will usually only be used after having been processed output to the final spreadsheet, so the readability of the data in the database is not critical.
    3. The search function requirements? The search function works well and would be useful for verifying an order that had been placed against the deliveries received.
    4. The automation requirements? The automation requirement was to output an order form that could be emailed to the supplier and this is achieved using the Generate function which produces an Excel spreadsheet. This separates all the different order items by type which is exactly what is required. This makes it easy for the supplier to place their order and for the swimming club to match orders with each customer.
- How easy would you say it is to use? The web based database input is very easy intuitive to use the database access interface is simple to use and achieves its purpose.
- Do you have any improvements to suggest? One improvement would be to include the date that each order was placed in the database. The requirement to have an automatic email sent with an order to the supplier might also be useful but would be tricky to achieve.
- Do you have any other comments on the solution? I am impressed with the solution that Tom has achieved which he has put a huge amount of time and effort into. It will

prove very useful in improving the club kit ordering process and will prevent the errors that occurred using the previous method of a mixture of paper email forms and manually entering all the data into a spreadsheet.

4.3 Analysis of User Feedback

- The database headings can be confusing, however there is an easy to use table in the user manual that fixes this issue, as it explains what each of the table headings actually means. I could use this to change the table headings to be more understandable by the every day user of this program.
- Other than this minor issue, the user had no problems with this program, which is a good thing, it means they're an easy user to deal with. However, I will take this opportunity to raise some issues of my own. For some reason unbeknownst to me, my web browser insists on placing Â in front of my £ signs and there are no errors that I can find on the internet. So this would be the first thing I would fix, followed by the GUI for the main program, as I had a nice one designed that would have kept everything kept together, however when it came to actually making it work it wouldn't, so I had to revert to the(admittedly useful but) not particularly interesting GUI that made it into production.

4.4 Improvements and Extensions

The main improvement that was suggested by the user was to add a column into the database for the date. This would be relatively easy to accomplish. It would require a date column in the Orders table of the database, and then it would require the use of a date function in PHP to get the current date and input it into the correct column in the database.

The other improvement suggested by the user would be to automatically send an email to the kit suppliers when the order form is generated, with it attached. This shouldn't be too hard to accomplish, it would require some research on Python libraries to see whether any of them allowed this sort of thing to happen. This would be a Python job more than a Java job as if the libraries for Excel are any indication, Python has far better support for this sort of thing than Java does.

Another improvement that would also be nice to have would be a better GUI, the lack of this can be chalked up to a lack of familiarity with JavaFX and the need to finish coding before a deadline. This would be a very early update to the program and would improve it massively, keeping everything in one window. This would increase user comprehension, as it would be easier to find everything.

Yet another update that would be useful would be the ability to send out a mass email to everyone on the order form for that month telling them when their kit is available. This would again be achieved through the use of Python scripts which would scrape the order form for unique email addresses and then input them into a blank email, which would have a form email inserted and then the email would be sent to the recipients. This should be a reasonably easy feature to implement.

The final feature that would be implemented at some point in the future would be the web payment system. This would require the use of SSL, and the whole website would have to encrypted in this way. Then a PayPal applet would be embedded into the page, which would mean that the club wouldn't have to do the debit card processing, PayPal would do the heavy lifting and the club would have the convenience of being able to pay for the order there and then. This would also require the implementation of a proper shopping basket, as opposed to the current implementation. This would be perfectly possible, however it would require a complete overhaul of the web code. This would be very doable, and may become reality in the future.

Appendices

A SOURCE CODE APPENDIX

A.1 MainMenu.java

```
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.layout.Priority;
13 import javafx.scene.paint.Color;
14 import javafx.stage.Stage;
15 import org.xml.sax.SAXException;
16
17 import javax.xml.parsers.ParserConfigurationException;
18 import java.io.IOException;
19 import java.io.PrintWriter;
20 import java.io.StringWriter;
21 import java.sql.SQLException;
22 import java.util.Optional;
23
24 /**
25  * This is, as the name suggests, the main menu for my programme. Where all
26  * the magic happens.
27  */
28 public class MainMenu extends Application {
29
30     public static String token;
31     public static String ordering;
32
33
34     public static void main(String[] args) throws SQLException, IOException {
35         launch(args);
36     }
37
38     /**
39      *Starts the stage, fairly self-evident
```

```
40          */
41         @Override
42         public void start(Stage primaryStage) {
43             primaryStage.setTitle("Ordr");
44             Button remakeButton = new Button("Remake");
45             Button dbViewButton = new Button("Contents");
46             Button emailButton = new Button("Generate");
47             Button searchButton = new Button("Search");
48             TextField searchField = new TextField();
49             ComboBox orderComboBox = new ComboBox();
50             Label label = new Label();
51             searchField.setPromptText("Search");
52             orderComboBox.getItems().addAll("Name A-Z", "Name Z-A", "Item",
53                     "Order ID", "Squad", "Customer ID", "Payment Method");
54             remakeButton.setOnAction(new EventHandler<ActionEvent>() {
55                 /**
56                  *
57                  * @param event
58                  * This controls what happens when the reset button is clicked,
59                  * calls the DBReset class followed by the CSVReader class, as
60                  * long as the string input into the dialog box is "remake"
61                  */
62                 @Override
63                 public void handle(ActionEvent event) {
64                     TextInputDialog dialog = new TextInputDialog();
65                     dialog.setTitle("Caution");
66                     dialog.setContentText("Are you sure you want to do this? " +
67                             "Enter 'remake' to remake the database");
68                     Optional<String> result = dialog.showAndWait();
69                     System.out.println(result);
70                     String result1 = result.toString();
71                     result1 = result1.replace("Optional[", "");
72                     result1 = result1.replace("]", "");
73                     System.out.println(result1);
74                     if (result.isPresent() && result1.equals("remake")) {
75                         System.out.println("Resetting");
76                         DBReset reset = new DBReset();
77
78                         try {
79                             CSVReader reader = new CSVReader();
```

```
80                      } catch (IOException | SQLException | ParserConfigurationException |
    ↪  SAXException e) {
81
82                          e.printStackTrace();
83                          Alert alert = new Alert(Alert.AlertType.ERROR);
84                          alert.setTitle("Exception Dialog");
85                          alert.setHeaderText("Exception");
86                          alert.setContentText(e.getMessage());
87
88
89  // Create expandable Exception.
90                          StringWriter sw = new StringWriter();
91                          PrintWriter pw = new PrintWriter(sw);
92                          e.printStackTrace(pw);
93                          String exceptionText = sw.toString();
94
95                          Label label = new Label("The exception stacktrace was:");
96
97                          TextArea textArea = new TextArea(exceptionText);
98                          textArea.setEditable(false);
99                          textArea.setWrapText(true);
100
101                         textArea.setMaxWidth(Double.MAX_VALUE);
102                         textArea.setMaxHeight(Double.MAX_VALUE);
103                         GridPane.setVgrow(textArea, Priority.ALWAYS);
104                         GridPane.setHgrow(textArea, Priority.ALWAYS);
105
106                         GridPane expContent = new GridPane();
107                         expContent.setMaxWidth(Double.MAX_VALUE);
108                         expContent.add(label, 0, 0);
109                         expContent.add(textArea, 0, 1);
110
111 // Set expandable Exception into the dialog pane.
112                         alert.getDialogPane().setExpandableContent(expContent);
113
114                         alert.showAndWait();
115                     }
116
117                 } else {
118
119                 }
```

Thomas Moffat, 51337, 4042

```
120
121
122             }
123         });
124         dbViewButton.setOnAction(new EventHandler<ActionEvent>() {
125             /**
126              *
127              * @param event
128              * This controls what happens when the button to view everything
129              * in the database at that time is called, takes the value from the
130              * combo box for the order and passes it through to the class,
131              * which then runs a SQL statement with an order by
132              */
133             @Override
134             public void handle(ActionEvent event) {
135                 ordering = orderComboBox.getValue().toString();
136                 Platform.runLater(new Runnable() {
137                     public void run() {
138                         try {
139                             new TableViewTest().start(new Stage());
140                         } catch (Exception e) {
141
142                             e.printStackTrace();
143                             e.printStackTrace();
144                             Alert alert = new Alert(Alert.AlertType.ERROR);
145                             alert.setTitle("Exception Dialog");
146                             alert.setHeaderText("Exception");
147                             alert.setContentText(e.getMessage());
148
149
150 // Create expandable Exception.
151                             StringWriter sw = new StringWriter();
152                             PrintWriter pw = new PrintWriter(sw);
153                             e.printStackTrace(pw);
154                             String exceptionText = sw.toString();
155
156                             Label label = new Label("The exception stacktrace was:");
157
158                             TextArea textArea = new TextArea(exceptionText);
159                             textArea.setEditable(false);
160                             textArea.setWrapText(true);
```

```
161
162                             textArea.setMaxWidth(Double.MAX_VALUE);
163                             textArea.setMaxHeight(Double.MAX_VALUE);
164                             GridPane.setVgrow(textArea, Priority.ALWAYS);
165                             GridPane.setHgrow(textArea, Priority.ALWAYS);
166
167                             GridPane expContent = new GridPane();
168                             expContent.setMaxWidth(Double.MAX_VALUE);
169                             expContent.add(label, 0, 0);
170                             expContent.add(textArea, 0, 1);
171
172 // Set expandable Exception into the dialog pane.
173                             alert.getDialogPane().setExpandableContent(expContent);
174
175                             alert.showAndWait();
176                         }
177
178                     }
179                 });
180             }
181
182         });
183         emailButton.setOnAction(new EventHandler<ActionEvent>() {
184             /**
185              *
186              * @param event
187              * Calls the document writer class, which then outputs everything
188              * to a spreadsheet
189              */
190
191             @Override
192             public void handle(ActionEvent event) {
193                 try {
194                     DocWriter writer = new DocWriter();
195                 } catch (Exception e) {
196                     e.printStackTrace();
197                     e.printStackTrace();
198                     Alert alert = new Alert(Alert.AlertType.ERROR);
199                     alert.setTitle("Exception Dialog");
200                     alert.setHeaderText("Exception");
201                     alert.setContentText(e.getMessage());
```

Thomas Moffat, 51337, 4042

```
202
203
204 // Create expandable Exception.
205                     StringWriter sw = new StringWriter();
206                     PrintWriter pw = new PrintWriter(sw);
207                     e.printStackTrace(pw);
208                     String exceptionText = sw.toString();
209
210                     Label label = new Label("The exception stacktrace was:");
211
212                     TextArea textArea = new TextArea(exceptionText);
213                     textArea.setEditable(false);
214                     textArea.setWrapText(true);
215
216                     textArea.setMaxWidth(Double.MAX_VALUE);
217                     textArea.setMaxHeight(Double.MAX_VALUE);
218                     GridPane.setVgrow(textArea, Priority.ALWAYS);
219                     GridPane.setHgrow(textArea, Priority.ALWAYS);
220
221                     GridPane expContent = new GridPane();
222                     expContent.setMaxWidth(Double.MAX_VALUE);
223                     expContent.add(label, 0, 0);
224                     expContent.add(textArea, 0, 1);
225
226 // Set expandable Exception into the dialog pane.
227                     alert.getDialogPane().setExpandableContent(expContent);
228
229                     alert.showAndWait();
230                 }
231             }
232         });
233         searchButton.setOnAction(new EventHandler<ActionEvent>() {
234             @Override
235             public void handle(ActionEvent event) {
236                 token = searchField.getText();
237                 ordering = orderComboBox.getValue().toString();
238                 try {
239                     new DBSearch().start(new Stage());
240                 } catch (Exception e) {
241                     e.printStackTrace();
242                     e.printStackTrace();
```

Thomas Moffat, 51337, 4042

```
243                     Alert alert = new Alert(Alert.AlertType.ERROR);
244                     alert.setTitle("Exception Dialog");
245                     alert.setHeaderText("Exception");
246                     alert.setContentText(e.getMessage());
247 
248 
249 // Create expandable Exception.
250                     StringWriter sw = new StringWriter();
251                     PrintWriter pw = new PrintWriter(sw);
252                     e.printStackTrace(pw);
253                     String exceptionText = sw.toString();
254 
255                     Label label = new Label("The exception stacktrace was:");
256 
257                     TextArea textArea = new TextArea(exceptionText);
258                     textArea.setEditable(false);
259                     textArea.setWrapText(true);
260 
261                     textArea.setMaxWidth(Double.MAX_VALUE);
262                     textArea.setMaxHeight(Double.MAX_VALUE);
263                     GridPane.setVgrow(textArea, Priority.ALWAYS);
264                     GridPane.setHgrow(textArea, Priority.ALWAYS);
265 
266                     GridPane expContent = new GridPane();
267                     expContent.setMaxWidth(Double.MAX_VALUE);
268                     expContent.add(label, 0, 0);
269                     expContent.add(textArea, 0, 1);
270 
271 // Set expandable Exception into the dialog pane.
272                     alert.getDialogPane().setExpandableContent(expContent);
273 
274                     alert.showAndWait();
275                 }
276             }
277         });
278 
279         GridPane grid = new GridPane();
280 
281         grid.setHgap(10);
282         grid.setVgap(10);
283         grid.setPadding(new Insets(25, 25, 25, 25));
```

```
284
285         Scene scene = new Scene(grid, 300, 275);
286         primaryStage.setScene(scene);
287         grid.add(dbViewButton, 0, 0, 1, 1);
288         grid.add(remakeButton, 1, 0, 1, 1);
289         remakeButton.setTextFill(Color.RED);
290         grid.add(searchButton, 0, 1, 1, 1);
291         grid.add(emailButton, 1, 1, 1, 1);
292         grid.add(searchField, 0, 2, 3, 1);
293         grid.add(new Label("Order by: "), 0, 3, 1, 1);
294         grid.add(orderComboBox, 1, 3, 2, 1);
295         GridPane.setHalignment(remakeButton, HPos.CENTER);
296         GridPane.setHalignment(emailButton, HPos.CENTER);
297         GridPane.setHalignment(dbViewButton, HPos.CENTER);
298         GridPane.setHalignment(searchButton, HPos.CENTER);
299         GridPane.setHalignment(searchField, HPos.CENTER);
300         GridPane.setHalignment(orderComboBox, HPos.CENTER);
301
302
303         primaryStage.show();
304     }
305
306 }
```

A.2 getConnection.java

```
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;
```

Thomas Moffat, 51337, 4042

```
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 + ":" + portNumber
51                 + "/";
52         System.out.println("Connecting to: " + dbms + "://" + serverName +
53                 ":3306");
54         conn = DriverManager.getConnection(JDBC_URL, userName, password);
55         return conn;
56     }
```

Thomas Moffat, 51337, 4042

```
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  }
```

Thomas Moffat, 51337, 4042

A.3 TableViewTest.java

```
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.*;
11
12 import javafx.scene.control.TableView;
13 import javafx.scene.layout.GridPane;
14 import javafx.scene.layout.Priority;
15 import javafx.stage.Stage;
16 import javafx.util.Callback;
17
18
19 import java.io.IOException;
20 import java.io.PrintWriter;
21 import java.io.StringWriter;
22 import java.sql.Connection;
23 import java.sql.ResultSet;
24 import java.sql.SQLException;
25
26 /**
27  * The class that shows everything in the database all at once. Excuse the name,
28  * I put it in as a test to see if it would work and then it worked so well that
29  * I just never changed it. This takes a global variable from the main menu for
30  * the ordering.
31  */
32 public class TableViewTest extends Application {
33
34     private TableView tableview = new TableView();
35
36     // MAIN EXECUTOR
37     public static void main(String[] args) {
38         launch(args);
39     }
```

Thomas Moffat, 51337, 4042

```
40
41      // CONNECTION DATABASE
42
43      /**
44       * @throws SQLException
45       * @throws IOException
46       */
47      public void buildData() throws SQLException, IOException {
48          Connection c;
49          ObservableList<ObservableList> data = FXCollections.observableArrayList();
50          try {
51              c = new getConnection().getConnection();
52              String SQL = "USE mydb";
53              c.createStatement().executeQuery(SQL);
54              // SQL FOR SELECTING TABLES
55              String ordering = MainMenu.ordering;
56              String order;
57              switch (ordering){
58                  case "Name A-Z":    order = "Name ASC";
59                      break;
60                  case "Name Z-A":    order = "Name DESC";
61                      break;
62                  case "Item":        order = "Orders";
63                      break;
64                  case "Order ID":    order = "ID";
65                      break;
66                  case "Squad":       order = "Squad";
67                      break;
68                  case "Customer ID:":order = "CustomerID";
69                      break;
70                  case "Payment Method":  order = "PaymentMethod";
71                      break;
72                  default:            order = "ID";
73              }
74
75              SQL = "select o.ID, o.CustomerID, c.Name, c.Email_Address, c.Squad," +
76                      "  o.Orders, o.OrderSize, o.OrderNumber, o.NameOnGarment," +
77                      "  o.PaidFor, o.PaymentMethod, i.Item from Orders o INNER" +
78                      "  JOIN Customers c ON o.CustomerID = c.ID INNER JOIN " +
79                      "Items i ON i.idItems=o.Orders ORDER BY " + order +";";
80
```

Thomas Moffat, 51337, 4042

```
81
82              // ResultSet
83
84              ResultSet rs = c.createStatement().executeQuery(SQL);
85
86              /**
87               * Gets column headings and adds them to the TableView dynamically
88               */
89
90              for (int i = 0; i < rs.getMetaData().getColumnCount(); i++) {
91
92                  // Adds the data from each column into a list
93
94                  final int j = i;
95
96                  TableColumn col = new TableColumn(rs.getMetaData()
97                          .getColumnName (i + 1));
98                  col.setCellValueFactory(
99                          new Callback<TableColumn
100                                 .CellDataFeatures<ObservableList , String>,
101                                 ObservableValue<String>>() {
102                             public ObservableValue<String> call(TableColumn
103
    ↪  .CellDataFeatures<ObservableList, String> param) {
104                                 return new SimpleStringProperty(param.
105                                         getValue().get(j).toString());
106                             }
107                         });
108                 tableview.getColumns().addAll(col);
109                 System.out.println("Column [" + i + "] ");
110
111             }
112             /********************************
113              *
114              * Data added to ObservableList *
115              *
116              ********************************/
117             while (rs.next()) {
118                 // Iterate Row
119                 ObservableList<String> row = FXCollections.observableArrayList();
120                 for (int i = 1; i <= rs.getMetaData().getColumnCount(); i++) {
```

```
121                     // Iterate Column
122                     row.add(rs.getString(i));
123                 }
124                 System.out.println("Row [1] added " + row);
125                 data.add(row);
126             }
127             // FINALLY ADDED TO TableView
128             tableview.setItems(data);
129         } catch (Exception e) {
130             e.printStackTrace();
131             System.out.println("Error on Building Data");
132             Alert alert = new Alert(Alert.AlertType.ERROR);
133             alert.setTitle("Exception Dialog");
134             alert.setHeaderText("Exception");
135             alert.setContentText(e.getMessage());
136 
137 
138 // Create expandable Exception.
139             StringWriter sw = new StringWriter();
140             PrintWriter pw = new PrintWriter(sw);
141             e.printStackTrace(pw);
142             String exceptionText = sw.toString();
143 
144             Label label = new Label("The exception stacktrace was:");
145 
146             TextArea textArea = new TextArea(exceptionText);
147             textArea.setEditable(false);
148             textArea.setWrapText(true);
149 
150             textArea.setMaxWidth(Double.MAX_VALUE);
151             textArea.setMaxHeight(Double.MAX_VALUE);
152             GridPane.setVgrow(textArea, Priority.ALWAYS);
153             GridPane.setHgrow(textArea, Priority.ALWAYS);
154 
155             GridPane expContent = new GridPane();
156             expContent.setMaxWidth(Double.MAX_VALUE);
157             expContent.add(label, 0, 0);
158             expContent.add(textArea, 0, 1);
159 
160 // Set expandable Exception into the dialog pane.
161             alert.getDialogPane().setExpandableContent(expContent);
```

Thomas Moffat, 51337, 4042

```
162
163             alert.showAndWait();
164         }
165     }
166
167     /**
168      * Makes the whole
169      */
170     @Override
171     public void start(Stage stage) throws Exception {
172         // TableView
173         buildData();
174         // Main Scene
175         Scene scene = new Scene(tableview);
176         stage.setScene(scene);
177         stage.show();
178     }
179 }
```

A.4 DocWriter.java

```
1  package kitordersystem;
2
3  /**
4   * Created by tsmoffat on 21/02/2016.
5   * A class to essentially dump the contents of the database to a spreadsheet
6   * in order to facilitate easy order sheet creation. This can be done as Word
7   * has an import from Excel option, although there is a possibility that the
8   * spreadsheet itself will be used as the order sheet
9   */
10 public class DocWriter {
11     public DocWriter() throws Exception {
12         String pythonScriptPath =
13                 "/Users/tsmoffat/kitordersystem/kitordersystem/kitordersystem" +
14                         "/docdump.py";
15         String[] cmd = new String[2];
16         cmd[0] = "python";
17         cmd[1] = pythonScriptPath;
18
19         Runtime rt = Runtime.getRuntime();
20         Process pr = rt.exec(cmd);
```

```
21
22          System.out.println("Program executed successfully");
23      }
24  }
```

A.5 DBSearch.java

```
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.*;
10  import javafx.scene.control.TableView;
11  import javafx.scene.layout.GridPane;
12  import javafx.scene.layout.Priority;
13  import javafx.stage.Stage;
14  import javafx.util.Callback;
15
16  import java.io.PrintWriter;
17  import java.io.StringWriter;
18  import java.sql.Connection;
19  import java.sql.PreparedStatement;
20  import java.sql.ResultSet;
21
22  /**
23   * Created by tsmoffat on 10/02/2016.
24   * This is a class to search the database for a specific string. Works in
25   * much the same way as the connection part in TableViewTest but uses a user
26   * inputted search string as opposed to returning everything. Uses
27   * PreparedStatement to sanitise inputs. Yay.ø
28   */
29  public class DBSearch extends Application {
30
31      private ObservableList<ObservableList> data;
32      private javafx.scene.control.TableView tableview;
33
34      // MAIN EXECUTOR
```

