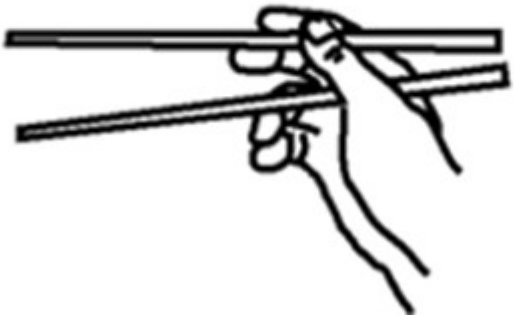
ASIAN FOOD FINDER



Asian Food Finder: project report

What we did, how we did it, the problems we faced and how we overcame them.

By group Curtana

Alexander Bartha, Magnus Johansson, Philip Malm, Henrik Persson, Nicole Musco, Meng-Jiao Wei
Software Engineering & Management Programme
Gothenburg University
2014

Table of Contents:

1. Introduction to Asian Food Finder

1.1 Group Members

Magnus Johansson, magnusjohansson82@gmail.com
Alexander Bartha, aabartha@gmail.com
Philip Malm, philip.malm@hotmail.se
Henrik Persson, henrik.persson1992@hotmail.com
Mengjiao Wei, weimenjiagillian@gmail.com
Nicole Musco, nicole.m.musco@gmail.com

1.2 The problem at hand

Asian food restaurants are very popular in the western world and have been for a very long time. However, as technology advances and the accessibility of delivery and ordering systems has sky-rocketed for markets for such things as fast food and pizza in recent years, the Asian food market has been lagging behind, especially in Sweden. There are currently no simple ways to find any nearby Asian food restaurants on the internet, let alone a system which actually lets you order the food to be made, and we plan to change that.
The purpose of this project was to develop a Decision Support System aimed towards guiding users to find and make use of Asian food restaurants in their vicinity. We planned to list the menus of each restaurant and equip each restaurant with a system that could receive orders from customers and respond to the customers requests. The program itself was developed and run through Java and the database of which was constructed in MySQL.

2 Methodology and tools

2.1 Methods used

We knew that we wanted an agile work method, seeing how we were all total beginners when it comes to both programming and all other aspects of software development. Agile meant we could do rapid changes and this was important because 1) we didn’t know anything about programming and it was impossible to estimate how much time any of the steps would take and 2) we had never designed a user interface before. Agile meant we could scrap our original designs, should we find better options along the way.

Our work method was a little bit similar to the scrum method with many milestones (sprints) and a project manager, Magnus Johansson, who acted only as a coach and communicator (scrum master), not a decision-maker. However as we didn't have a real product owner, we just kept a product backlog for our own sake and there was much freedom in what each

member could try and implement on their own. There wasn't really a need for daily scrum meetings either, because when you have six beginners, progress can be slow and harassing group members about what they had achieved since the day before seemed a little harsh. Instead we held three longer meetings per week on Mondays, Wednesdays and Fridays. This proved sufficient.

All individual members were also required to document their own work. These notes proved very helpful when we wrote this final report. For our project, we separated ourselves into three smaller groups. Each group was in charge of an interface and its entirety. More about this in chapter 5, work distribution.

2.2 Tools used

- phpMyAdmin(for handling our MySQL database)
- Eclipse IDE
- TeamworkPM
- Dropbox
- Google docs
- draw.io

All programming of the Asian Food Finder system was done in Java. We chose to use Eclipse simply because it was recommended by our supervisors. We benefitted from the fact that we were all using the same IDE, problems related to Eclipse became easier to solve, as we all trained more and more with it and could help each other out.

PhpMyAdmin is a MySQL interface for usage in a web browser. Basically we had a server up and running 24/7 so everyone could access the database at anytime they wanted. In order for this to work we acquired a free DNS host from no-ip.biz which we then connected to our server.
PhpMyAdmin proved a very good tool, not only was the interface fairly simple and easy to use, but it was also powerful. Not needing the rather bulky MySQL software installed locally and being able to access the server from anywhere and test different SQL queries on the fly before you try them in Java was immensely useful.

TeamworkPM was used for scheduling, setting up deadlines for our sprints and for taking notes in the notebooks.

Dropbox only served as storage for the project files.