Thomas Moffat, 51337, 4042

```
35      public static void main(String[] args) {
36          launch(args);
37      }
38
39      public void Search() {
40          Connection c;
41          data = FXCollections.observableArrayList();
42          try {
43              c = new getConnection().getConnection();
44              String SQL = "USE mydb";
45              c.createStatement().executeQuery(SQL);
46              // SQL FOR SELECTING TABLES
47              String token = MainMenu.token;
48              String ordering = MainMenu.ordering;
49              String order;
50              switch (ordering){
51                  case "Name A-Z":    order = "Name ASC";
52                                      break;
53                  case "Name Z-A":    order = "Name DESC";
54                                      break;
55                  case "Item":        order = "Order";
56                                      break;
57                  case "Order ID":    order = "ID";
58                                      break;
59                  case "Squad":       order = "Squad";
60                                      break;
61                  case "Customer ID:":order = "CustomerID";
62                                      break;
63                  case "Payment Method":  order = "PaymentMethod";
64                      break;
65                  default:            order = "ID";
66              }
67              PreparedStatement statement = c.prepareStatement("select o.ID,  o" +
68                      ".CustomerID, c.Name, c.Email_Address, c.Squad, o.Orders," +
69                      " o.OrderSize, o.OrderNumber, o.NameOnGarment, o.PaidFor, " +
70                      "o.PaymentMethod, i.Item from Orders o INNER JOIN " +
71                      "Customers c ON o.CustomerID = c.ID INNER JOIN Items i ON" +
72                      " i.idItems=o.Orders where o.ID like " +
73                      "? or o.CustomerID like ? or c.Name like ? or c" +
74                      ".Email_Address like ? or c.Squad like ? or o.Orders like" +
75                      " ? or o.OrderSize like ? or o.OrderNumber like ? or  o" +
```

```
                    ".NameOnGarment like ? or o.PaymentMethod like ? or i" +
                    ".Item  like ? ORDER BY "+ order +";");

            statement.setString(1, "%" + token + "%");
            statement.setString(2, "%" + token + "%");
            statement.setString(3, "%" + token + "%");
            statement.setString(4, "%" + token + "%");
            statement.setString(5, "%" + token + "%");
            statement.setString(6, "%" + token + "%");
            statement.setString(7, "%" + token + "%");
            statement.setString(8, "%" + token + "%");
            statement.setString(9, "%" + token + "%");
            statement.setString(10, "%" + token + "%");
            statement.setString(11, "%" + token + "%");
            System.out.println("Executing: " + statement);
            ResultSet rs = statement.executeQuery();

            /**********************************
             *
             * TABLE COLUMN ADDED DYNAMICALLY *
             *
             **********************************/

            for (int i = 0; i < rs.getMetaData().getColumnCount(); i++) {

                // We are using non property style for making dynamic table

                final int j = i;

                TableColumn col = new TableColumn(rs.getMetaData()
                        .getColumnName(i + 1));
                col.setCellValueFactory(
                        new Callback<TableColumn.CellDataFeatures
                                <ObservableList, String>, ObservableValue
                                <String>>() {
                            public ObservableValue<String> call(TableColumn.

    ↪  CellDataFeatures<ObservableList, String> param) {
                                return new SimpleStringProperty(param
                                        .getValue().get(j).toString());
                            }
```

Thomas Moffat, 51337, 4042

```
116                     });
117             tableview.getColumns().addAll(col);
118             System.out.println("Column [" + i + "] ");
119 
120         }
121         /********************************
122          *
123          * Data added to ObservableList *
124          *
125          ********************************/
126         while (rs.next()) {
127             // Iterate Row
128             ObservableList<String> row = FXCollections.observableArrayList();
129             for (int i = 1; i <= rs.getMetaData().getColumnCount(); i++) {
130                 // Iterate Column
131                 row.add(rs.getString(i));
132             }
133             System.out.println("Row [1] added " + row);
134             data.add(row);
135         }
136         // FINALLY ADDED TO TableView
137         tableview.setItems(data);
138     } catch (Exception e) {
139         e.printStackTrace();
140         System.out.println("Error on Building Data");
141         e.printStackTrace();
142         Alert alert = new Alert(Alert.AlertType.ERROR);
143         alert.setTitle("Exception Dialog");
144         alert.setHeaderText("Exception");
145         alert.setContentText(e.getMessage());
146 
147 
148 // Create expandable Exception.
149         StringWriter sw = new StringWriter();
150         PrintWriter pw = new PrintWriter(sw);
151         e.printStackTrace(pw);
152         String exceptionText = sw.toString();
153 
154         Label label = new Label("The exception stacktrace was:");
155 
156         TextArea textArea = new TextArea(exceptionText);
```

```
157             textArea.setEditable(false);
158             textArea.setWrapText(true);
159 
160             textArea.setMaxWidth(Double.MAX_VALUE);
161             textArea.setMaxHeight(Double.MAX_VALUE);
162             GridPane.setVgrow(textArea, Priority.ALWAYS);
163             GridPane.setHgrow(textArea, Priority.ALWAYS);
164 
165             GridPane expContent = new GridPane();
166             expContent.setMaxWidth(Double.MAX_VALUE);
167             expContent.add(label, 0, 0);
168             expContent.add(textArea, 0, 1);
169 
170 // Set expandable Exception into the dialog pane.
171             alert.getDialogPane().setExpandableContent(expContent);
172 
173             alert.showAndWait();
174         }
175     }
176 
177     /**
178      *
179      */
180     @Override
181     public void start(Stage stage) throws Exception {
182         // TableView
183         tableview = new TableView();
184         Search();
185         // Main Scene
186         Scene scene = new Scene(tableview);
187         stage.setScene(scene);
188         stage.show();
189     }
190 }
```

A.6 CSVReader.java

```
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  }
```

A.7 DBReset.java

```
1   package kitordersystem;
2
3   import javafx.scene.control.Alert;
4   import javafx.scene.control.Label;
5   import javafx.scene.control.TextArea;
6   import javafx.scene.layout.GridPane;
7   import javafx.scene.layout.Priority;
8   import org.apache.ibatis.jdbc.ScriptRunner;
9   import org.xml.sax.SAXException;
10
11  import javax.xml.parsers.ParserConfigurationException;
12  import java.io.*;
13  import java.sql.Connection;
14  import java.sql.SQLException;
15
16  /**
17   * How the program resets the database if everything goes wrong with it, or
18   * it gets too cluttered. Most of the heavy lifting is done through Mybatis,
19   * which is far more powerful than this project needs but it works very well.
20   */
21  public class DBReset {
22
23      public DBReset(){
24          String SQLFilePath = "/Users/tsmoffat/kitordersystem/kitordersystem/" +
25                  "kitordersystem/KitOrderProper.sql";
26          String sqlCustomerExport = "SELECT * INTO OUTFILE " +
27                  "'/Users/tsmoffat/kitordersystem/kitordersystem" +
28                  "/kitordersystem/customer.csv'\n" +
29                  "    FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '\"'\n" +
30                  "    LINES TERMINATED BY '\\n'\n" +
31                  "FROM Customers";
32          String sqlOrderExport = "SELECT * INTO OUTFILE '/Users/tsmoffat/" +
33                  "kitordersystem/kitordersystem/kitordersystem/order.csv'\n" +
34                  "    FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '\"'\n" +
35                  "    LINES TERMINATED BY '\\n'\n" +
36                  "FROM Orders";
```

```
37          try{
38              System.out.println("RESETTING");
39              Connection conn = new getConnection().getConnection();
40              conn.createStatement().executeQuery("Use mydb");
41              conn.createStatement().executeQuery(sqlCustomerExport);
42              conn.createStatement().executeQuery(sqlOrderExport);
43              ScriptRunner sr = new ScriptRunner(conn);
44              Reader reader = new BufferedReader(new FileReader(SQLFilePath));
45              sr.runScript(reader);
46          } catch (SAXException | SQLException | ParserConfigurationException |
47                  IOException e) {
48                  e.printStackTrace();
49                  Alert alert = new Alert(Alert.AlertType.ERROR);
50                  alert.setTitle("Exception Dialog");
51                  alert.setHeaderText("Exception");
52                  alert.setContentText(e.getMessage());
53
54
55  // Create expandable Exception.
56                  StringWriter sw = new StringWriter();
57                  PrintWriter pw = new PrintWriter(sw);
58                  e.printStackTrace(pw);
59                  String exceptionText = sw.toString();
60
61                  Label label = new Label("The exception stacktrace was:");
62
63                  TextArea textArea = new TextArea(exceptionText);
64                  textArea.setEditable(false);
65                  textArea.setWrapText(true);
66
67                  textArea.setMaxWidth(Double.MAX_VALUE);
68                  textArea.setMaxHeight(Double.MAX_VALUE);
69                  GridPane.setVgrow(textArea, Priority.ALWAYS);
70                  GridPane.setHgrow(textArea, Priority.ALWAYS);
71
72                  GridPane expContent = new GridPane();
73                  expContent.setMaxWidth(Double.MAX_VALUE);
74                  expContent.add(label, 0, 0);
75                  expContent.add(textArea, 0, 1);
76
77  // Set expandable Exception into the dialog pane.
```

Thomas Moffat, 51337, 4042

```
78                  alert.getDialogPane().setExpandableContent(expContent);
79
80                  alert.showAndWait();
81          } catch (Exception e){
82              System.err.println("Failed to execute " + SQLFilePath + ". The  " +
83                      "error is " + e.getMessage());
84              Alert alert = new Alert(Alert.AlertType.ERROR);
85              alert.setTitle("Exception Dialog");
86              alert.setHeaderText("Exception");
87              alert.setContentText(e.getMessage());
88
89
90  // Create expandable Exception.
91              StringWriter sw = new StringWriter();
92              PrintWriter pw = new PrintWriter(sw);
93              e.printStackTrace(pw);
94              String exceptionText = sw.toString();
95
96              Label label = new Label("The exception stacktrace was:");
97
98              TextArea textArea = new TextArea(exceptionText);
99              textArea.setEditable(false);
100             textArea.setWrapText(true);
101
102             textArea.setMaxWidth(Double.MAX_VALUE);
103             textArea.setMaxHeight(Double.MAX_VALUE);
104             GridPane.setVgrow(textArea, Priority.ALWAYS);
105             GridPane.setHgrow(textArea, Priority.ALWAYS);
106
107             GridPane expContent = new GridPane();
108             expContent.setMaxWidth(Double.MAX_VALUE);
109             expContent.add(label, 0, 0);
110             expContent.add(textArea, 0, 1);
111
112 // Set expandable Exception into the dialog pane.
113             alert.getDialogPane().setExpandableContent(expContent);
114
115             alert.showAndWait();
116         }
117
118     }
```

Thomas Moffat, 51337, 4042

```
119
120 }
```

A.8 KitOrderProper.sql

```
1  SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
2  SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
3  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,
```

```
  `OrderNumber` VARCHAR(45) NOT NULL,
  `CustomerID` INT NULL,
  `NameOnGarment` VARCHAR(45) NULL,
  `PaidFor` TINYINT(1) NOT NULL,
  `PaymentMethod` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`ID`),
  INDEX `Order1_idx` (`Orders` ASC),
  INDEX `CustomerID_idx` (`CustomerID` ASC),
  CONSTRAINT `Order1`
    FOREIGN KEY (`Orders`)
    REFERENCES `mydb`.`Items` (`idItems`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `CustomerID`
    FOREIGN KEY (`CustomerID`)
    REFERENCES `mydb`.`Customers` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

A.9 WebInput.html

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
    <link rel="stylesheet" type="text/css" href="styling.css" />
    <meta char-set="utf-8"/>
    <title>Customer Input</title>
  </head>
  <body>
<div>
    <form id="customerform" method="post" >
      Name:<br>
      <input type="text" name="name"><br>
      Email:<br/>
      <input type="email" name="email"/><br/>
      Squad:<br/>
```

Thomas Moffat, 51337, 4042

```
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          })
```

Thomas Moffat, 51337, 4042

```
        window.location="./WebInputOrder.html"
        return false;
      })
    </script>
  </body>
</html>
```

A.10 WebInputOrder.html

```
<!DOCTYPE html>
<html>
<head>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
  <link rel="stylesheet" type="text/css" href="styling.css" />
  <meta char-set="utf-8" />
  <title>Order Input</title>
</head>
<body>
  <div>
  Please enter in your items one at a time<br />
  Item List:<br />
  Poolside Shirt: £12<br />
  Polo Shirt: £12<br />
  Over-the-head Hoodie: £17<br />
  Zippy Hoodie: £20<br />
  Rucksack: £22<br />
  Holdall: £27<br  />
  Hat: £6<br />
  Shorts: £4<br />
  <form method="post" id="orderform">
    Item:<br />
    <select name="item">
      <option value="Poolside Shirt">
        Poolside Shirt
      </option>
      <option value="Polo Shirt">Polo Shirt</option>

      <option value="Hoodie">
        Hoodie
      </option>
      <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 />
```

Thomas Moffat, 51337, 4042

```
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>
```

A.11 styling.css

```
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%
```

Thomas Moffat, 51337, 4042

```
}
input[type=email] {
  padding: 12px 20px;
  margin: 8px 0;
  box-sizing: border-box;
  font-size: 1.2em;
  width: 100%
}
input[type=text]:focus {
  border: 3px solid #555;
  outline: none;
}
input[type=email]:focus {
  border: 3px solid #555;
  outline: none;
}
select {
  padding: 16px 20px;
  border: none;
  border-radius: 4px;
  background-color: #f1f1f1;
  font-size: 1.2em;
  width: 100%;
}
input[type=submit] {
  background-color: #4CAF50;
  border: none;
  color: white;
  padding: 32px, 64px;
  text-decoration: none;
  margin: 4px 2px;
  cursor: pointer;
  font-size: 1.5em;
  width: 100%;
}
div {
  font-family: "Arial", Helvetica, sans-serif;
  font-size: 1em;


}
```

Thomas Moffat, 51337, 4042

```
input[type=number]{
  padding: 12px 20px;
  margin: auto;
  box-sizing: border-box;
  font-size: 1.2em;
  width: 50%
}
input[type=number]:focus {
  border: 3px solid #555;
  outline: none;
}
```

A.12 dbcustomerupdate.php

```
<?php
session_start();
if (file_exists('kitorder2.xml')) {
    $xml = simplexml_load_file('kitorder2.xml');
    $dbms = $xml->connection->dbms;
    $dbname = $xml->connection->database_name;
    $user_name = $xml->connection->user_name;
    $password = $xml->connection->password;
    $port_numberraw = $xml->connection->port_number;
    $port_number = (int) $port_numberraw;
    $servername = $xml->connection->server_name;
    $conn = new mysqli($servername, $user_name, $password, $dbname, $port_number);
    if ($conn->connect_error){
        die("Connection failed: " . $conn->connect_error);
    }


        $name = test_input($_POST['name']);
        $email = test_input($_POST['email']);
        $squad = test_input($_POST['squad']);

        $paymethod = test_input($_POST['paymethod']);

        $_SESSION["paymethod"]=$paymethod;
        $stmt = $conn->prepare("INSERT INTO Customers (Name, Email_Address, Squad) VALUES (?,
↪  ?, ?);");
        $stmt->bind_param("sss", $name, $email, $squad);
```

Thomas Moffat, 51337, 4042

```
        $stmt->execute();

        $sql = $conn->prepare("SELECT MAX(ID) FROM Customers WHERE Name = ?;");
        $sql->bind_param("s", $name);
        $sql->execute();
        $result = $sql->get_result()->fetch_row()[0];
        $_SESSION["ID"] = $result;


} else {
    exit('Failed to open kitorder2.xml.');
}
function test_input($data){
    $data=trim($data);
    $data= stripslashes($data);
    $data=htmlspecialchars($data);
    return $data;
}
?>
```

A.13 DBOrderUpdate.php

```
<?php
// The file test.xml contains an XML document with a root element
// and at least an element /[root]/title.
session_start();
error_reporting(E_ALL);
ini_set('display_errors', 1);
if (file_exists('kitorder2.xml')) {
    $xml = simplexml_load_file('kitorder2.xml');
    $dbms = $xml->connection->dbms;
    $dbname = $xml->connection->database_name;
    $user_name = $xml->connection->user_name;
    $password = $xml->connection->password;
    $port_numberraw = $xml->connection->port_number;
    $port_number = (int) $port_numberraw;
    $servername = $xml->connection->server_name;
    $conn = new mysqli($servername, $user_name, $password, $dbname, $port_number);
    if ($conn->connect_error){
        die("Connection failed: " . $conn->connect_error);
    }
```

Thomas Moffat, 51337, 4042

```
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  ?>
```

A.14 docdump.py

```
#pylint: disable=C0103

import xml.etree.ElementTree as ET
import MySQLdb
from openpyxl import Workbook
tree = ET.parse('/Users/tsmoffat/kitordersystem/kitordersystem/kitordersystem/kitorder.xml')
root = tree.getroot()
dbms = root[0][0].text
database_name = root[0][1].text
user_name = root[0][2].text
password = root[0][3].text
port_number = root[0][4].text
server_name = root[0][5].text
db = MySQLdb.connect(host=server_name, user=user_name, passwd=password, db=database_name)
cur = db.cursor()
wb = Workbook()
ws1 = wb.active
for x in range(1, 9):
    xstring = str(x)
    cur.execute("SELECT Item from Items where idItems = " + xstring + ";")
    for row in cur.fetchall():
        item = row[0]
        print item
        cur.execute("SELECT c.Name, o.OrderSize, o.NameOnGarment FROM Orders o INNER JOIN
 ↪  Customers c on o.CustomerID = c.ID where o.Orders = " + xstring + ";")
        row = cur.fetchone()
        if x == 1:
            ws1.title = item
            ws1['A1'] = item
            ws1['A2'] = 'Name'
            ws1['B2'] = 'Size'
            ws1['C2'] = 'Printed On'
        else:
            ws1 = wb.create_sheet()
            ws1.title = item
            ws1['A1'] = item
            ws1['A2'] = 'Name'
            ws1['B2'] = 'Size'
            ws1['C2'] = 'Printed On'
```

Thomas Moffat, 51337, 4042

```
        i = 3
        while row is not None:
            d = ws1.cell(row=i, column=1)
            e = ws1.cell(row=i, column=2)
            f = ws1.cell(row=i, column=3)
            d.value = row[0]
            e.value = row[1]
            f.value = row[2]
            i += 1
            row = cur.fetchone()
    wb.save('Tilehurst_Order_Raw.xlsx')

```

A.15 kitorder.xml and kitorder2.xml

```
<?xml version="1.0" standalone = "yes"?>
<!DOCTYPE database [
<!ELEMENT IP (#PCDATA)>
<!ELEMENT user_name (#PCDATA)>
<!ELEMENT password (#PCDATA)>
<!ELEMENT database (connection*)>
<!ELEMENT connection (dbms, database_name, user_name, password, port_number, server_name)>
<!ELEMENT dbms (#PCDATA)>
<!ELEMENT database_name (#PCDATA)>
<!ELEMENT port_number (#PCDATA)>
<!ELEMENT server_name (#PCDATA)>
]>
<database>
  <connection>
          <dbms>mysql</dbms>
    <database_name>mydb</database_name>
    <user_name>root</user_name>
    <password>root</password>
    <port_number>3306</port_number>
    <server_name>localhost</server_name>
  </connection>
</database>
```

```
<?xml version="1.0" encoding="utf-8"?>
<database>
  <connection>
```

```
4           <dbms>mysql</dbms>
5       <database_name>mydb</database_name>
6       <user_name>root</user_name>
7       <password>root</password>
8       <port_number>3306</port_number>
9       <server_name>localhost</server_name>
10    </connection>
11  </database>
```

CLASS HIERARCHY

Classes

- java.lang.Object
  - javafx.application.Application
    - kitordersystem.DBSearch (in B.4, page 94)
    - kitordersystem.MainMenu (in B.8, page 98)
    - kitordersystem.TableViewTest (in B.11, page 102)
  - kitordersystem.CSVReader (in B.1, page 92)
  - kitordersystem.DBCreate (in B.2, page 93)
  - kitordersystem.DBReset (in B.3, page 94)
  - kitordersystem.DocWriter (in B.5, page 95)
  - kitordersystem.Main (in B.7, page 98)
  - kitordersystem.ScriptReader (in B.9, page 100)
  - kitordersystem.TableView (in B.10, page 101)
  - kitordersystem.getConnection (in B.6, page 96)

B PACKAGE KITORDERSYSTEM

Package Contents *Page*

Classes

B.1 Class CSVReader

This is called when the database is reset, from the main menu, its only function is to repopulate the items database, which it does from a CSV file so that the contents of the table can be edited if the items ever change without having to recompile the whole program.

B.1.1 Declaration

```
public class CSVReader
 extends java.lang.Object
```

B.1.2 Constructor summary

CSVReader()

B.1.3 Constructors

- **CSVReader**

```
public CSVReader() throws java.io.IOException, java.sql.
    SQLException, org.xml.sax.SAXException, javax.xml.parsers.
    ParserConfigurationException
```

– **Throws**
  * `java.io.IOException` –
  * `java.sql.SQLException` –
  * `org.xml.sax.SAXException` –
  * `javax.xml.parsers.ParserConfigurationException` –

B.2 Class DBCreate

The class that makes at least some of the fun happen, this was put in so the end user has the option to completely drop the table and rebuild it if something gets messed up for some reason or another.

B.2.1 Declaration

```
public class DBCreate
 extends java.lang.Object
```

B.2.2 Constructor summary

DBCreate()

B.2.3 Method summary

main(String[])

B.2.4 Constructors

- **DBCreate**

```
public DBCreate() throws java.io.IOException, java.sql.
    SQLException, org.xml.sax.SAXException, javax.xml.parsers.
    ParserConfigurationException
```

  - **Throws**
    * `java.io.IOException` –
    * `java.sql.SQLException` –
    * `org.xml.sax.SAXException` –
    * `javax.xml.parsers.ParserConfigurationException` –

B.2.5 Methods

- **main**

```
public static void main(java.lang.String[] args) throws java.io.
    IOException, java.sql.SQLException, org.xml.sax.SAXException,
     javax.xml.parsers.ParserConfigurationException
```

B.3 Class DBReset

How the program resets the database if everything goes wrong with it, or it gets too cluttered. Most of the heavy lifting is done through Mybatis, which is far more powerful than this project needs but it works very well.

B.3.1 Declaration

```
public class DBReset
 extends java.lang.Object
```

B.3.2 Constructor summary

DBReset()

B.3.3 Constructors

- **DBReset**

```
public DBReset()
```

B.4 Class DBSearch

Created by tsmoffat on 10/02/2016. This is a class to search the database for a specific string. Works in much the same way as the connection part in TableViewTest but uses a user inputted search string as opposed to returning everything. Uses PreparedStatement to sanitise inputs. Yay.ø

B.4.1 Declaration

```
public class DBSearch
 extends javafx.application.Application
```

B.4.2 Constructor summary

DBSearch()

B.4.3 Method summary

main(String[])
Search()
start(Stage)

B.4.4 Constructors

- **DBSearch**

```
public DBSearch()
```

B.4.5 Methods

- **main**

```
public static void main(java.lang.String[] args)
```

- **Search**

```
public void Search()
```

- **start**

```
public abstract void start(javafx.stage.Stage arg0) throws java.
    lang.Exception
```

B.4.6 Members inherited from class Application

`javafx.application.Application`

- `public final HostServices` **getHostServices**`()`
- `public final Application.Parameters` **getParameters**`()`
- `public static String` **getUserAgentStylesheet**`()`
- `public void` **init**`() throws java.lang.Exception`
- `public static void` **launch**`(java.lang.Class` **arg0**`, java.lang.String[]` **arg1**`)`
- `public static void` **launch**`(java.lang.String[]` **arg0**`)`
- `public final void` **notifyPreloader**`(Preloader.PreloaderNotification` **arg0**`)`
- `public static void` **setUserAgentStylesheet**`(java.lang.String` **arg0**`)`
- `public abstract void` **start**`(javafx.stage.Stage` **arg0**`) throws java.lang.Exception`
- `public void` **stop**`() throws java.lang.Exception`
- `public static final` **STYLESHEET_CASPIAN**
- `public static final` **STYLESHEET_MODENA**

B.5 Class DocWriter

Created by tsmoffat on 21/02/2016. A class to essentially dump the contents of the database to a spreadsheet in order to facilitate easy order sheet creation. This can be done as Word has an import from Excel option, although there is a possibility that the spreadsheet itself will be used as the order sheet

B.5.1 Declaration

```
public class DocWriter
 extends java.lang.Object
```

B.5.2 Constructor summary

DocWriter()

B.5.3 Constructors

- **DocWriter**

```
public DocWriter() throws java.lang.Exception
```

B.6 Class getConnection

This class gets all the connection properties by opening an XML file (set as kitorder.xml) and then use the nodes in that to connect to the database itself

B.6.1 Declaration

```
public class getConnection
 extends java.lang.Object
```

B.6.2 Field summary

dbms
dbName
password
userName

B.6.3 Constructor summary

getConnection()

B.6.4 Method summary

getConnection()
setProperties(String)

B.6.5 Fields

- `public java.lang.String` **dbms**
- `public java.lang.String` **dbName**

Thomas Moffat, 51337, 4042

- `public java.lang.String` **userName**
- `public java.lang.String` **password**

B.6.6 Constructors

- **getConnection**

```
public getConnection()
```

B.6.7 Methods

- **getConnection**

```
public java.sql.Connection getConnection() throws java.sql.
    SQLException, java.io.IOException, org.xml.sax.SAXException,
    javax.xml.parsers.ParserConfigurationException
```

  - **Returns** –
  - **Throws**
    * `java.sql.SQLException` –
    * `java.io.IOException` –
    * `org.xml.sax.SAXException` –
    * `javax.xml.parsers.ParserConfigurationException` –

- **setProperties**

```
public void setProperties(java.lang.String fileName) throws java
    .io.FileNotFoundException, java.io.IOException, java.util.
    InvalidPropertiesFormatException, org.xml.sax.SAXException,
    javax.xml.parsers.ParserConfigurationException
```

  - **Parameters**
    * `fileName` – - The file where the connection details can be found, it's easier to pass it through from elsewhere
  - **Throws**
    * `java.io.FileNotFoundException` –
    * `java.io.IOException` –
    * `java.util.InvalidPropertiesFormatException` –
    * `org.xml.sax.SAXException` –
    * `javax.xml.parsers.ParserConfigurationException` –

B.7 Class Main

B.7.1 Declaration

```
public class Main
  extends java.lang.Object
```

B.7.2 Constructor summary

Main()

B.7.3 Method summary

main(String[])

B.7.4 Constructors

- **Main**

```
public Main()
```

B.7.5 Methods

- **main**

```
public static void main(java.lang.String[] args)
```

B.8 Class MainMenu

This is, as the name suggests, the main menu for my programme. Where all the magic happens.

B.8.1 Declaration

```
public class MainMenu
  extends javafx.application.Application
```

B.8.2 Field summary

ordering
token

B.8.3 Constructor summary

MainMenu()

B.8.4 Method summary

main(String[])
start(Stage) Starts the stage, fairly self-evident

B.8.5 Fields

- `public static java.lang.String` **token**
- `public static java.lang.String` **ordering**

B.8.6 Constructors

- **MainMenu**

```
public MainMenu()
```

B.8.7 Methods

- **main**

```
public static void main(java.lang.String[] args) throws java.sql
    .SQLException, java.io.IOException
```

- **start**

```
public void start(javafx.stage.Stage primaryStage)
```

  - **Description**

    Starts the stage, fairly self-evident

B.8.8 Members inherited from class Application

`javafx.application.Application`

- `public final HostServices` **getHostServices**`()`
- `public final Application.Parameters` **getParameters**`()`
- `public static String` **getUserAgentStylesheet**`()`
- `public void` **init**`() throws java.lang.Exception`
- `public static void` **launch**`(java.lang.Class` **arg0**`, java.lang.String[]` **arg1**`)`
- `public static void` **launch**`(java.lang.String[]` **arg0**`)`
- `public final void` **notifyPreloader**`(Preloader.PreloaderNotification` **arg0**`)`
- `public static void` **setUserAgentStylesheet**`(java.lang.String` **arg0**`)`
- `public abstract void` **start**`(javafx.stage.Stage` **arg0**`) throws java.lang.Exception`
- `public void` **stop**`() throws java.lang.Exception`
- `public static final` **STYLESHEET_CASPIAN**
- `public static final` **STYLESHEET_MODENA**

B.9 Class ScriptReader

B.9.1 Declaration

```
public class ScriptReader
 extends java.lang.Object
```

B.9.2 Constructor summary

ScriptReader(Connection, boolean, boolean)

B.9.3 Method summary

runScript(Reader) runs the SQL script, which is read in using Reader
setDelimiter(String, boolean)
setErrorLogWriter(PrintWriter) Sets errorLogWriter
setLogWriter(PrintWriter) Sets logWriter

B.9.4 Constructors

- **ScriptReader**

```
public ScriptReader(java.sql.Connection connection,boolean
    autoCommit,boolean stopOnError)
```

B.9.5 Methods

- **runScript**

```
public void runScript(java.io.Reader reader) throws java.io.
    IOException, java.sql.SQLException
```

  - **Description**

    runs the SQL script, which is read in using Reader

  - **Parameters**

    * `reader` – - source of the script

  - **Throws**

    * `java.io.IOException` –
    * `java.sql.SQLException` –

- **setDelimiter**

```
public void setDelimiter(java.lang.String delimiter,boolean
    fullLineDelimiter)
```

- **setErrorLogWriter**

```
public void setErrorLogWriter(java.io.PrintWriter errorLogWriter
    )
```

  - **Description**

    Sets errorLogWriter

  - **Parameters**

    * `errorLogWriter` – - new value of errorLogWriter

- **setLogWriter**

```
public void setLogWriter(java.io.PrintWriter logWriter)
```

  - **Description**

    Sets logWriter

  - **Parameters**

    * `logWriter` – - new value of logWriter

B.10 Class TableView

B.10.1 Declaration

```
public class TableView
  extends java.lang.Object
```

B.10.2 Constructor summary

TableView()

B.10.3 Method summary

buildData()

B.10.4 Constructors

- **TableView**

```
public TableView()
```

B.10.5 Methods

- **buildData**

```
public void buildData() throws java.sql.SQLException, java.io.
    IOException
```

B.11 Class TableViewTest

The class that shows everything in the database all at once. Excuse the name, I put it in as a test to see if it would work and then it worked so well that I just never changed it. This takes a global variable from the main menu for the ordering.

B.11.1 Declaration

```
public class TableViewTest
 extends javafx.application.Application
```

B.11.2 Constructor summary

TableViewTest()

B.11.3 Method summary

buildData()
main(String[])
start(Stage) Makes the whole

B.11.4 Constructors

- **TableViewTest**

```
public TableViewTest()
```

B.11.5 Methods

- **buildData**

```
public void buildData() throws java.sql.SQLException, java.io.
    IOException
```

  - **Throws**
    * `java.sql.SQLException` –

* `java.io.IOException` –

- **main**

```
public static void main(java.lang.String[] args)
```

- **start**

```
public void start(javafx.stage.Stage stage) throws java.lang.
    Exception
```

  - **Description**

    Makes the whole

B.11.6 *Members inherited from class Application*

`javafx.application.Application`

- `public final HostServices` **getHostServices**`()`
- `public final Application.Parameters` **getParameters**`()`
- `public static String` **getUserAgentStylesheet**`()`
- `public void` **init**`() throws java.lang.Exception`
- `public static void` **launch**`(java.lang.Class` **arg0**`, java.lang.String[]` **arg1**`)`
- `public static void` **launch**`(java.lang.String[]` **arg0**`)`
- `public final void` **notifyPreloader**`(Preloader.PreloaderNotification` **arg0**`)`
- `public static void` **setUserAgentStylesheet**`(java.lang.String` **arg0**`)`
- `public abstract void` **start**`(javafx.stage.Stage` **arg0**`) throws java.lang.Exception`
- `public void` **stop**`() throws java.lang.Exception`
- `public static final` **STYLESHEET_CASPIAN**
- `public static final` **STYLESHEET_MODENA**

C TESTING SCREENSHOTS

C.1 Web Form Testing Screenshots

C.1.1 *Customer Detail Form Testing Screenshots*

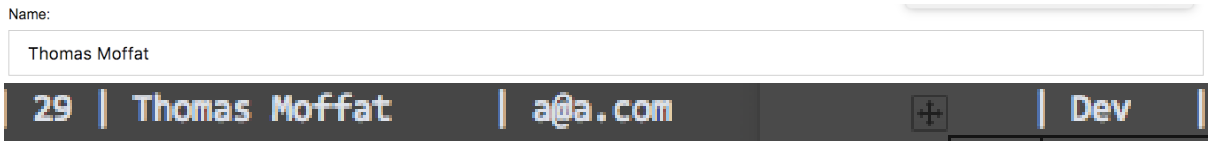


Figure 15: Evidence for test 1

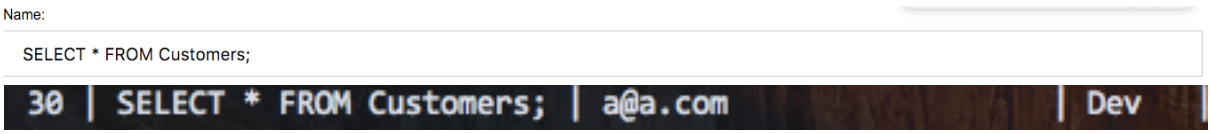


Figure 16: Evidence for test 2

Figure 17: Evidence for test 3

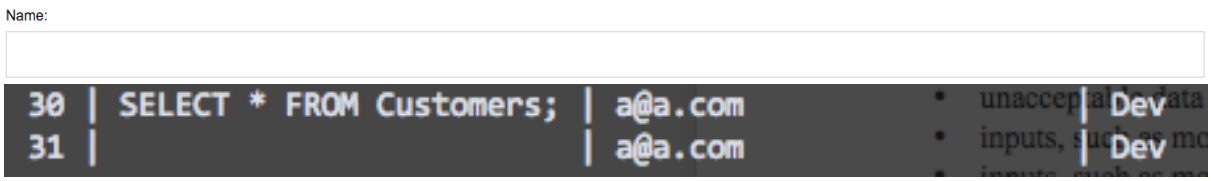


Figure 18: Evidence for test 4

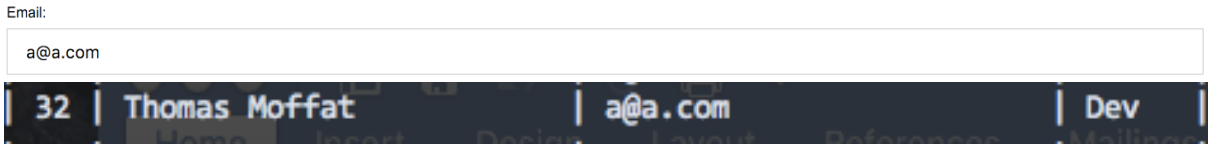


Figure 19: Evidence for test 5

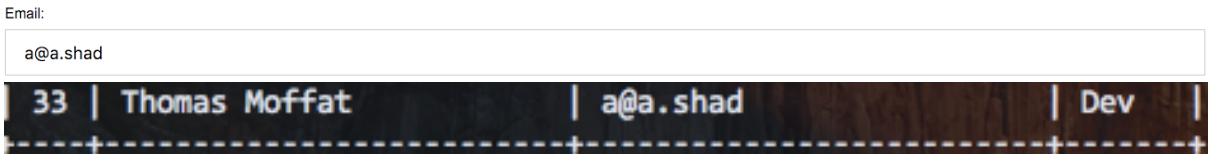


Figure 21: Evidence for test 7

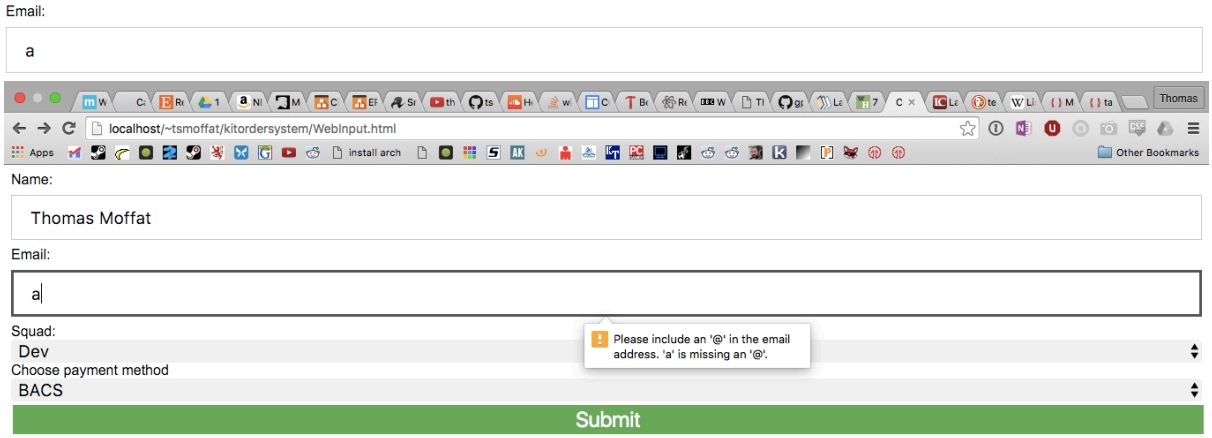


Figure 20: Evidence for test 6

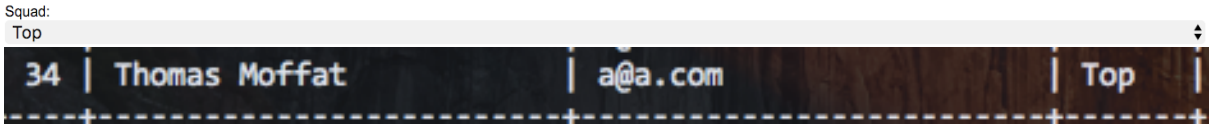


Figure 22: Evidence for test 8

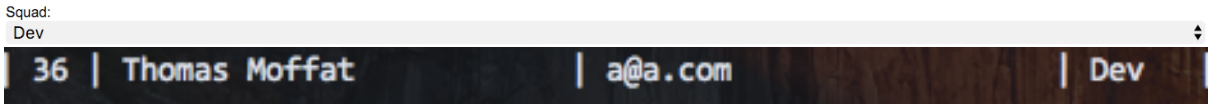


Figure 23: Evidence for test 9

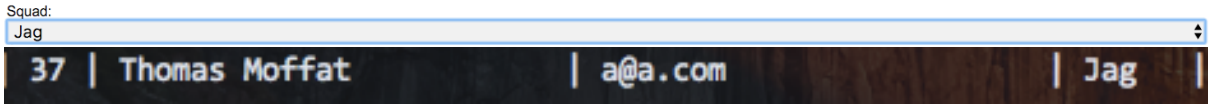


Figure 24: Evidence for test 10

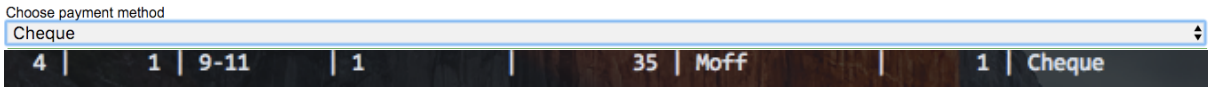


Figure 25: Evidence for test 11

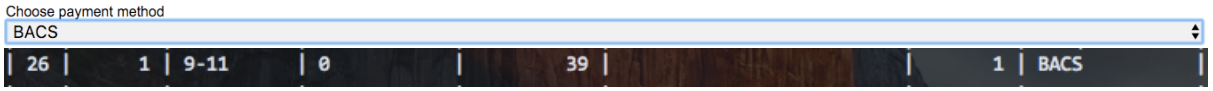


Figure 26: Evidence for test 12

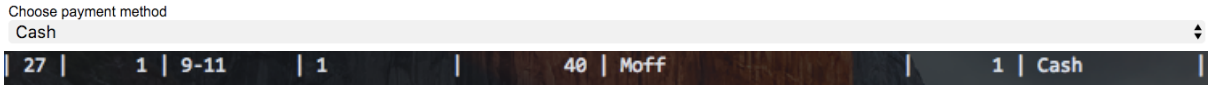


Figure 27: Evidence for test 13

c.1.2 Order Form Testing Screenshots

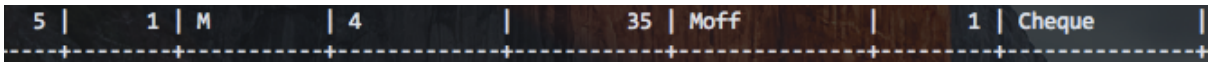


Figure 28: Evidence for test 1

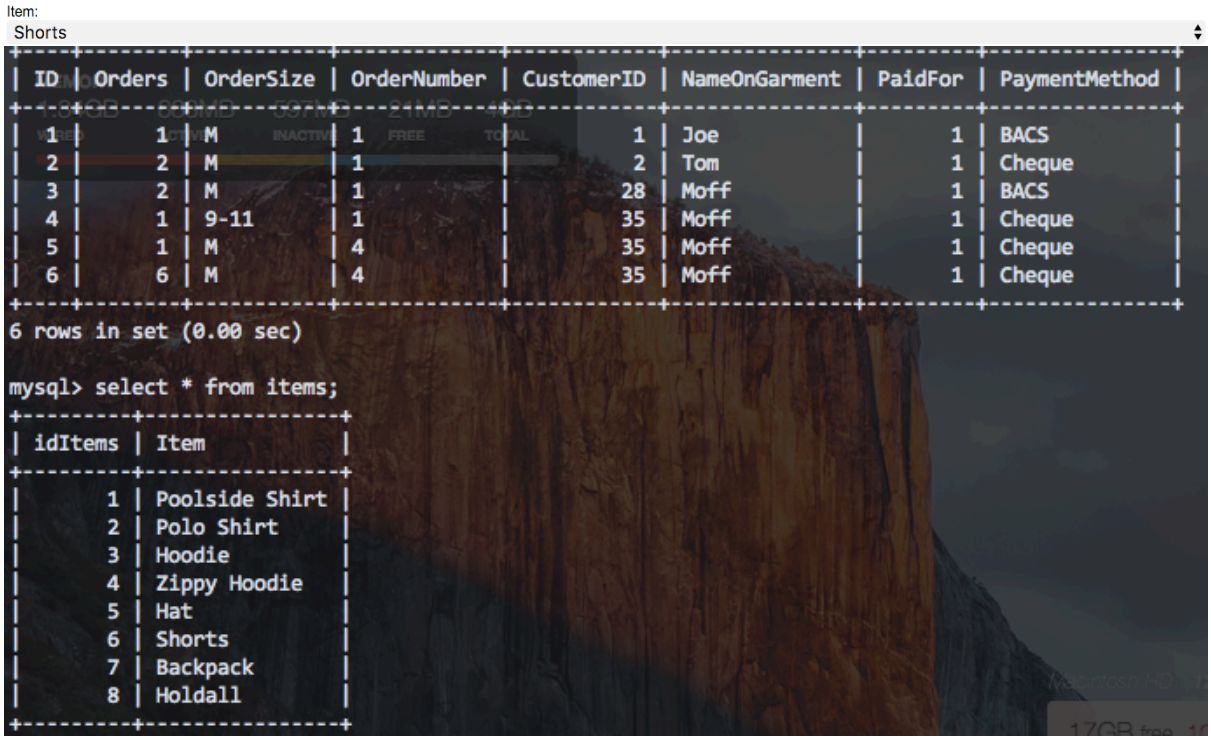


Figure 29: Evidence for test 2, please note the reference table beneath

Figure 30: Evidence for test 3

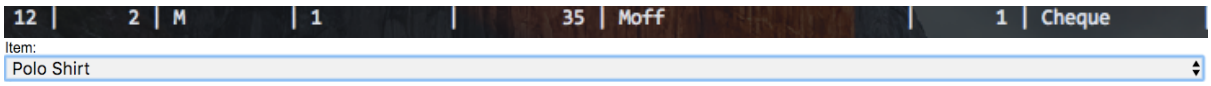


Figure 31: Evidence for test 4

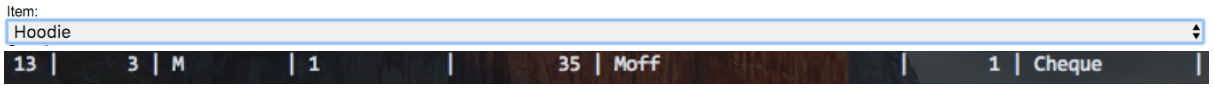


Figure 32: Evidence for test 5

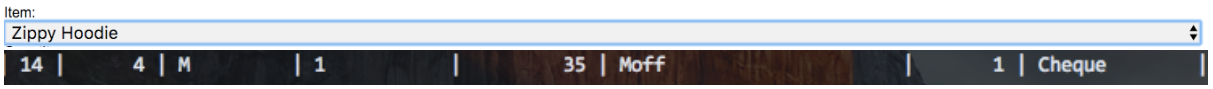


Figure 33: Evidence for test 6

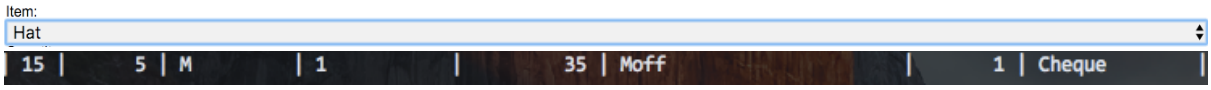


Figure 34: Evidence for test 7

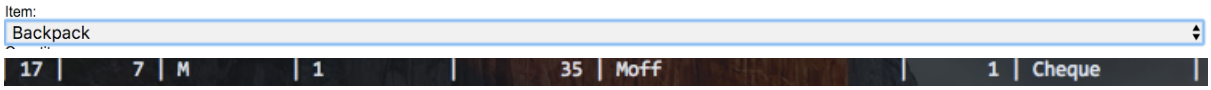


Figure 35: Evidence for test 8

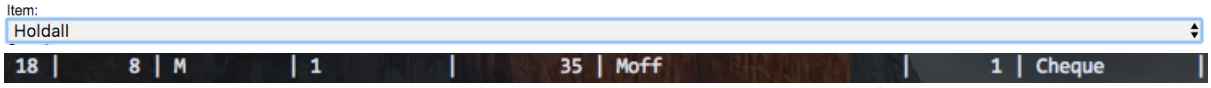


Figure 36: Evidence for test 9

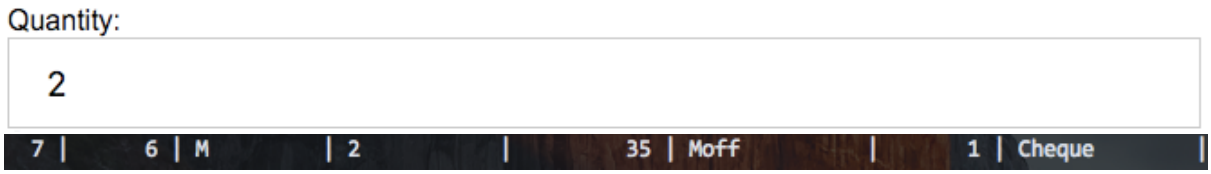


Figure 37: Evidence for test 10

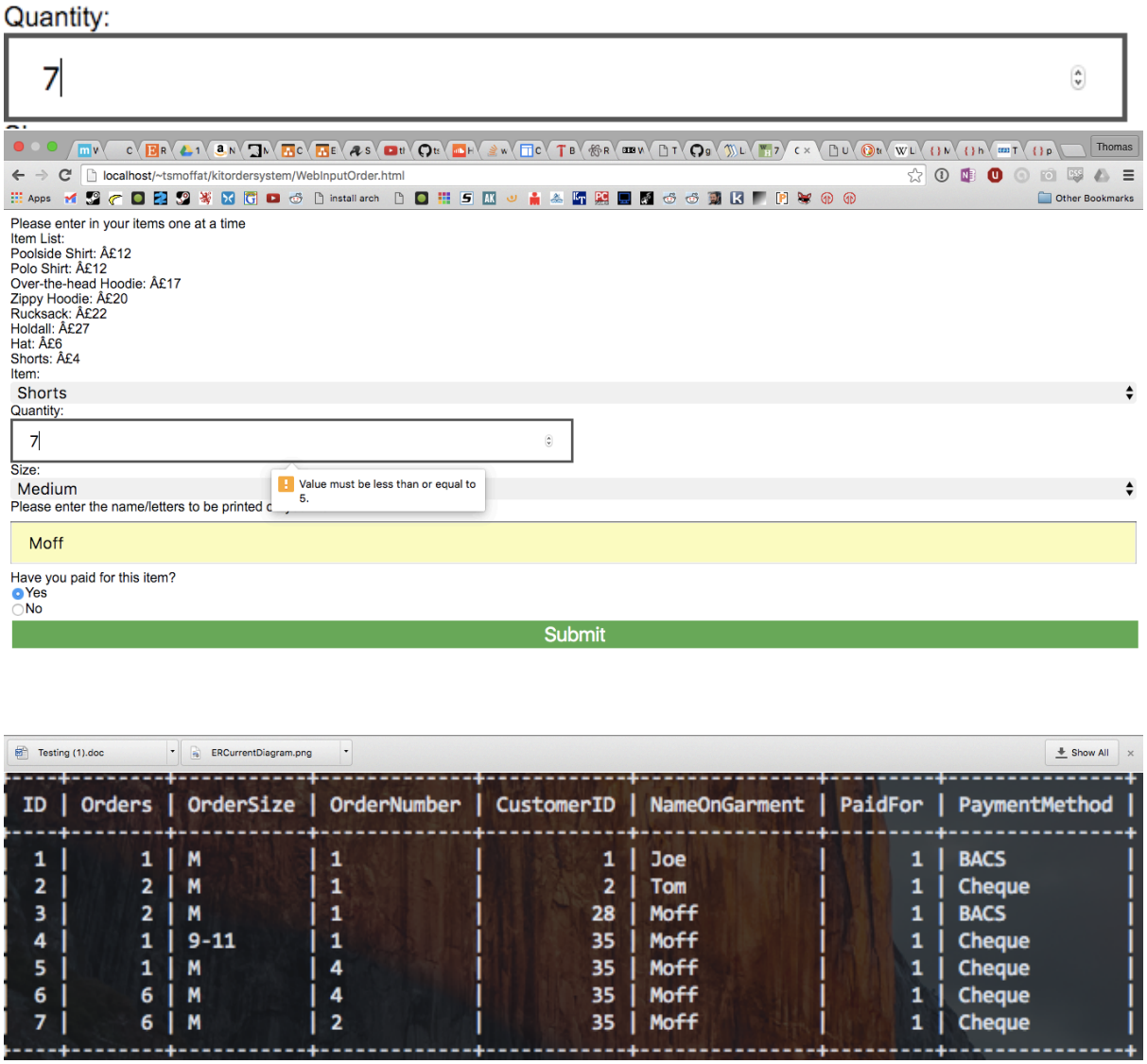


Figure 38: Evidence for test 11

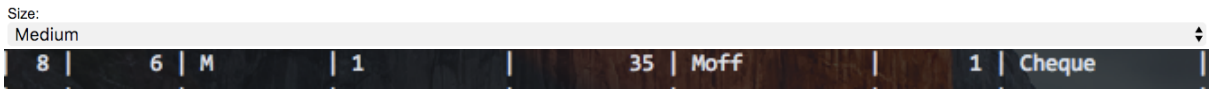


Figure 39: Evidence for test 12

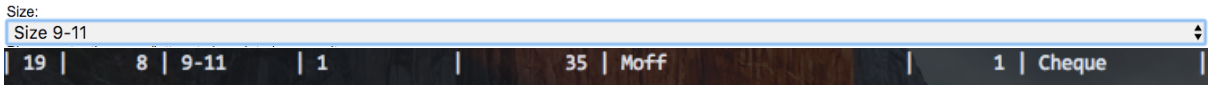


Figure 40: Evidence for test 13

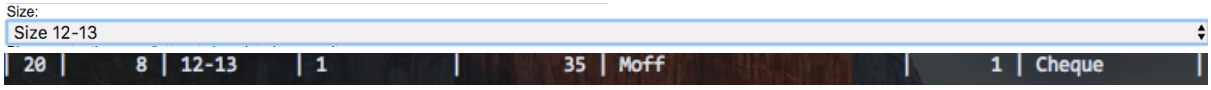


Figure 41: Evidence for test 14

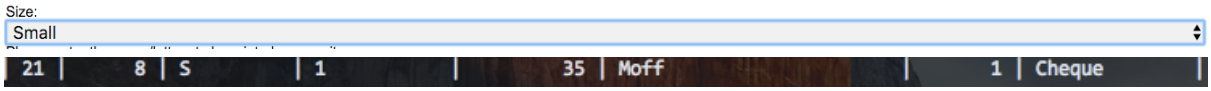


Figure 42: Evidence for test 15

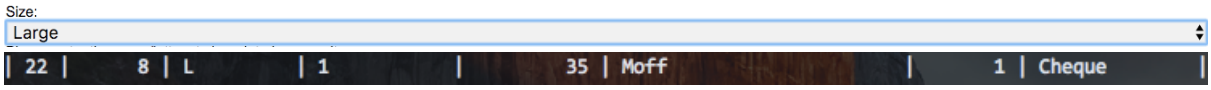


Figure 43: Evidence for test 16

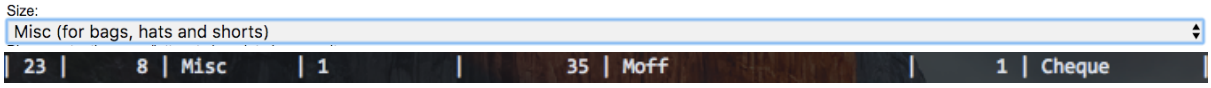


Figure 44: Evidence for test 17

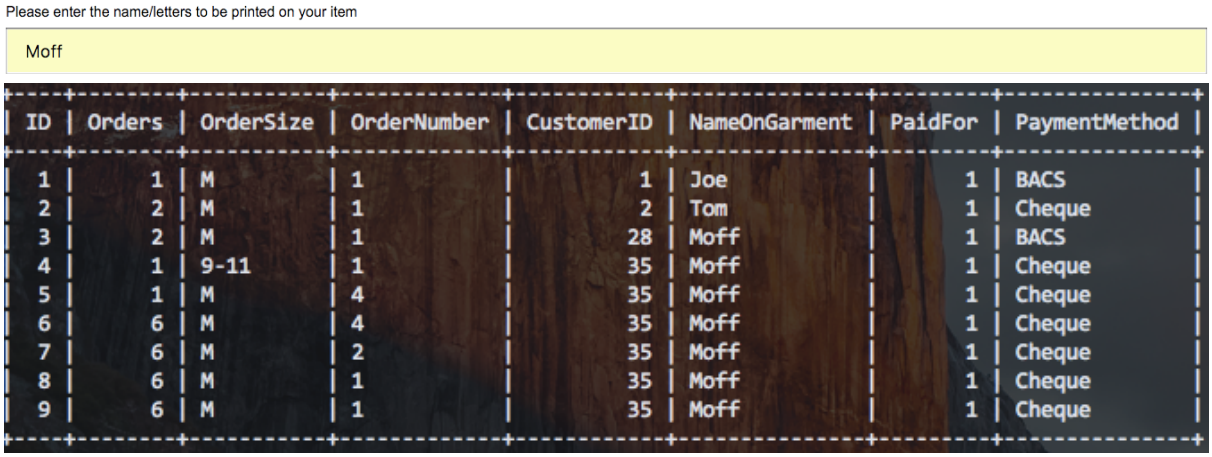
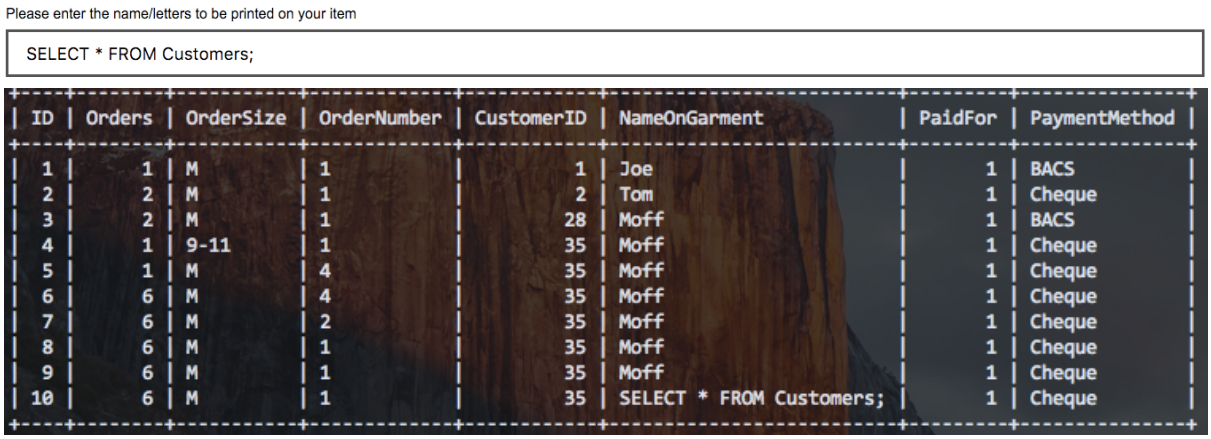
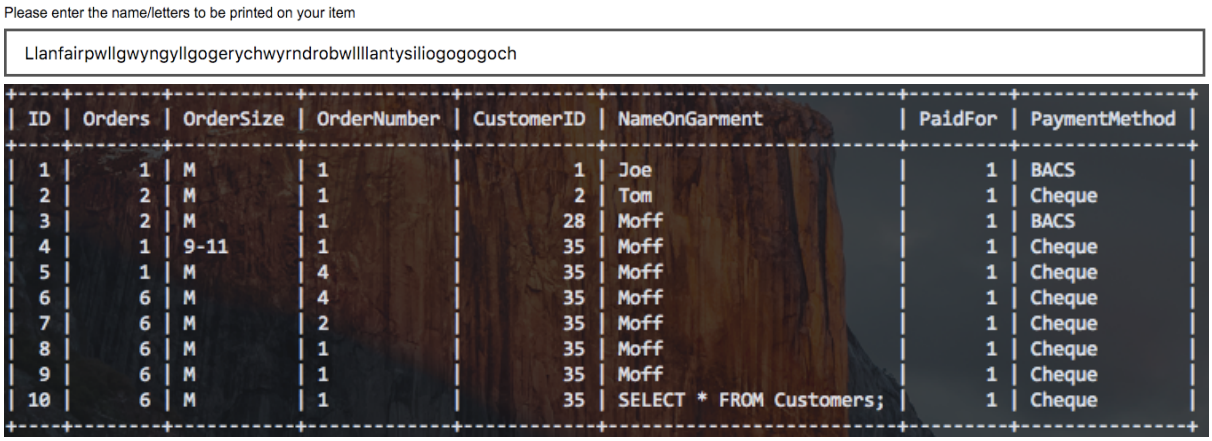


Figure 45: Evidence for test 18



ID	Orders	OrderSize	OrderNumber	CustomerID	NameOnGarment	PaidFor	PaymentMethod
1	1	M	1	1	Joe	1	BACS
2	2	M	1	2	Tom	1	Cheque
3	2	M	1	28	Moff	1	BACS
4	1	9-11	1	35	Moff	1	Cheque
5	1	M	4	35	Moff	1	Cheque
6	6	M	4	35	Moff	1	Cheque
7	6	M	2	35	Moff	1	Cheque
8	6	M	1	35	Moff	1	Cheque
9	6	M	1	35	Moff	1	Cheque
10	6	M	1	35	SELECT * FROM Customers;	1	Cheque

Figure 46: Evidence for test 19



ID	Orders	OrderSize	OrderNumber	CustomerID	NameOnGarment	PaidFor	PaymentMethod
1	1	M	1	1	Joe	1	BACS
2	2	M	1	2	Tom	1	Cheque
3	2	M	1	28	Moff	1	BACS
4	1	9-11	1	35	Moff	1	Cheque
5	1	M	4	35	Moff	1	Cheque
6	6	M	4	35	Moff	1	Cheque
7	6	M	2	35	Moff	1	Cheque
8	6	M	1	35	Moff	1	Cheque
9	6	M	1	35	Moff	1	Cheque
10	6	M	1	35	SELECT * FROM Customers;	1	Cheque

Figure 47: Evidence for test 20

24	8	Misc	1	35	Moff	1	Cheque

Figure 48: Evidence for test 21

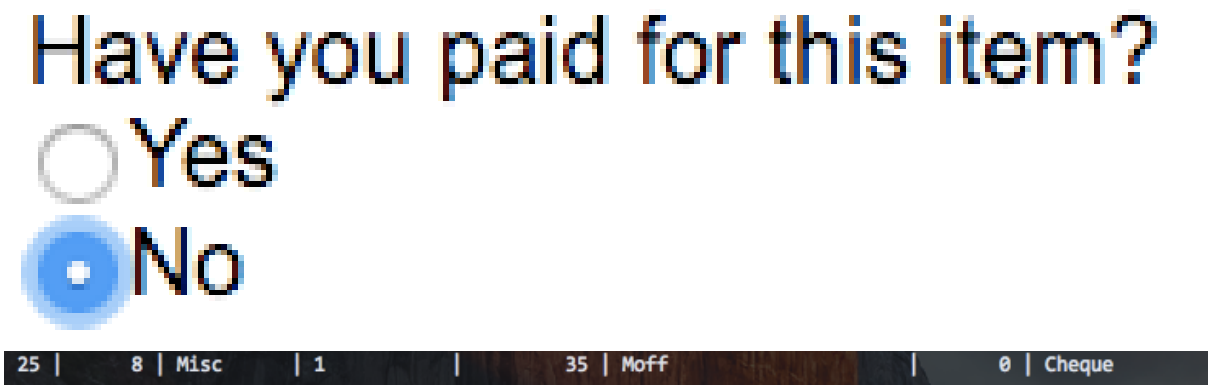


Figure 49: Evidence for test 22

c.1.3 Miscellaneous Web Testing Screenshots

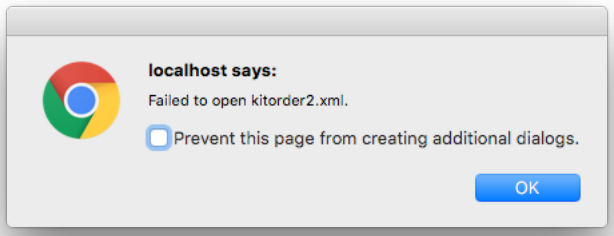


Figure 50: Evidence for test 1

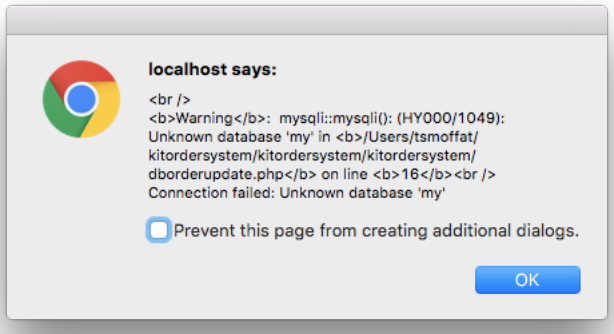


Figure 51: Evidence for test 3

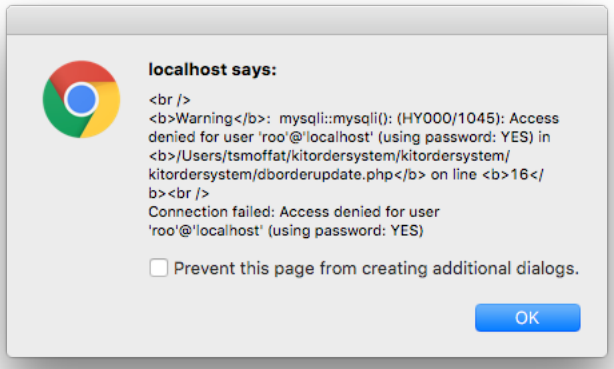


Figure 52: Evidence for test 4

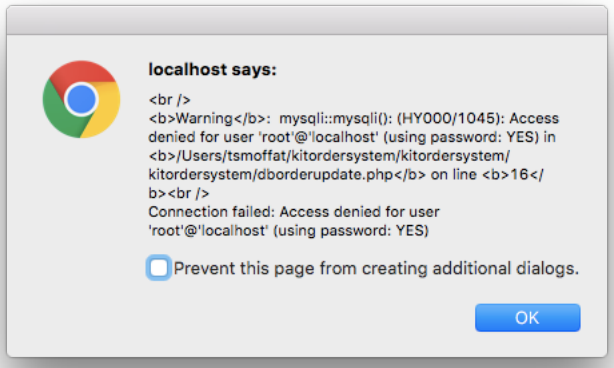


Figure 53: Evidence for test 5

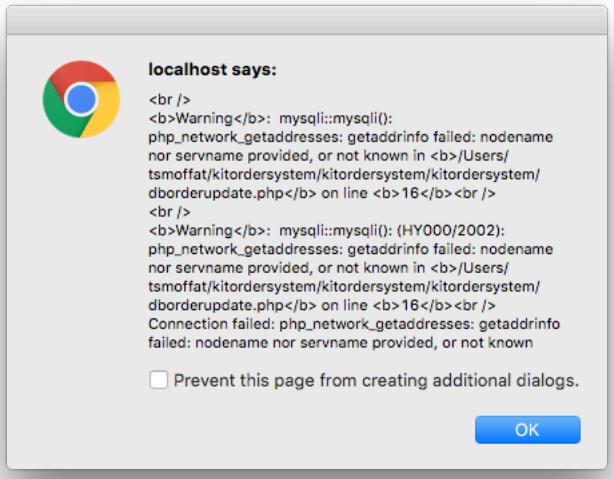


Figure 54: Evidence for test 7

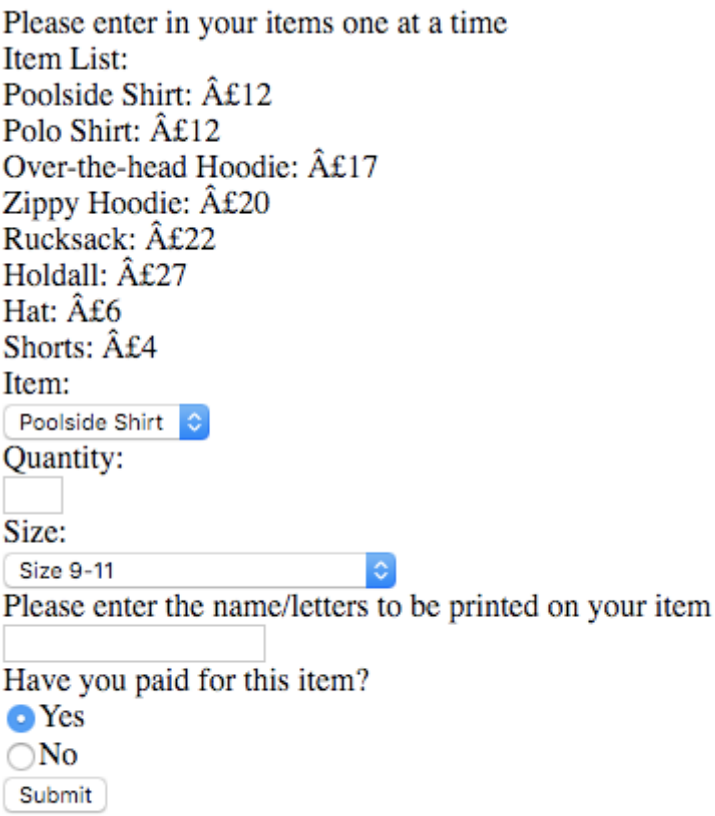


Figure 55: Evidence for test 8

c.2 Java Testing Screenshots

c.2.1 Java General Testing Screenshots

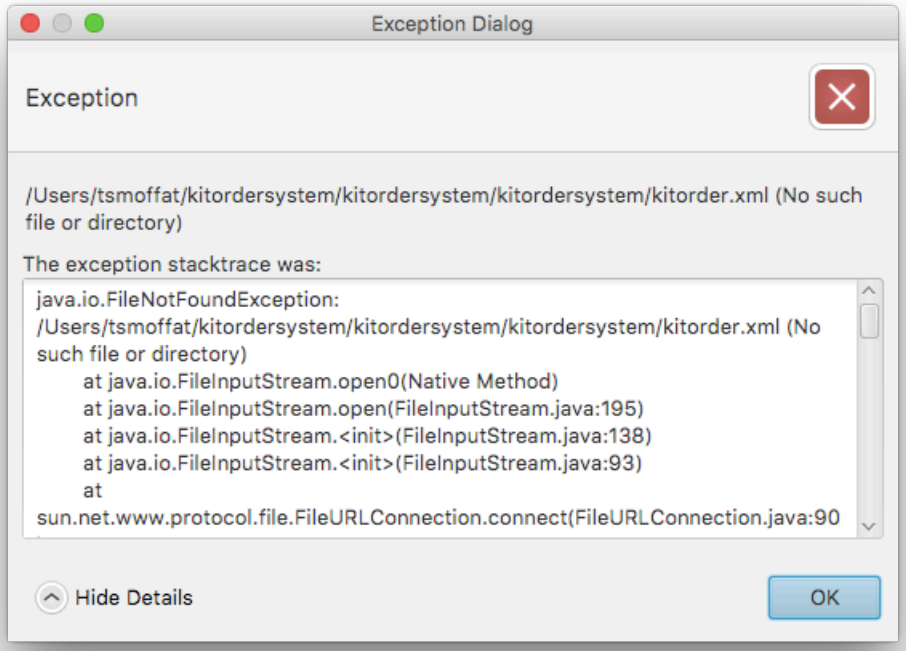
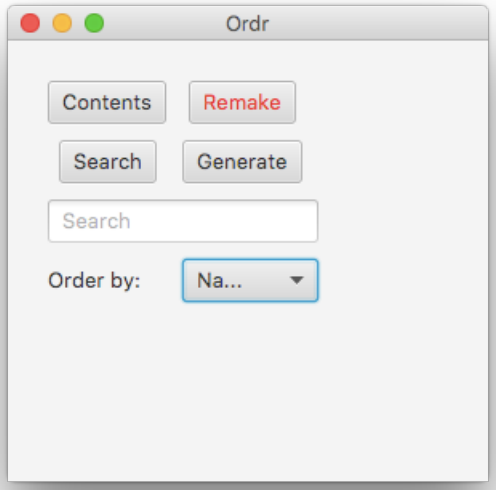
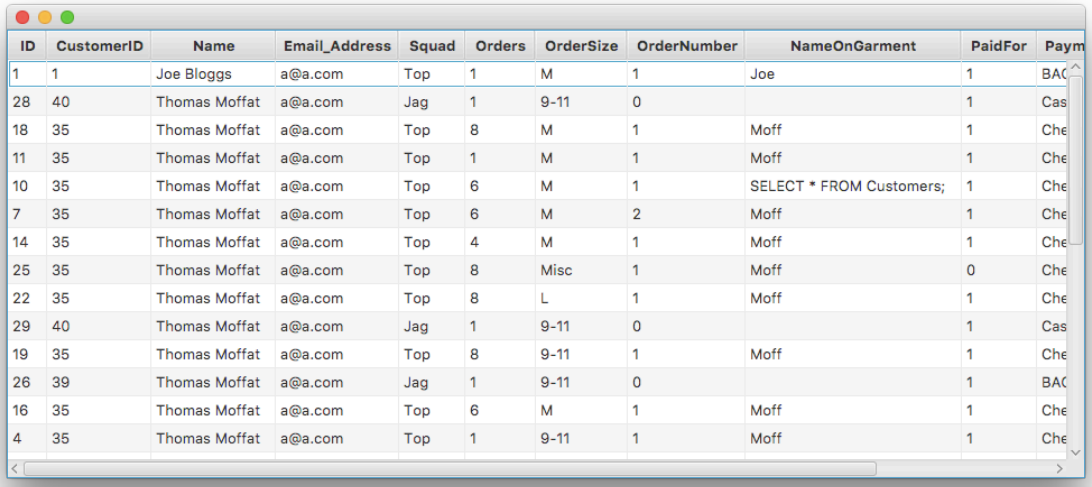


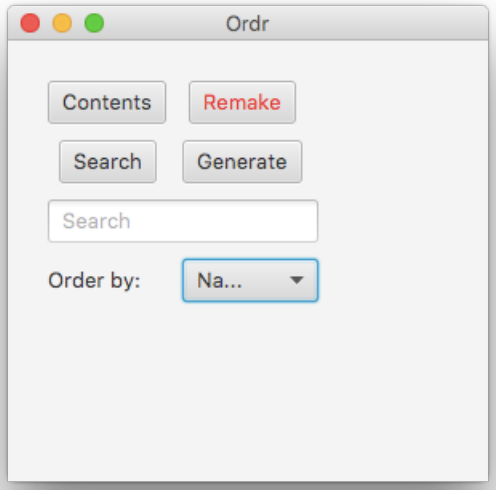
Figure 56: The updated error dialog, this gives exactly the same output as the following tests, it is just updated





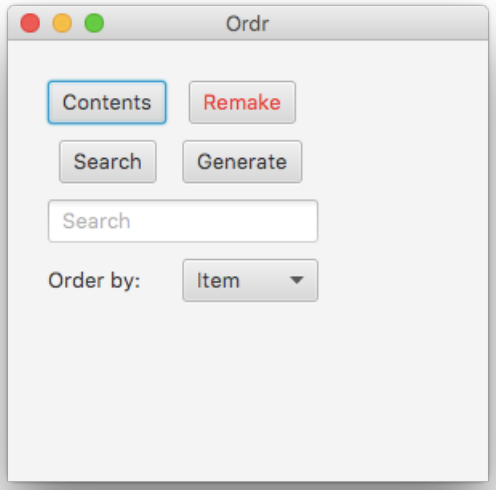
ID	CustomerID	Name	Email_Address	Squad	Orders	OrderSize	OrderNumber	NameOnGarment	PaidFor	Paym
1	1	Joe Bloggs	a@a.com	Top	1	M	1	Joe	1	BAC
28	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cas
18	35	Thomas Moffat	a@a.com	Top	8	M	1	Moff	1	Che
11	35	Thomas Moffat	a@a.com	Top	1	M	1	Moff	1	Che
10	35	Thomas Moffat	a@a.com	Top	6	M	1	SELECT * FROM Customers;	1	Che
7	35	Thomas Moffat	a@a.com	Top	6	M	2	Moff	1	Che
14	35	Thomas Moffat	a@a.com	Top	4	M	1	Moff	1	Che
25	35	Thomas Moffat	a@a.com	Top	8	Misc	1	Moff	0	Che
22	35	Thomas Moffat	a@a.com	Top	8	L	1	Moff	1	Che
29	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cas
19	35	Thomas Moffat	a@a.com	Top	8	9-11	1	Moff	1	Che
26	39	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	BAC
16	35	Thomas Moffat	a@a.com	Top	6	M	1	Moff	1	Che
4	35	Thomas Moffat	a@a.com	Top	1	9-11	1	Moff	1	Che

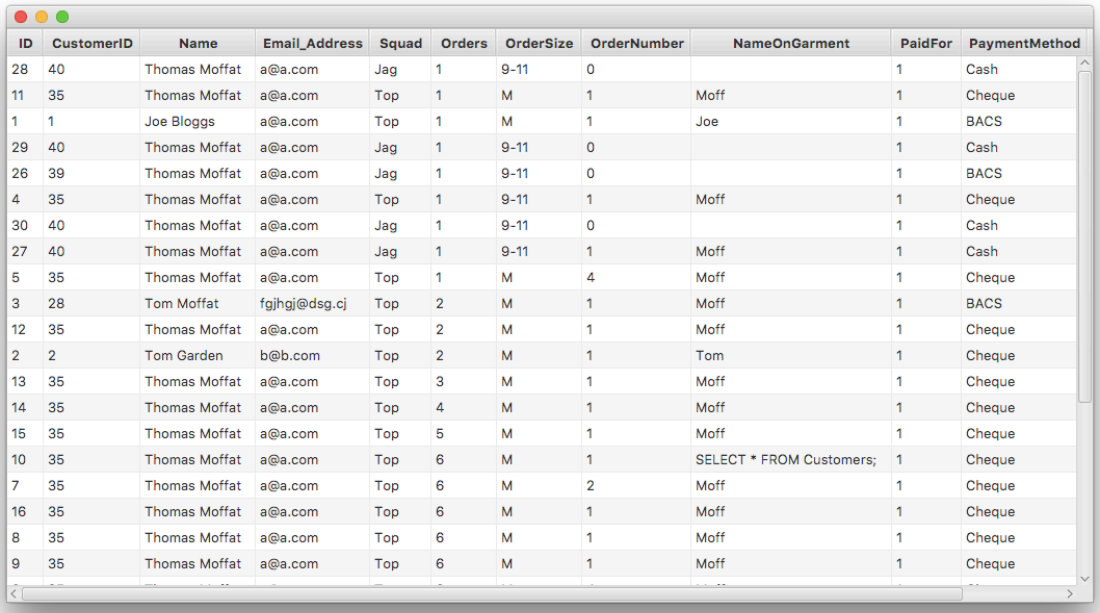
Figure 57: Evidence for test 1



ID	CustomerID	Name	Email_Address	Squad	Orders	OrderSize	OrderNumber	NameOnGarment	PaidFor	Paym
3	28	Tom Moffat	fgjhgj@dsg.cj	Top	2	M	1	Moff	1	BAC
2	2	Tom Garden	b@b.com	Top	2	M	1	Tom	1	Che
28	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cas
18	35	Thomas Moffat	a@a.com	Top	8	M	1	Moff	1	Che
11	35	Thomas Moffat	a@a.com	Top	1	M	1	Moff	1	Che
10	35	Thomas Moffat	a@a.com	Top	6	M	1	SELECT * FROM Customers;	1	Che
7	35	Thomas Moffat	a@a.com	Top	6	M	2	Moff	1	Che
14	35	Thomas Moffat	a@a.com	Top	4	M	1	Moff	1	Che
25	35	Thomas Moffat	a@a.com	Top	8	Misc	1	Moff	0	Che
22	35	Thomas Moffat	a@a.com	Top	8	L	1	Moff	1	Che
29	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cas
19	35	Thomas Moffat	a@a.com	Top	8	9-11	1	Moff	1	Che
26	39	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	BAC
16	35	Thomas Moffat	a@a.com	Top	6	M	1	Moff	1	Che

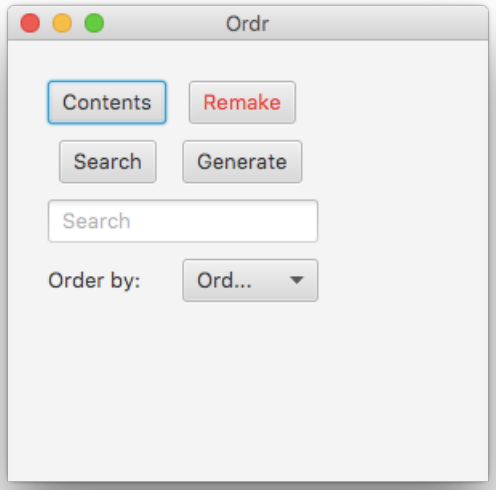
Figure 58: Evidence for test 2

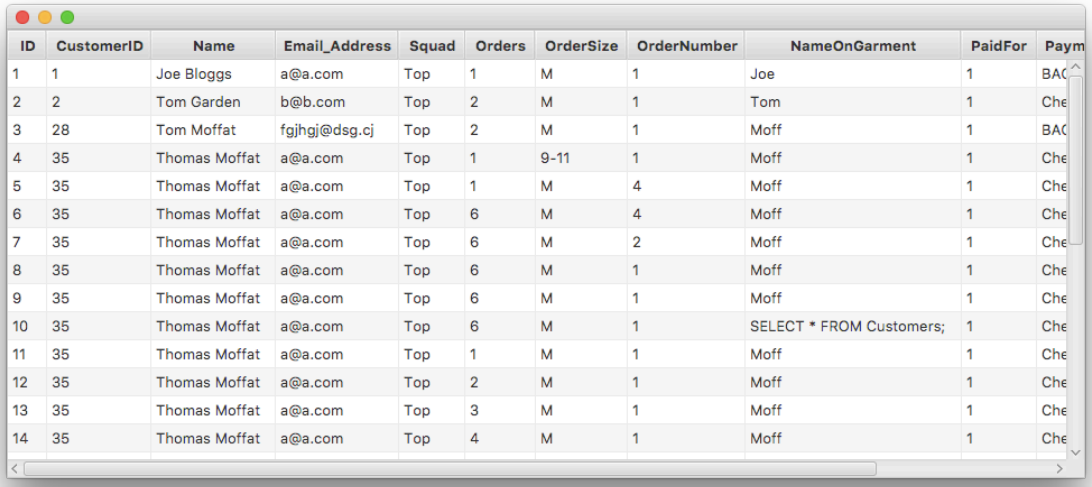




ID	CustomerID	Name	Email_Address	Squad	Orders	OrderSize	OrderNumber	NameOnGarment	PaidFor	PaymentMethod
28	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cash
11	35	Thomas Moffat	a@a.com	Top	1	M	1	Moff	1	Cheque
1	1	Joe Bloggs	a@a.com	Top	1	M	1	Joe	1	BACS
29	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cash
26	39	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	BACS
4	35	Thomas Moffat	a@a.com	Top	1	9-11	1	Moff	1	Cheque
30	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cash
27	40	Thomas Moffat	a@a.com	Jag	1	9-11	1	Moff	1	Cash
5	35	Thomas Moffat	a@a.com	Top	1	M	4	Moff	1	Cheque
3	28	Tom Moffat	fgjhgj@dsg.cj	Top	2	M	1	Moff	1	BACS
12	35	Thomas Moffat	a@a.com	Top	2	M	1	Moff	1	Cheque
2	2	Tom Garden	b@b.com	Top	2	M	1	Tom	1	Cheque
13	35	Thomas Moffat	a@a.com	Top	3	M	1	Moff	1	Cheque
14	35	Thomas Moffat	a@a.com	Top	4	M	1	Moff	1	Cheque
15	35	Thomas Moffat	a@a.com	Top	5	M	1	Moff	1	Cheque
10	35	Thomas Moffat	a@a.com	Top	6	M	1	SELECT * FROM Customers;	1	Cheque
7	35	Thomas Moffat	a@a.com	Top	6	M	2	Moff	1	Cheque
16	35	Thomas Moffat	a@a.com	Top	6	M	1	Moff	1	Cheque
8	35	Thomas Moffat	a@a.com	Top	6	M	1	Moff	1	Cheque
9	35	Thomas Moffat	a@a.com	Top	6	M	1	Moff	1	Cheque

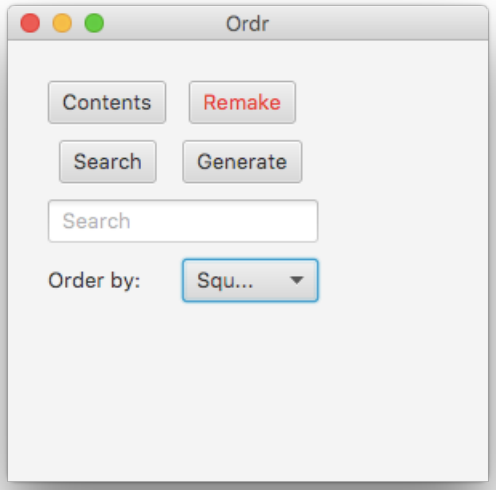
Figure 59: Evidence for test 3

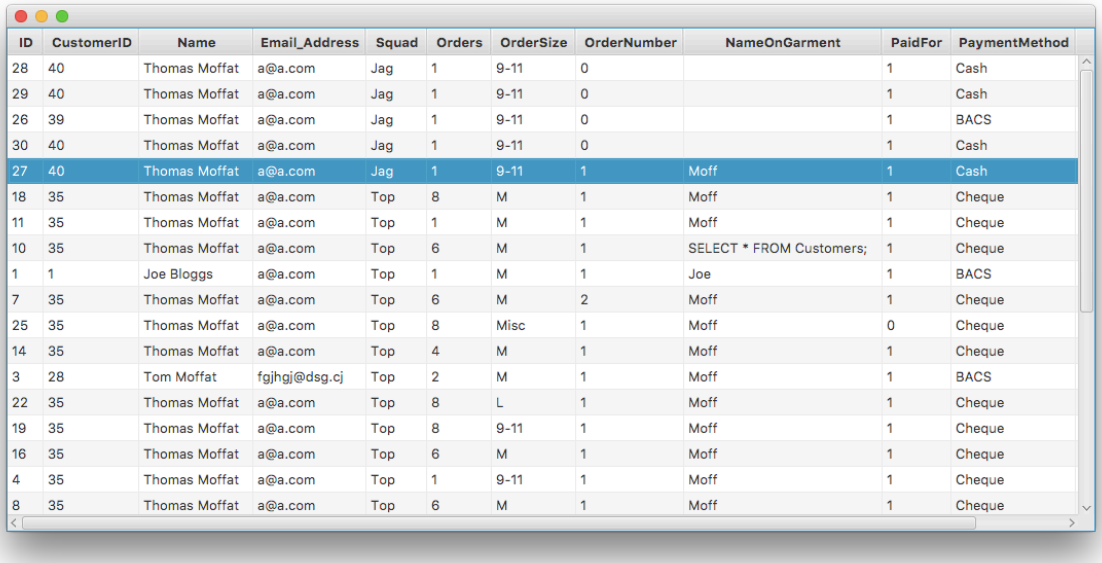




ID	CustomerID	Name	Email_Address	Squad	Orders	OrderSize	OrderNumber	NameOnGarment	PaidFor	Paym
1	1	Joe Bloggs	a@a.com	Top	1	M	1	Joe	1	BAC
2	2	Tom Garden	b@b.com	Top	2	M	1	Tom	1	Che
3	28	Tom Moffat	fgjhgj@dsg.cj	Top	2	M	1	Moff	1	BAC
4	35	Thomas Moffat	a@a.com	Top	1	9-11	1	Moff	1	Che
5	35	Thomas Moffat	a@a.com	Top	1	M	4	Moff	1	Che
6	35	Thomas Moffat	a@a.com	Top	6	M	4	Moff	1	Che
7	35	Thomas Moffat	a@a.com	Top	6	M	2	Moff	1	Che
8	35	Thomas Moffat	a@a.com	Top	6	M	1	Moff	1	Che
9	35	Thomas Moffat	a@a.com	Top	6	M	1	Moff	1	Che
10	35	Thomas Moffat	a@a.com	Top	6	M	1	SELECT * FROM Customers;	1	Che
11	35	Thomas Moffat	a@a.com	Top	1	M	1	Moff	1	Che
12	35	Thomas Moffat	a@a.com	Top	2	M	1	Moff	1	Che
13	35	Thomas Moffat	a@a.com	Top	3	M	1	Moff	1	Che
14	35	Thomas Moffat	a@a.com	Top	4	M	1	Moff	1	Che

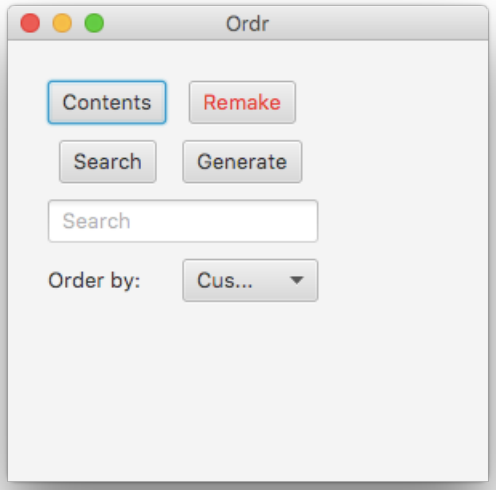
Figure 60: Evidence for test 4

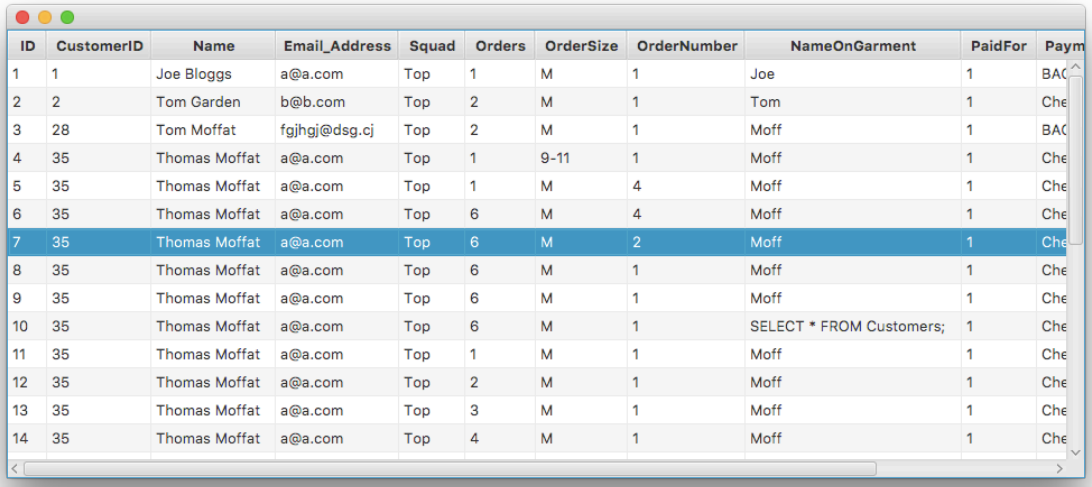




ID	CustomerID	Name	Email_Address	Squad	Orders	OrderSize	OrderNumber	NameOnGarment	PaidFor	PaymentMethod
28	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cash
29	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cash
26	39	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	BACS
30	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cash
27	40	Thomas Moffat	a@a.com	Jag	1	9-11	1	Moff	1	Cash
18	35	Thomas Moffat	a@a.com	Top	8	M	1	Moff	1	Cheque
11	35	Thomas Moffat	a@a.com	Top	1	M	1	Moff	1	Cheque
10	35	Thomas Moffat	a@a.com	Top	6	M	1	SELECT * FROM Customers;	1	Cheque
1	1	Joe Bloggs	a@a.com	Top	1	M	1	Joe	1	BACS
7	35	Thomas Moffat	a@a.com	Top	6	M	2	Moff	1	Cheque
25	35	Thomas Moffat	a@a.com	Top	8	Misc	1	Moff	0	Cheque
14	35	Thomas Moffat	a@a.com	Top	4	M	1	Moff	1	Cheque
3	28	Tom Moffat	fgjhgj@dsg.cj	Top	2	M	1	Moff	1	BACS
22	35	Thomas Moffat	a@a.com	Top	8	L	1	Moff	1	Cheque
19	35	Thomas Moffat	a@a.com	Top	8	9-11	1	Moff	1	Cheque
16	35	Thomas Moffat	a@a.com	Top	6	M	1	Moff	1	Cheque
4	35	Thomas Moffat	a@a.com	Top	1	9-11	1	Moff	1	Cheque
8	35	Thomas Moffat	a@a.com	Top	6	M	1	Moff	1	Cheque

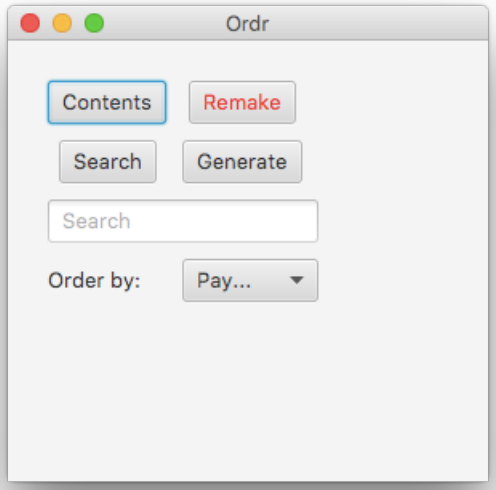
Figure 61: Evidence for test 5

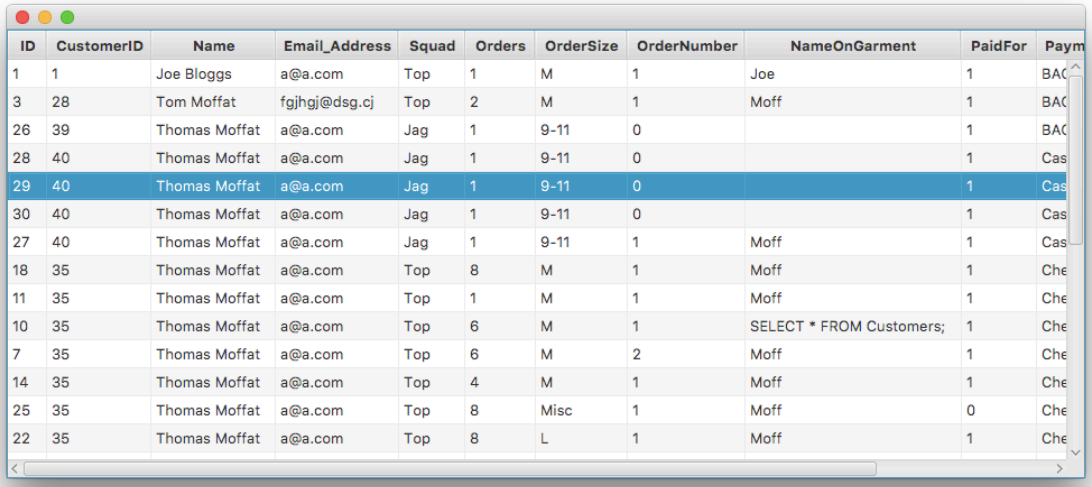




ID	CustomerID	Name	Email_Address	Squad	Orders	OrderSize	OrderNumber	NameOnGarment	PaidFor	Paym
1	1	Joe Bloggs	a@a.com	Top	1	M	1	Joe	1	BAC
2	2	Tom Garden	b@b.com	Top	2	M	1	Tom	1	Che
3	28	Tom Moffat	fgjhgj@dsg.cj	Top	2	M	1	Moff	1	BAC
4	35	Thomas Moffat	a@a.com	Top	1	9-11	1	Moff	1	Che
5	35	Thomas Moffat	a@a.com	Top	1	M	4	Moff	1	Che
6	35	Thomas Moffat	a@a.com	Top	6	M	4	Moff	1	Che
7	35	Thomas Moffat	a@a.com	Top	6	M	2	Moff	1	Che
8	35	Thomas Moffat	a@a.com	Top	6	M	1	Moff	1	Che
9	35	Thomas Moffat	a@a.com	Top	6	M	1	Moff	1	Che
10	35	Thomas Moffat	a@a.com	Top	6	M	1	SELECT * FROM Customers;	1	Che
11	35	Thomas Moffat	a@a.com	Top	1	M	1	Moff	1	Che
12	35	Thomas Moffat	a@a.com	Top	2	M	1	Moff	1	Che
13	35	Thomas Moffat	a@a.com	Top	3	M	1	Moff	1	Che
14	35	Thomas Moffat	a@a.com	Top	4	M	1	Moff	1	Che

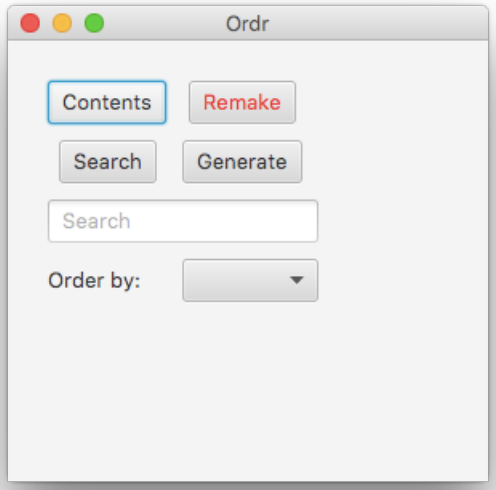
Figure 62: Evidence for test 6





ID	CustomerID	Name	Email_Address	Squad	Orders	OrderSize	OrderNumber	NameOnGarment	PaidFor	Paym
1	1	Joe Bloggs	a@a.com	Top	1	M	1	Joe	1	BAC
3	28	Tom Moffat	fgjhgj@dsg.cj	Top	2	M	1	Moff	1	BAC
26	39	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	BAC
28	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cas
29	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cas
30	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cas
27	40	Thomas Moffat	a@a.com	Jag	1	9-11	1	Moff	1	Cas
18	35	Thomas Moffat	a@a.com	Top	8	M	1	Moff	1	Che
11	35	Thomas Moffat	a@a.com	Top	1	M	1	Moff	1	Che
10	35	Thomas Moffat	a@a.com	Top	6	M	1	SELECT * FROM Customers;	1	Che
7	35	Thomas Moffat	a@a.com	Top	6	M	2	Moff	1	Che
14	35	Thomas Moffat	a@a.com	Top	4	M	1	Moff	1	Che
25	35	Thomas Moffat	a@a.com	Top	8	Misc	1	Moff	0	Che
22	35	Thomas Moffat	a@a.com	Top	8	L	1	Moff	1	Che

Figure 63: Evidence for test 7




```
Exception in thread "JavaFX Application Thread" java.lang.NullPointerException
    at kitordersystem.MainMenu$2.handle(MainMenu.java:109)
    at kitordersystem.MainMenu$2.handle(MainMenu.java:98)
    at com.sun.javafx.event.CompositeEventHandler.dispatchBubblingEvent(CompositeEventHandler.java:86)
    at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:238)
    at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:191)
    at com.sun.javafx.event.CompositeEventDispatcher.dispatchBubblingEvent(CompositeEventDispatcher.java:59)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:58)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
    at com.sun.javafx.event.EventUtil.fireEventImpl(EventUtil.java:74)
    at com.sun.javafx.event.EventUtil.fireEvent(EventUtil.java:49)
    at javafx.event.Event.fireEvent(Event.java:198)
    at javafx.scene.Node.fireEvent(Node.java:8411)
    at javafx.scene.control.Button.fire(Button.java:185)
```

Figure 64: Evidence for test 8

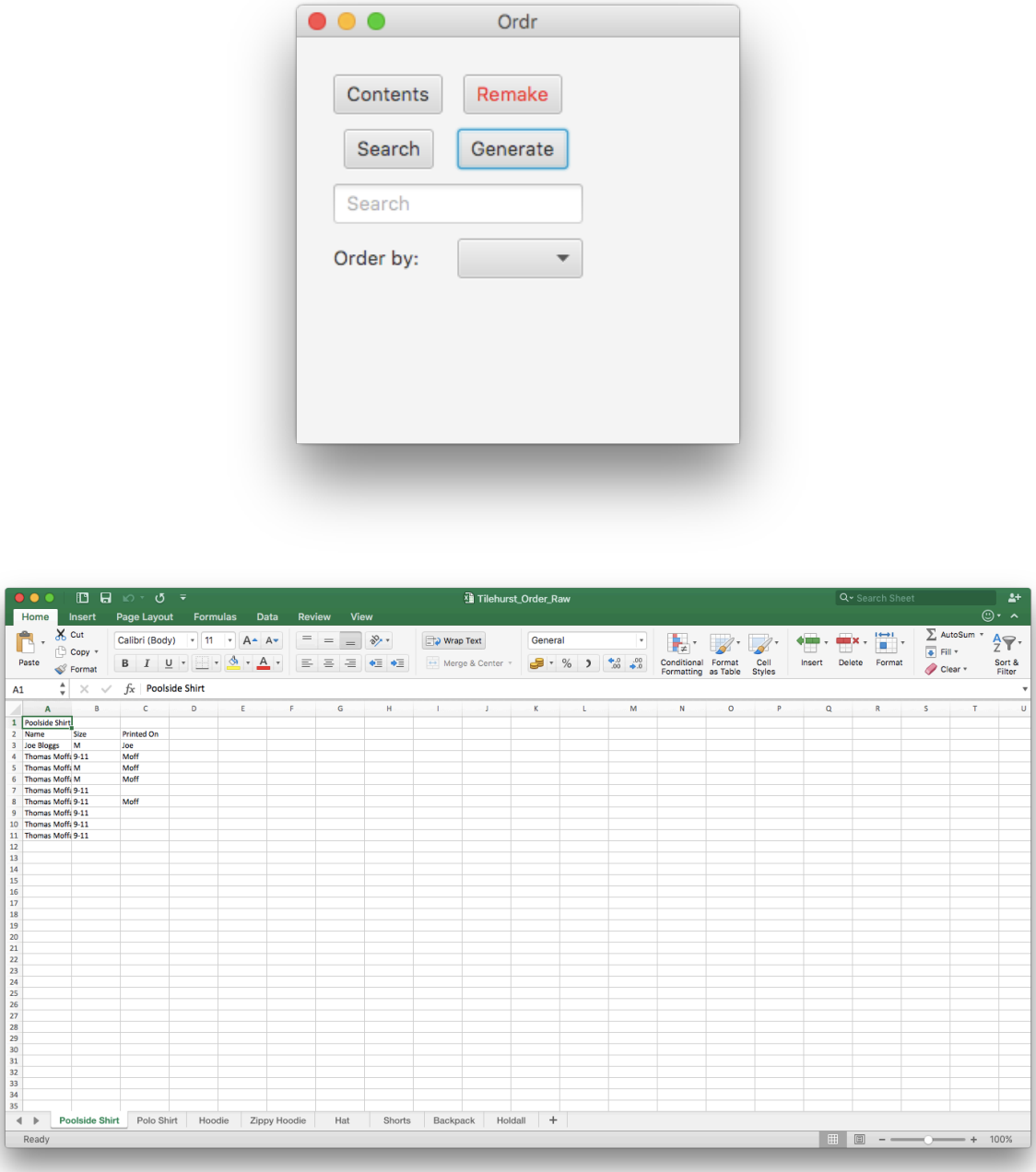
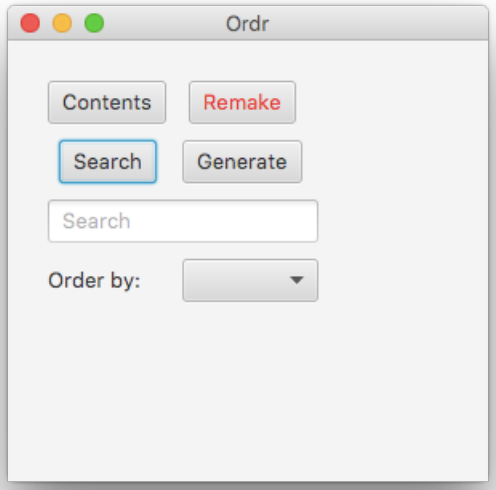
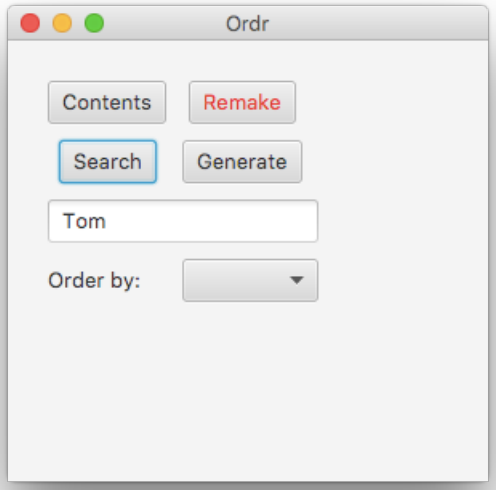


Figure 65: Evidence for test 9



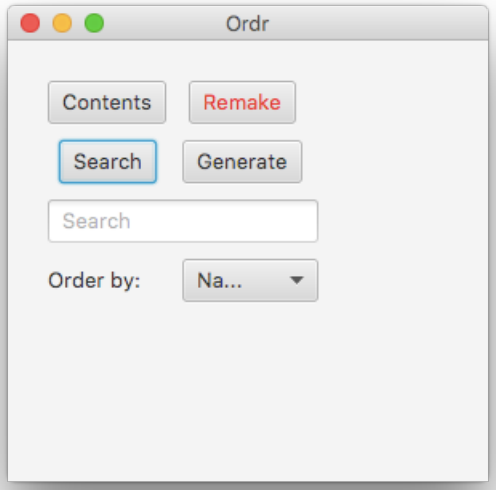

```
Exception in thread "JavaFX Application Thread" java.lang.NullPointerException
    at kitordersystem.MainMenu$4.handle(MainMenu.java:145)
    at kitordersystem.MainMenu$4.handle(MainMenu.java:141)
    at com.sun.javafx.event.CompositeEventHandler.dispatchBubblingEvent(CompositeEventHandler.java:86)
    at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:238)
    at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:191)
    at com.sun.javafx.event.CompositeEventDispatcher.dispatchBubblingEvent(CompositeEventDispatcher.java:59)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:58)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
    at com.sun.javafx.event.EventUtil.fireEventImpl(EventUtil.java:74)
    at com.sun.javafx.event.EventUtil.fireEvent(EventUtil.java:49)
```

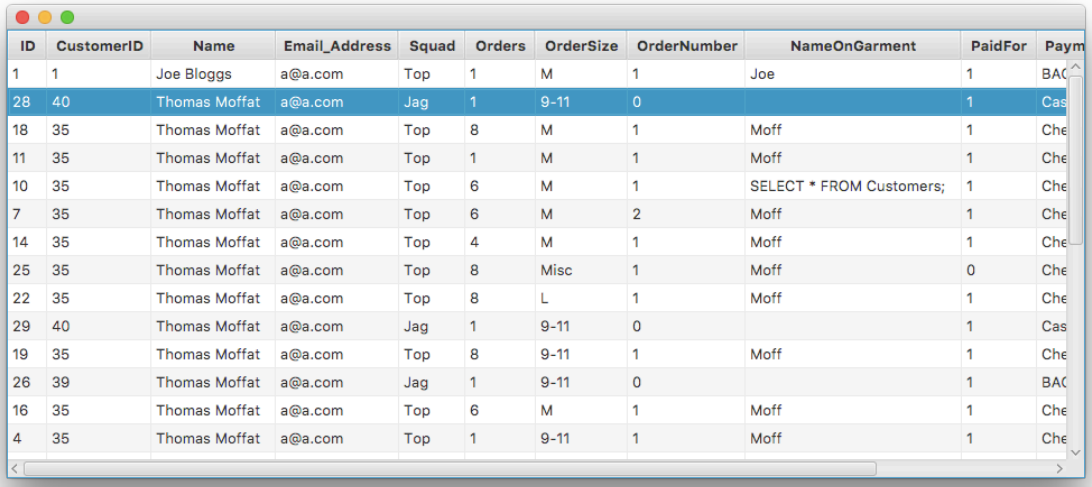
Figure 66: Evidence for test 10




```
Exception in thread "JavaFX Application Thread" java.lang.NullPointerException
    at kitordersystem.MainMenu$4.handle(MainMenu.java:145)
    at kitordersystem.MainMenu$4.handle(MainMenu.java:141)
    at com.sun.javafx.event.CompositeEventHandler.dispatchBubblingEvent(CompositeEventHandler.java:86)
    at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:238)
    at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:191)
    at com.sun.javafx.event.CompositeEventDispatcher.dispatchBubblingEvent(CompositeEventDispatcher.java:59)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:58)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
    at com.sun.javafx.event.EventUtil.fireEventImpl(EventUtil.java:74)
    at com.sun.javafx.event.EventUtil.fireEvent(EventUtil.java:49)
    at javafx.event.Event.fireEvent(Event.java:198)
    at javafx.scene.Node.fireEvent(Node.java:8411)
    at javafx.scene.control.Button.fire(Button.java:185)
    at com.sun.javafx.scene.control.behavior.ButtonBehavior.mouseReleased(ButtonBehavior.java:182)
    at com.sun.javafx.scene.control.skin.BehaviorSkinBase$1.handle(BehaviorSkinBase.java:96)
    at com.sun.javafx.scene.control.skin.BehaviorSkinBase$1.handle(BehaviorSkinBase.java:89)
    at com.sun.javafx.event.CompositeEventHandler$NormalEventHandlerRecord.handleBubblingEvent(CompositeEventHandler.java:218)
    at com.sun.javafx.event.CompositeEventHandler.dispatchBubblingEvent(CompositeEventHandler.java:80)
    at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:238)
    at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:191)
    at com.sun.javafx.event.CompositeEventDispatcher.dispatchBubblingEvent(CompositeEventDispatcher.java:59)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:58)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
    at com.sun.javafx.event.EventUtil.fireEventImpl(EventUtil.java:74)
    at com.sun.javafx.event.EventUtil.fireEvent(EventUtil.java:54)
    at javafx.event.Event.fireEvent(Event.java:198)
    at javafx.scene.Scene$MouseHandler.process(Scene.java:3757)
    at javafx.scene.Scene$MouseHandler.access$1500(Scene.java:3485)
    at javafx.scene.Scene.impl_processMouseEvent(Scene.java:1762)
    at javafx.scene.Scene$ScenePeerListener.mouseEvent(Scene.java:2494)
    at com.sun.javafx.tk.quantum.GlassViewEventHandler$MouseEventNotification.run(GlassViewEventHandler.java:352)
    at com.sun.javafx.tk.quantum.GlassViewEventHandler$MouseEventNotification.run(GlassViewEventHandler.java:275) <1 internal calls>
    at com.sun.javafx.tk.quantum.GlassViewEventHandler.lambda$handleMouseEvent$354(GlassViewEventHandler.java:388)
    at com.sun.javafx.tk.quantum.QuantumToolkit.runWithoutRenderLock(QuantumToolkit.java:389)
    at com.sun.javafx.tk.quantum.GlassViewEventHandler.handleMouseEvent(GlassViewEventHandler.java:387)
    at com.sun.glass.ui.View.handleMouseEvent(View.java:555)
    at com.sun.glass.ui.View.notifyMouse(View.java:937)
```

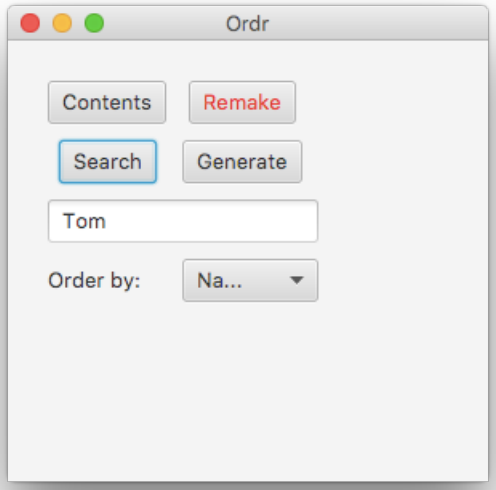
Figure 67: Evidence for test 11

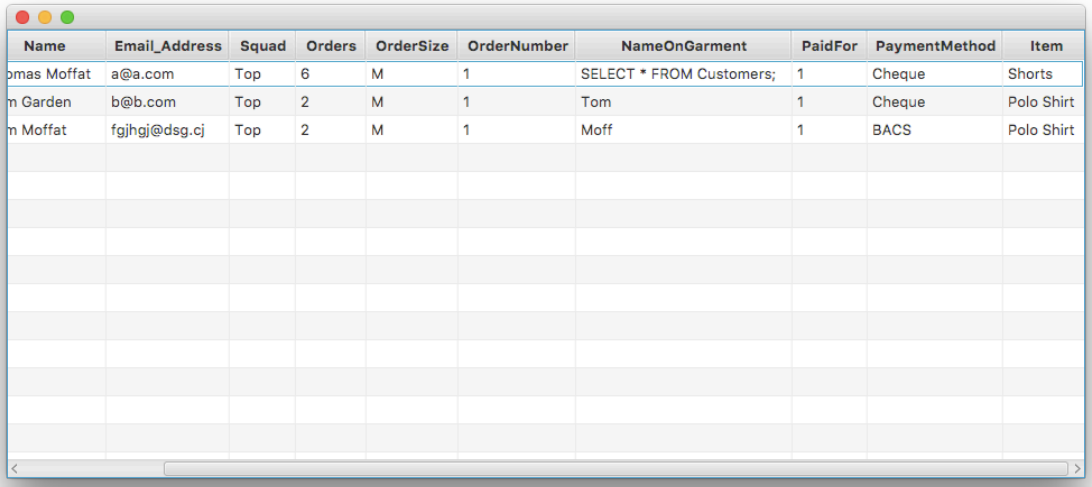




ID	CustomerID	Name	Email_Address	Squad	Orders	OrderSize	OrderNumber	NameOnGarment	PaidFor	Paym
1	1	Joe Bloggs	a@a.com	Top	1	M	1	Joe	1	BAC
28	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cas
18	35	Thomas Moffat	a@a.com	Top	8	M	1	Moff	1	Che
11	35	Thomas Moffat	a@a.com	Top	1	M	1	Moff	1	Che
10	35	Thomas Moffat	a@a.com	Top	6	M	1	SELECT * FROM Customers;	1	Che
7	35	Thomas Moffat	a@a.com	Top	6	M	2	Moff	1	Che
14	35	Thomas Moffat	a@a.com	Top	4	M	1	Moff	1	Che
25	35	Thomas Moffat	a@a.com	Top	8	Misc	1	Moff	0	Che
22	35	Thomas Moffat	a@a.com	Top	8	L	1	Moff	1	Che
29	40	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	Cas
19	35	Thomas Moffat	a@a.com	Top	8	9-11	1	Moff	1	Che
26	39	Thomas Moffat	a@a.com	Jag	1	9-11	0		1	BAC
16	35	Thomas Moffat	a@a.com	Top	6	M	1	Moff	1	Che
4	35	Thomas Moffat	a@a.com	Top	1	9-11	1	Moff	1	Che

Figure 68: Evidence for test 12





Name	Email_Address	Squad	Orders	OrderSize	OrderNumber	NameOnGarment	PaidFor	PaymentMethod	Item
omas Moffat	a@a.com	Top	6	M	1	SELECT * FROM Customers;	1	Cheque	Shorts
n Garden	b@b.com	Top	2	M	1	Tom	1	Cheque	Polo Shirt
n Moffat	fgjhgj@dsg.cj	Top	2	M	1	Moff	1	BACS	Polo Shirt

Figure 69: Evidence for test 13

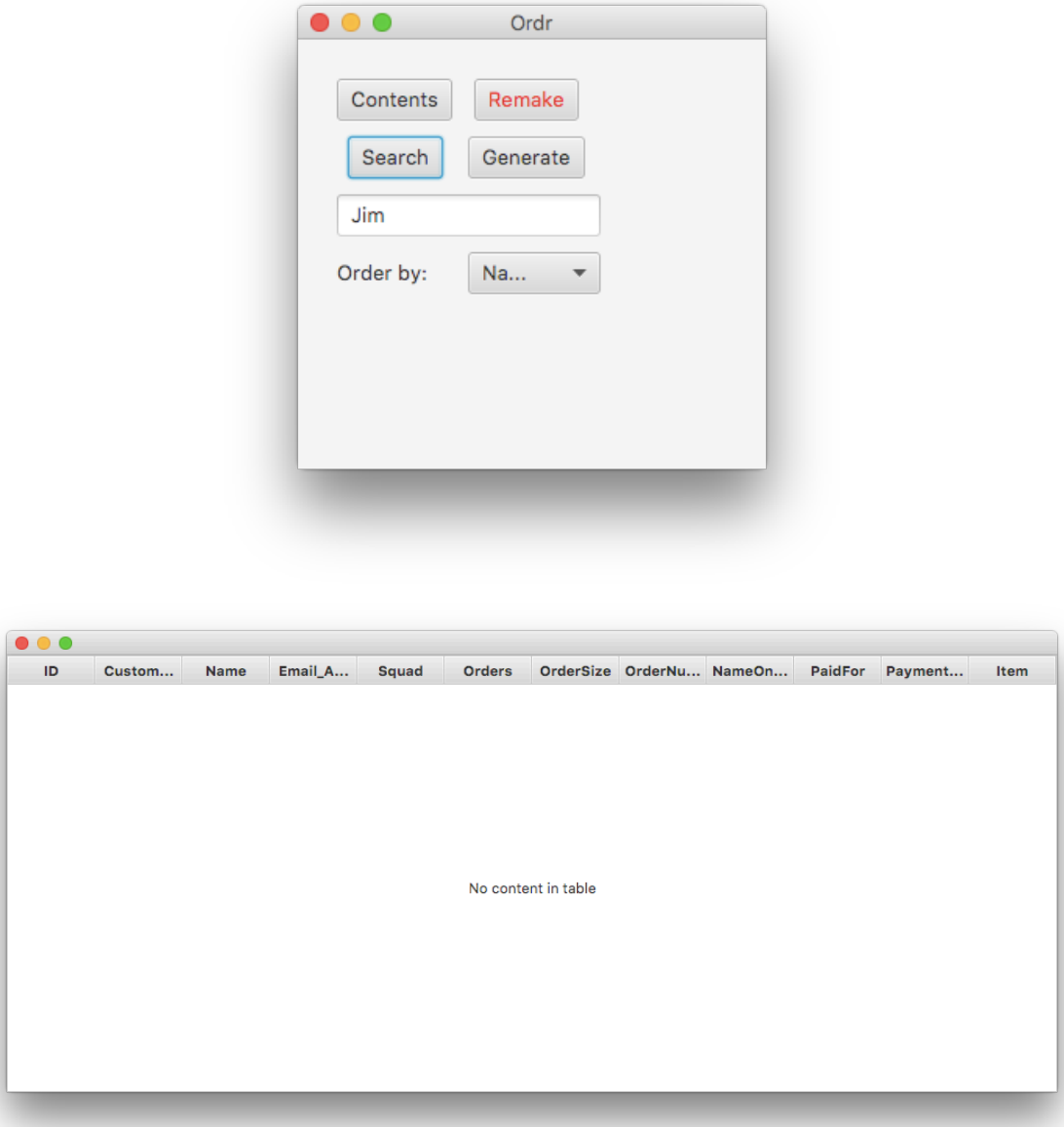


Figure 70: Evidence for test 14

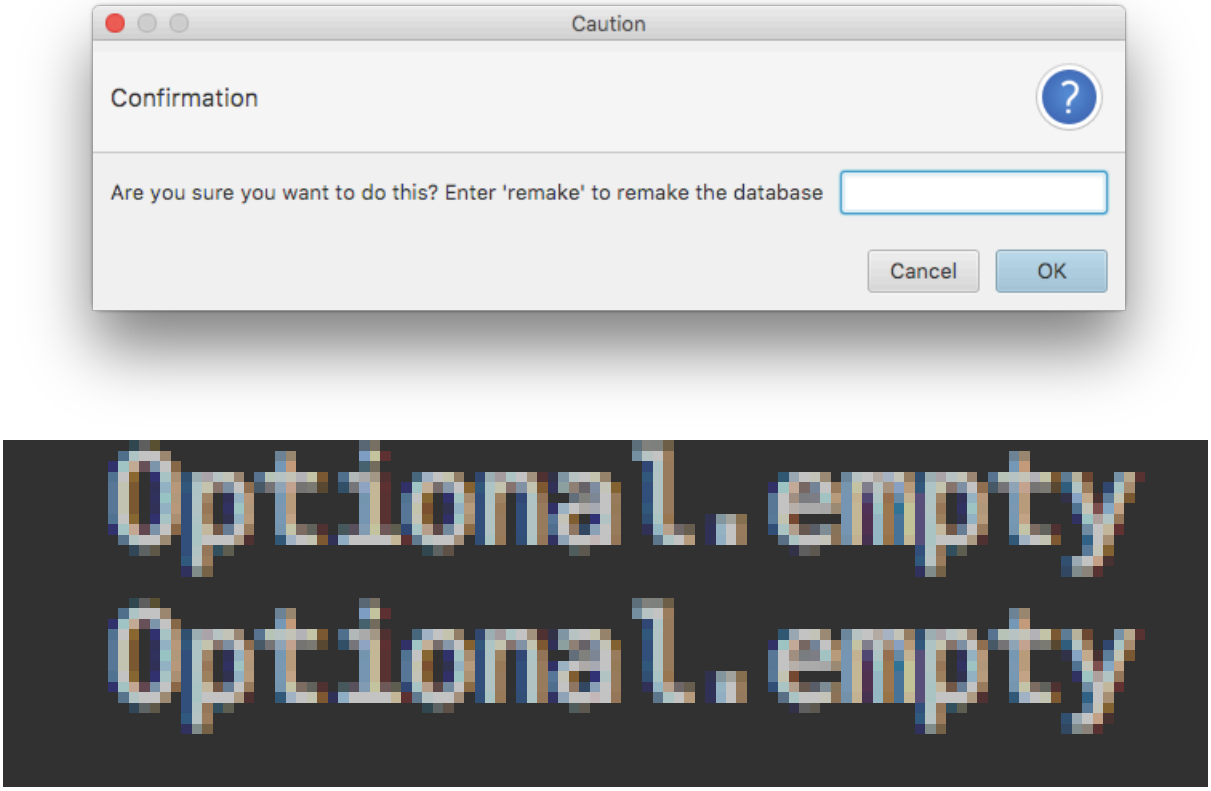


Figure 71: Evidence for test 15

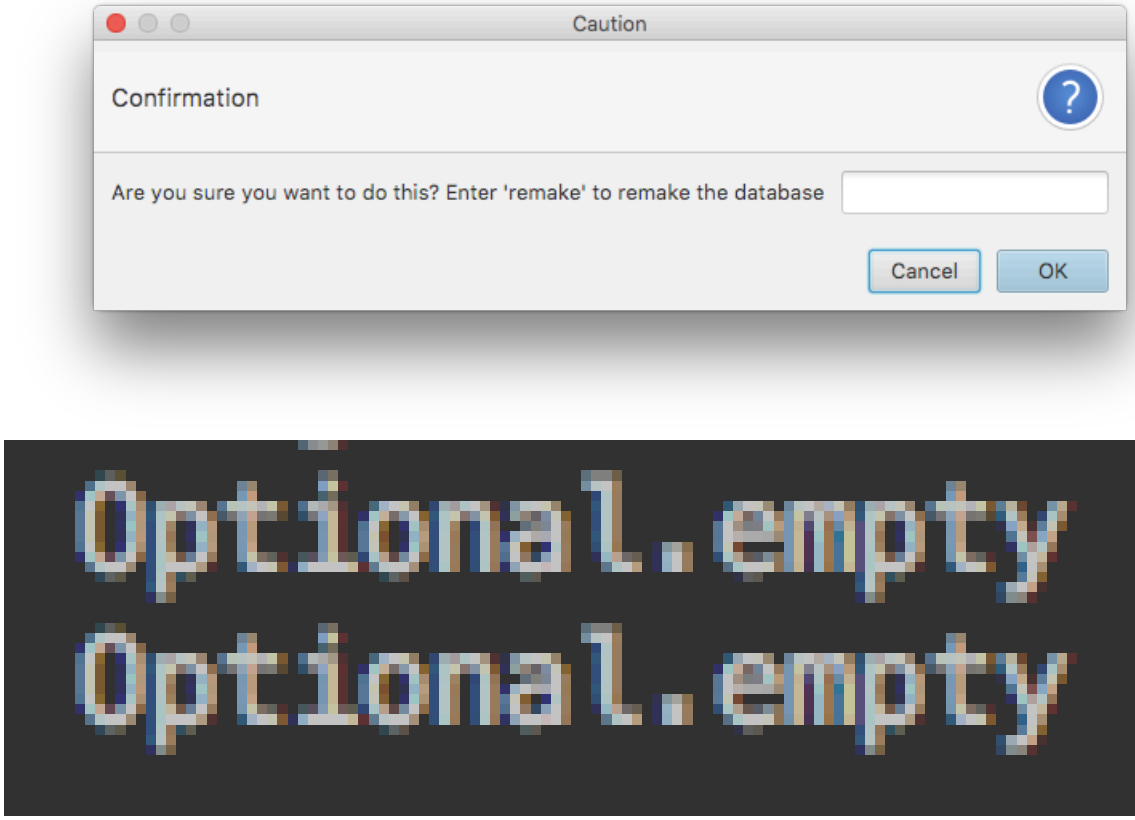


Figure 72: Evidence for test 16

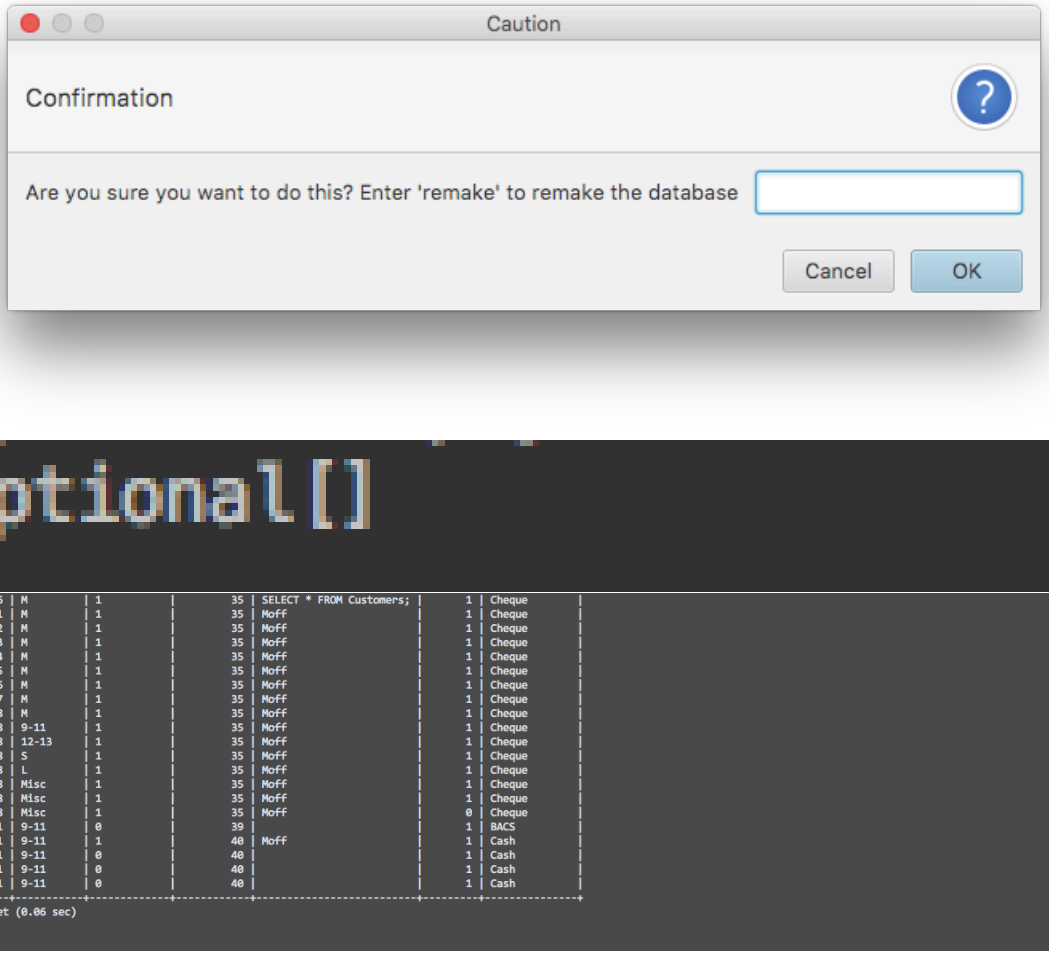
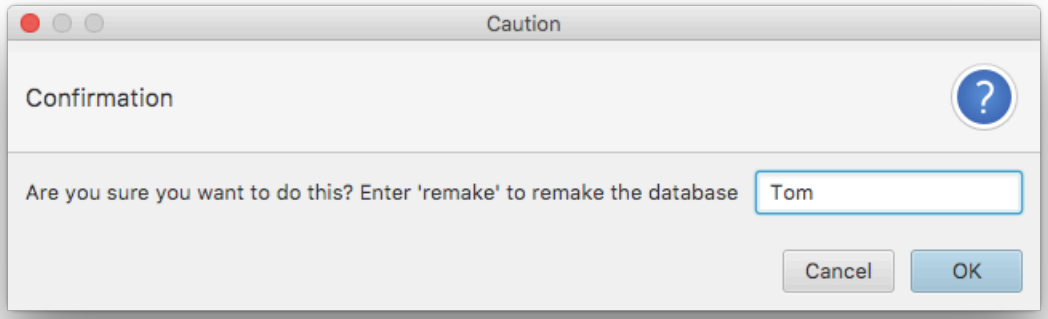


Figure 73: Evidence for test 17



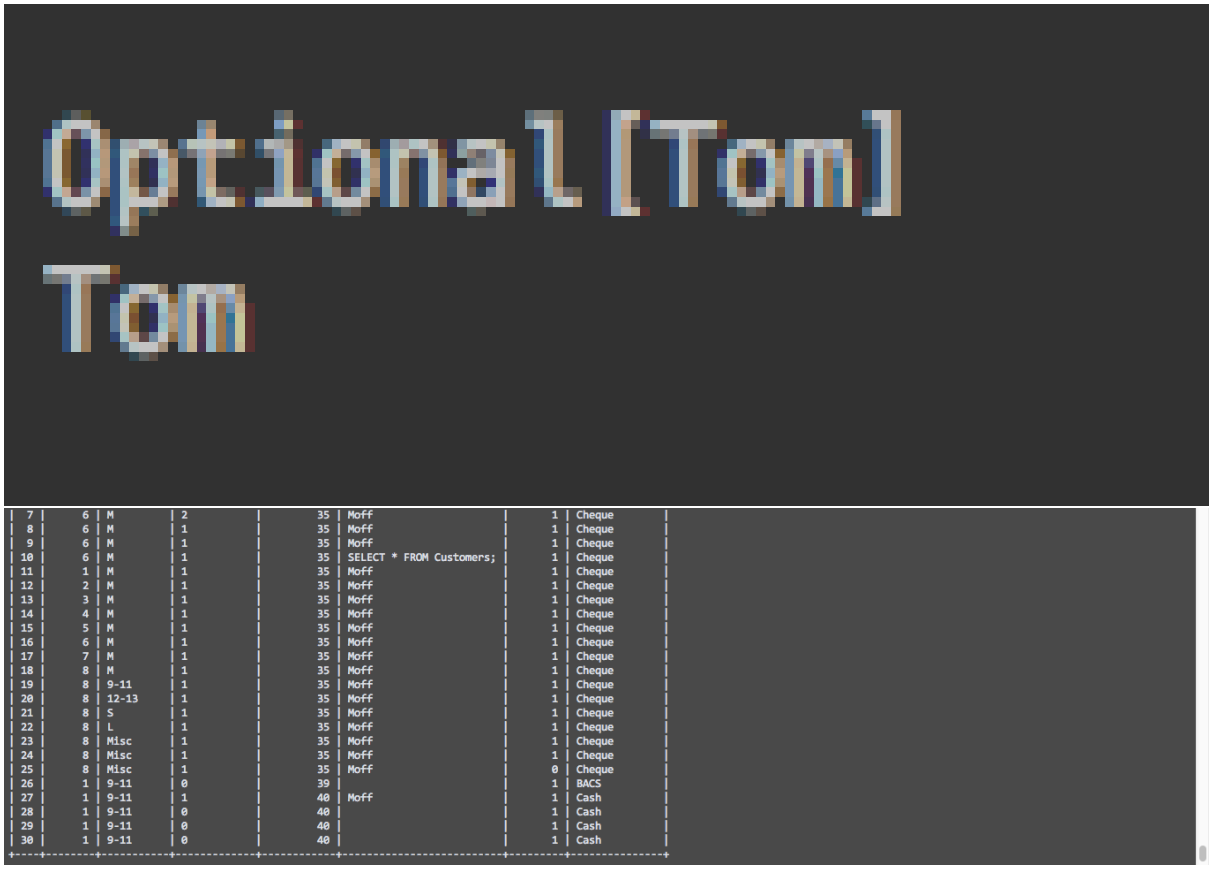
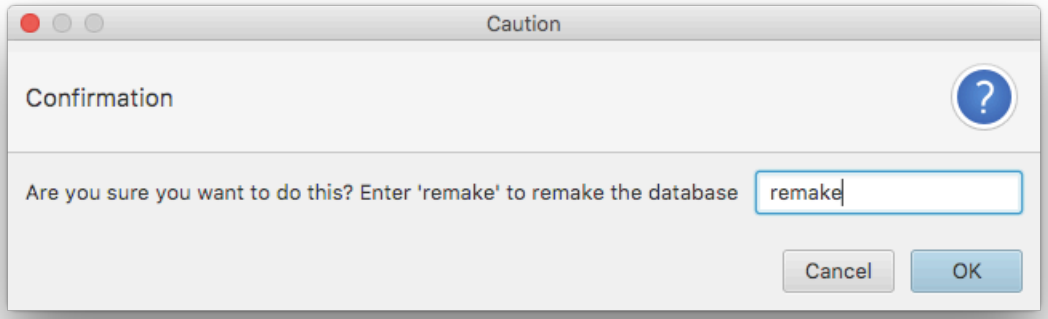
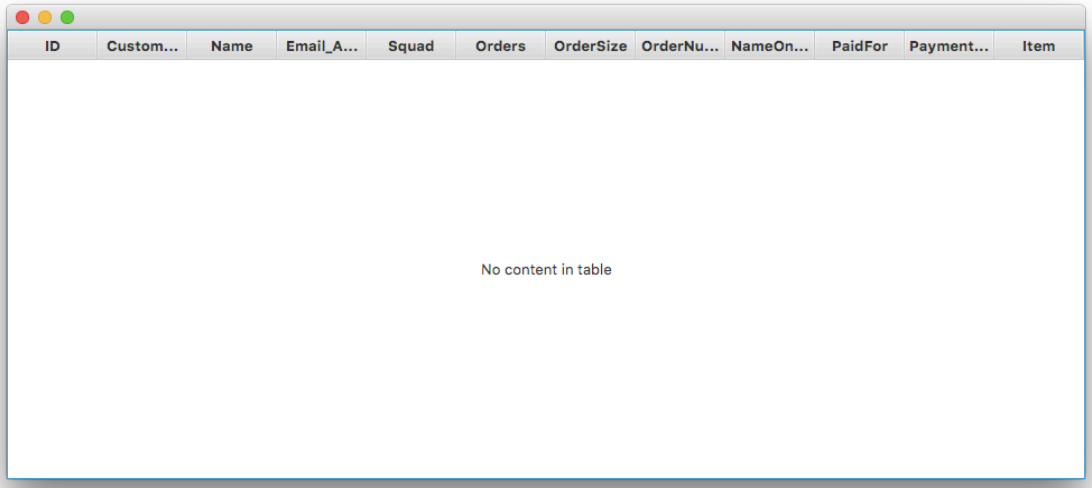


Figure 74: Evidence for test 18





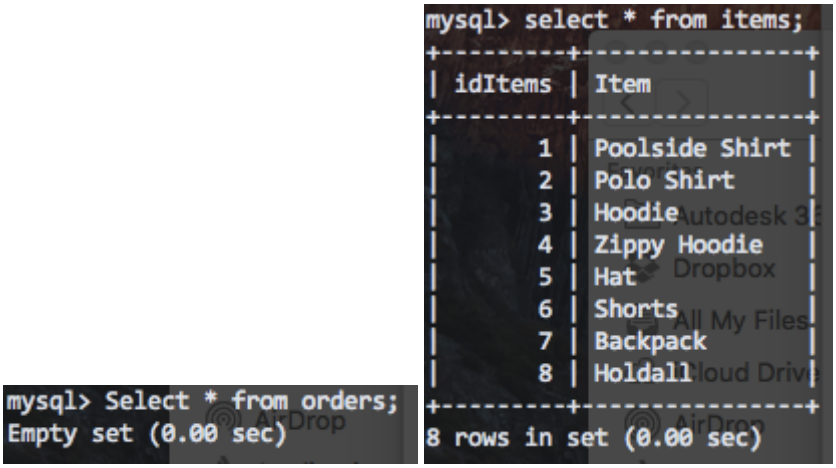
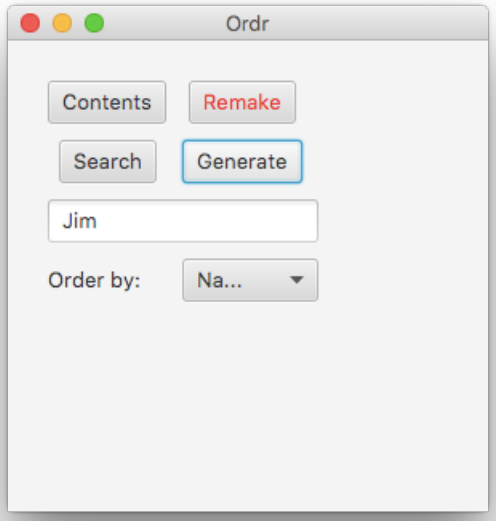


Figure 75: Evidence for test 19



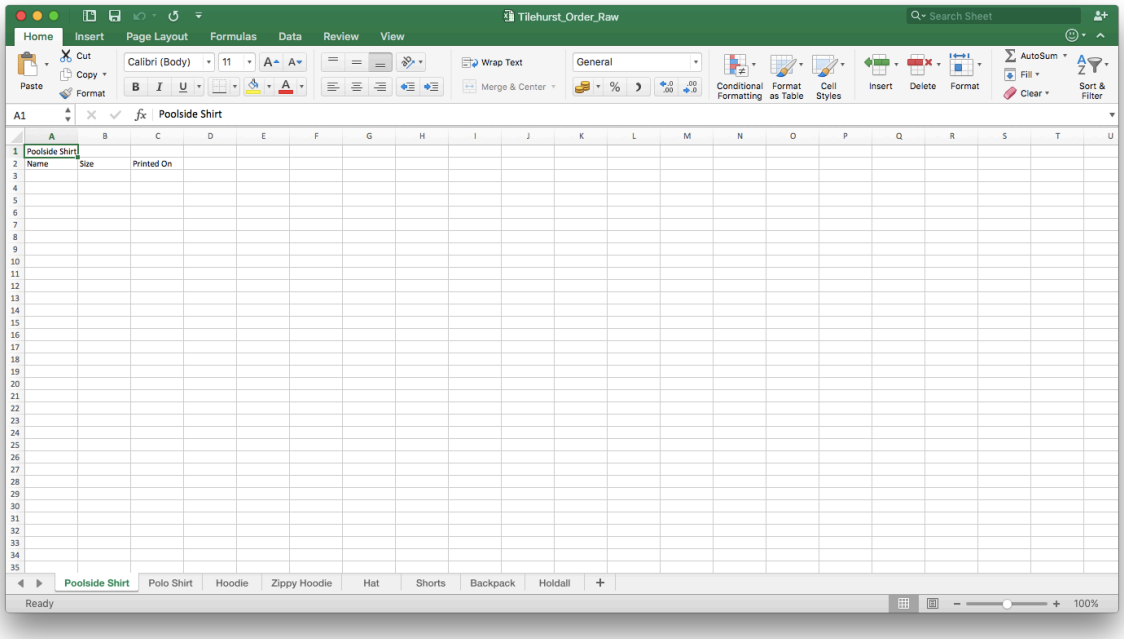
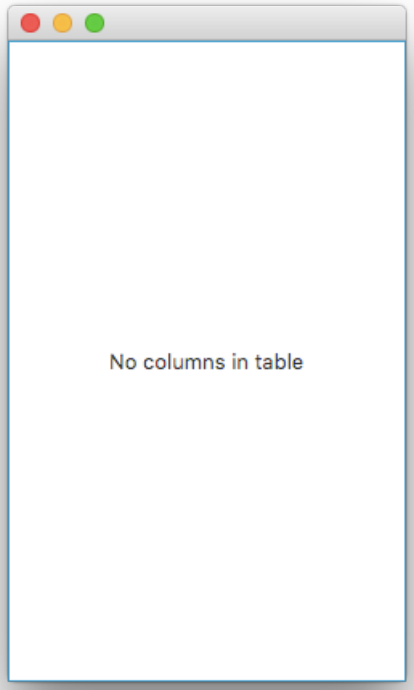


Figure 76: Evidence for test 20

```
Process finished with exit code 0
```

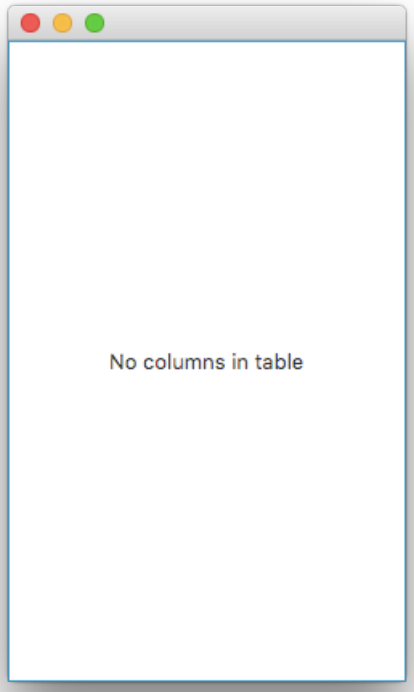
Figure 77: Evidence for test 21

c.2.2 Java Connection Testing Screenshots




```
Error on Building Data
java.io.FileNotFoundException: /Users/tsmoffat/kitordersystem/kitordersystem/kitordersystem/kitorder.xml (No such file or directory)
    at java.io.FileInputStream.open0(Native Method)
    at java.io.FileInputStream.open(FileInputStream.java:195)
    at java.io.FileInputStream.<init>(FileInputStream.java:138)
    at java.io.FileInputStream.<init>(FileInputStream.java:93)
    at sun.net.www.protocol.file.FileURLConnection.connect(FileURLConnection.java:90)
    at sun.net.www.protocol.file.FileURLConnection.getInputStream(FileURLConnection.java:188)
    at com.sun.org.apache.xerces.internal.impl.XMLEntityManager.setupCurrentEntity(XMLEntityManager.java:623)
    at com.sun.org.apache.xerces.internal.impl.XMLVersionDetector.determineDocVersion(XMLVersionDetector.java:189)
    at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(XML11Configuration.java:812)
    at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(XML11Configuration.java:777)
    at com.sun.org.apache.xerces.internal.parsers.XMLParser.parse(XMLParser.java:141)
    at com.sun.org.apache.xerces.internal.parsers.DOMParser.parse(DOMParser.java:243)
    at com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderImpl.parse(DocumentBuilderImpl.java:339)
    at javax.xml.parsers.DocumentBuilder.parse(DocumentBuilder.java:177)
    at kitordersystem.getConnection.setProperties(getConnection.java:72)
    at kitordersystem.getConnection.getConnection(getConnection.java:48)
    at kitordersystem.TableViewTest.buildData(TableViewTest.java:47)
    at kitordersystem.TableViewTest.start(TableViewTest.java:137)
    at kitordersystem.MainMenu$2$1.run(MainMenu.java:113)
    at com.sun.javafx.application.PlatformImpl.lambda$null$173(PlatformImpl.java:295) <1 internal calls>
    at com.sun.javafx.application.PlatformImpl.lambda$runLater$174(PlatformImpl.java:294)
    at com.sun.glass.ui.InvokeLaterDispatcher$Future.run(InvokeLaterDispatcher.java:95)
```

Figure 78: Evidence for test 1




```
java.sql.SQLException: No suitable driver found for jdbc:mysq://localhost:3306/
    at java.sql.DriverManager.getConnection(DriverManager.java:689)
    at java.sql.DriverManager.getConnection(DriverManager.java:247)
    at kitordersystem.getConnection.getConnection(getConnection.java:54)
    at kitordersystem.TableViewTest.buildData(TableViewTest.java:47)
    at kitordersystem.TableViewTest.start(TableViewTest.java:137)
    at kitordersystem.MainMenu$2$1.run(MainMenu.java:113)
    at com.sun.javafx.application.PlatformImpl.lambda$null$173(PlatformImpl.java:295) <1 internal calls>
    at com.sun.javafx.application.PlatformImpl.lambda$runLater$174(PlatformImpl.java:294)
    at com.sun.glass.ui.InvokeLaterDispatcher$Future.run(InvokeLaterDispatcher.java:95)
```

Figure 79: Evidence for test 2

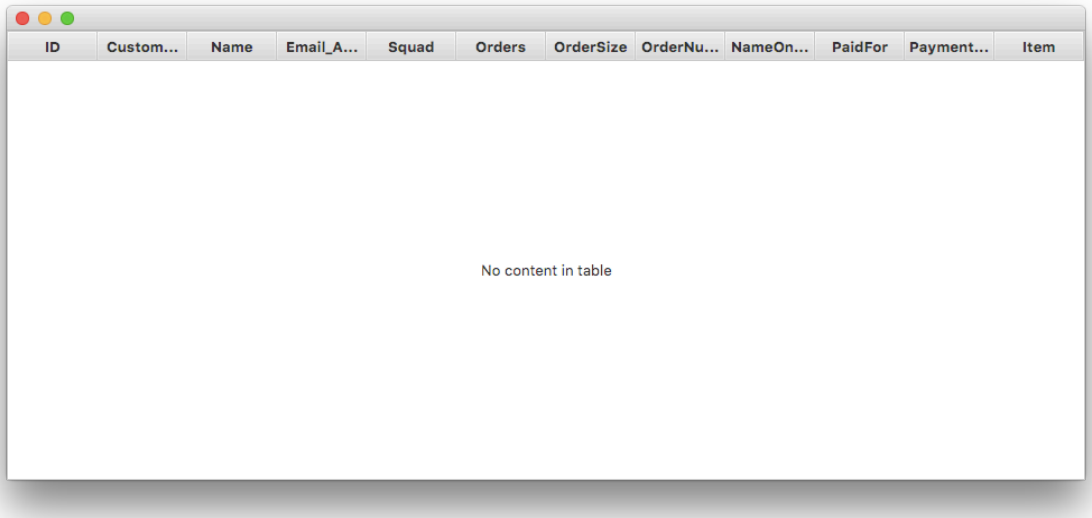
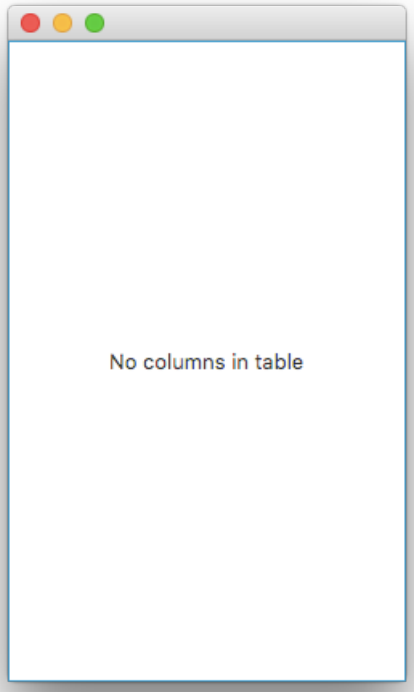
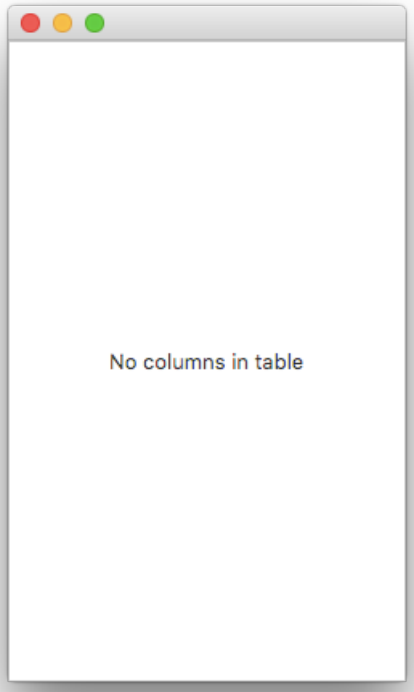


Figure 80: Evidence for test 3



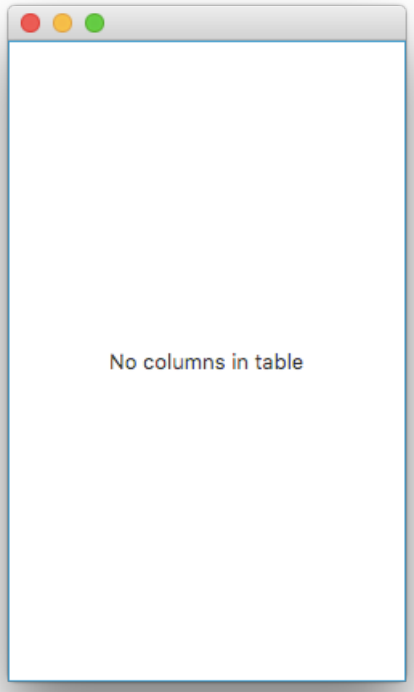

```
Connecting to: mysql://localhost:3306
java.sql.SQLException: Access denied for user 'roo'@'localhost' (using password: YES)
Error on Building Data
    at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:957)
    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3878)
    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3814)
    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:871)
    at com.mysql.jdbc.MysqlIO.proceedHandshakeWithPluggableAuthentication(MysqlIO.java:1694)
    at com.mysql.jdbc.MysqlIO.doHandshake(MysqlIO.java:1215)
    at com.mysql.jdbc.ConnectionImpl.coreConnect(ConnectionImpl.java:2255)
    at com.mysql.jdbc.ConnectionImpl.connectOneTryOnly(ConnectionImpl.java:2286)
    at com.mysql.jdbc.ConnectionImpl.createNewIO(ConnectionImpl.java:2085)
    at com.mysql.jdbc.ConnectionImpl.<init>(ConnectionImpl.java:795)
    at com.mysql.jdbc.JDBC4Connection.<init>(JDBC4Connection.java:44) <4 internal calls>
    at com.mysql.jdbc.Util.handleNewInstance(Util.java:404)
    at com.mysql.jdbc.ConnectionImpl.getInstance(ConnectionImpl.java:400)
    at com.mysql.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:327)
    at java.sql.DriverManager.getConnection(DriverManager.java:664)
    at java.sql.DriverManager.getConnection(DriverManager.java:247)
    at kitordersystem.getConnection.getConnection(getConnection.java:54)
    at kitordersystem.TableViewTest.buildData(TableViewTest.java:47)
    at kitordersystem.TableViewTest.start(TableViewTest.java:137)
```

Figure 81: Evidence for test 4




```
java.sql.SQLException: Access denied for user 'root'@'localhost' (using password: YES)
    at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:957)
    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3878)
    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3814)
    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:871)
    at com.mysql.jdbc.MysqlIO.proceedHandshakeWithPluggableAuthentication(MysqlIO.java:1694)
    at com.mysql.jdbc.MysqlIO.doHandshake(MysqlIO.java:1215)
    at com.mysql.jdbc.ConnectionImpl.coreConnect(ConnectionImpl.java:2255)
    at com.mysql.jdbc.ConnectionImpl.connectOneTryOnly(ConnectionImpl.java:2286)
    at com.mysql.jdbc.ConnectionImpl.createNewIO(ConnectionImpl.java:2085)
    at com.mysql.jdbc.ConnectionImpl.<init>(ConnectionImpl.java:795)
    at com.mysql.jdbc.JDBC4Connection.<init>(JDBC4Connection.java:44) <4 internal calls>
    at com.mysql.jdbc.Util.handleNewInstance(Util.java:404)
    at com.mysql.jdbc.ConnectionImpl.getInstance(ConnectionImpl.java:400)
Connecting to: mysql://localhost:3306
    at com.mysql.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:327)
Error on Building Data
    at java.sql.DriverManager.getConnection(DriverManager.java:664)
    at java.sql.DriverManager.getConnection(DriverManager.java:247)
    at kitordersystem.getConnection.getConnection(getConnection.java:54)
    at kitordersystem.TableViewTest.buildData(TableViewTest.java:47)
    at kitordersystem.TableViewTest.start(TableViewTest.java:137)
```

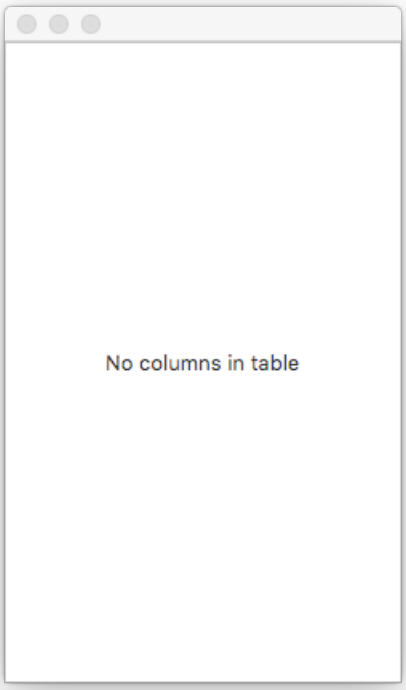
Figure 82: Evidence for test 5




```
com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link failure

The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server. <4 internal calls>
    at com.mysql.jdbc.Util.handleNewInstance(Util.java:404)
    at com.mysql.jdbc.SQLError.createCommunicationsException(SQLError.java:981)
    at com.mysql.jdbc.MysqlIO.<init>(MysqlIO.java:339)
    at com.mysql.jdbc.ConnectionImpl.coreConnect(ConnectionImpl.java:2253)
    at com.mysql.jdbc.ConnectionImpl.connectOneTryOnly(ConnectionImpl.java:2286)
    at com.mysql.jdbc.ConnectionImpl.createNewIO(ConnectionImpl.java:2085)
    at com.mysql.jdbc.ConnectionImpl.<init>(ConnectionImpl.java:795)
    at com.mysql.jdbc.JDBC4Connection.<init>(JDBC4Connection.java:44) <4 internal calls>
    at com.mysql.jdbc.Util.handleNewInstance(Util.java:404)
    at com.mysql.jdbc.ConnectionImpl.getInstance(ConnectionImpl.java:400)
    at com.mysql.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:327)
    at java.sql.DriverManager.getConnection(DriverManager.java:664)
    at java.sql.DriverManager.getConnection(DriverManager.java:247)
    at kitordersystem.getConnection.getConnection(getConnection.java:54)
    at kitordersystem.TableViewTest.buildData(TableViewTest.java:47)
    at kitordersystem.TableViewTest.start(TableViewTest.java:137)
    at kitordersystem.MainMenu$2$1.run(MainMenu.java:113)
    at com.sun.javafx.application.PlatformImpl.lambda$null$173(PlatformImpl.java:295) <1 internal calls>
    at com.sun.javafx.application.PlatformImpl.lambda$runLater$174(PlatformImpl.java:294)
    at com.sun.glass.ui.InvokeLaterDispatcher$Future.run(InvokeLaterDispatcher.java:95)
```

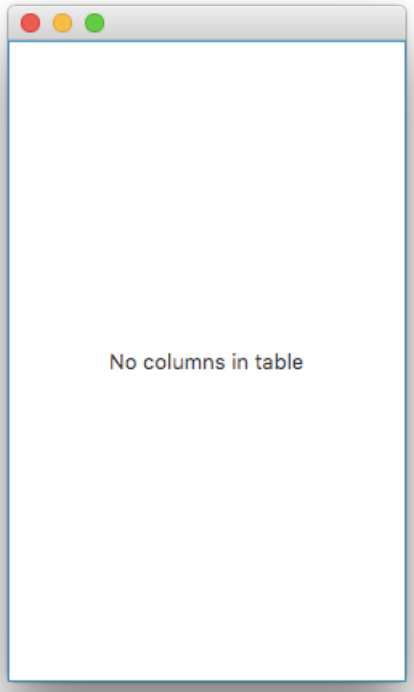
Figure 83: Evidence for test 6




```
Connecting to: mysql://localhos:3306
com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link failure
Error on Building Data

The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server. <4 internal calls>
    at com.mysql.jdbc.Util.handleNewInstance(Util.java:404)
    at com.mysql.jdbc.SQLError.createCommunicationsException(SQLError.java:981)
    at com.mysql.jdbc.MysqlIO.<init>(MysqlIO.java:339)
    at com.mysql.jdbc.ConnectionImpl.coreConnect(ConnectionImpl.java:2253)
    at com.mysql.jdbc.ConnectionImpl.connectOneTryOnly(ConnectionImpl.java:2286)
    at com.mysql.jdbc.ConnectionImpl.createNewIO(ConnectionImpl.java:2085)
    at com.mysql.jdbc.ConnectionImpl.<init>(ConnectionImpl.java:795)
    at com.mysql.jdbc.JDBC4Connection.<init>(JDBC4Connection.java:44) <4 internal calls>
    at com.mysql.jdbc.Util.handleNewInstance(Util.java:404)
    at com.mysql.jdbc.ConnectionImpl.getInstance(ConnectionImpl.java:400)
    at com.mysql.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:327)
    at java.sql.DriverManager.getConnection(DriverManager.java:664)
    at java.sql.DriverManager.getConnection(DriverManager.java:247)
    at kitordersystem.getConnection.getConnection(getConnection.java:54)
    at kitordersystem.TableViewTest.buildData(TableViewTest.java:47)
    at kitordersystem.TableViewTest.start(TableViewTest.java:137)
    at kitordersystem.MainMenu$2$1.run(MainMenu.java:113)
    at com.sun.javafx.application.PlatformImpl.lambda$null$173(PlatformImpl.java:295) <1 internal calls>
```

Figure 84: Evidence for test 7




```
Connecting to: mysql://localhost:3306
com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link failure

The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server. <4 internal calls>
    at com.mysql.jdbc.Util.handleNewInstance(Util.java:404)
    at com.mysql.jdbc.SQLError.createCommunicationsException(SQLError.java:981)
    at com.mysql.jdbc.MysqlIO.<init>(MysqlIO.java:339)
    at com.mysql.jdbc.ConnectionImpl.coreConnect(ConnectionImpl.java:2253)
    at com.mysql.jdbc.ConnectionImpl.connectOneTryOnly(ConnectionImpl.java:2286)
    at com.mysql.jdbc.ConnectionImpl.createNewIO(ConnectionImpl.java:2085)
    at com.mysql.jdbc.ConnectionImpl.<init>(ConnectionImpl.java:795)
    at com.mysql.jdbc.JDBC4Connection.<init>(JDBC4Connection.java:44) <4 internal calls>
    at com.mysql.jdbc.Util.handleNewInstance(Util.java:404)
    at com.mysql.jdbc.ConnectionImpl.getInstance(ConnectionImpl.java:400)
    at com.mysql.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:327)
    at java.sql.DriverManager.getConnection(DriverManager.java:664)
    at java.sql.DriverManager.getConnection(DriverManager.java:247)
    at kitordersystem.getConnection.getConnection(getConnection.java:54)
    at kitordersystem.TableViewTest.buildData(TableViewTest.java:47)
    at kitordersystem.TableViewTest.start(TableViewTest.java:137)
    at kitordersystem.MainMenu$2$1.run(MainMenu.java:113)
    at com.sun.javafx.application.PlatformImpl.lambda$null$173(PlatformImpl.java:295) <1 internal calls>
    at com.sun.javafx.application.PlatformImpl.lambda$runLater$174(PlatformImpl.java:294)
    at com.sun.glass.ui.InvokeLaterDispatcher$Future.run(InvokeLaterDispatcher.java:95)
Caused by: java.net.ConnectException: Connection refused
    at java.net.PlainSocketImpl.socketConnect(Native Method)
    at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:350)
    at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:206)
    at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:188)
    at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
    at java.net.Socket.connect(Socket.java:589)
    at com.mysql.jdbc.StandardSocketFactory.connect(StandardSocketFactory.java:211)
    at com.mysql.jdbc.MysqlIO.<init>(MysqlIO.java:298)
    ... 22 more
Error on Building Data
```

Figure 85: Evidence for test 8