Google docs was used only for this final report. The fact that it allows for simultaneous editing and can show you revision history (with color coded changes) makes this an excellent tool for this kind of documentation.

Draw.io is a simple free online drawing tool for things like E/R and UML diagrams, tables and such. While Oracles MySQL comes with built-in tools for this kind of thing, it was nice to use something that was available online from anywhere.

3. System overview

Curtana set out to develop a system which helps hungry people to find nearby restaurants, to browse their menus and order food from a restaurant of their choosing.
The customer will first need to provide their general area, the category they wish to choose from (Indian, Thai etcetera) or their local postal code. All of these may help to locate the restaurants in the desired area for the customer.
They will then be presented with a list of restaurants from which they will select the one whose menu they would like to see. If they wish to place an order, they will select the meals they desire. Upon adding the desired dishes, these will be placed in the 'shopping cart' displayed along with the total price of the order. The user will then need to confirm the order and provide their personal information, such as phone number and email address. Once the customer has confirmed the order, it appears in the restaurant's interface along with contact information to the customer.
The restaurant personnel will then need to state an estimate of how long the order will take to be completed. Finishing this prompt will send the user a message to the provided email address, stating that the order has been received and is being prepared, as well as the estimated time that the restaurant has given for pick-up.
The customer can then go to the restaurant and pick up his or her order.
The restaurant personnel can also update their menu and information about their business in the same interface as the orders are placed.
Finally there is an administrator application that allows the company owner to add/remove and change information about the restaurants in the system

4. Requirements

4.1 Customer Interface

Our goal was to create an easy and smooth interface for the customer with minimum amounts of buttons, to not create confusion.

Search frame
This is the first interface that the customer will see which only contains a Search button that takes them to the next step after they have made their choice. There is also a simple dropdown menu containing three different alternatives to search by which is Area, Postal number and Category. Each one of them has different options that the customer can choose from.

Area contains the choices:

- Central Gothenburg
- East Gothenburg
- West Gothenburg

- North Gothenburg
- South Gothenburg

Postal number is a simple textfield where you enter a postal number.

Category search contains:
- Chinese food
- Thai food
- Sushi
- Indian

After the customer has chosen a search method and pressed the Search button, they are taken to the “selectframe”.

Select frame

This is where the list of restaurant will be shown depending on the user’s choice of search option.
A simple JTable shows the restaurants with all their information and two buttons are on the bottom “Back” and “Next”.
The customer can either press the “Back” button to return to the “Searchframe” or they can press “Next” when they have chosen a restaurant to their liking.

Restaurant frame

Consist of two tables (Menu table and shopping cart table) and four buttons, which are;

- Add: Adds a dish from the menu to the shopping cart
- Remove: Removes the selected dish from your shopping cart
- Back: Takes you back to the previous frame.
- Checkout: Takes you to the final step.

Basically the customer selects a dish from the restaurant’s menu and presses the add button which copies the dish and adds it to the empty table. In the bottom there is also a simple counter that keeps track of the total order price based on which dishes (and their quantity) are selected

Checkout frame
The customer enters necessary information for delivery or if the restaurant needs to contact the customer. That is Name, Email address, phone number and home address.

After the “Finished” button has been pressed the contact information and the order will be sent to the database which in turn will be sent to the restaurant owner.

4.2 Restaurant interface

The restaurant interface is the most important one and needs to be working at all time without any errors. It also needs to be simple and easy to use for the restaurant owner so that mistakes are harder to do.
To be able to access the restaurant interface the admin has to create an account for the restaurant owner (via the administrator interface) so the owner can login and use the system. When an account has been made they are assigned a unique id (restID) which is used to display correct information for the restaurant depending on which account is logged on. The restaurant interface contains four different tabs, Menu, Orderlist, New orders and Restaurant info.
The menu tab is really simple. It retrieves the menu from the database table “Menu” depending on which user that logs on and displays it as an table. Other than that there are three buttons “Add dish”, “Delete dish” and “Update menu”.

The “New orders” tab is the most important one, this is where the restaurant owner will see the orders that have been placed and are given the options to accept the order or decline it for various reasons. If the restaurant is not able to receive orders the system has no use besides showing restaurant info.

New Orders are divided into two halves.

The incoming orders that are received are displayed as branches on a filetree on the left side. Each order is given a node which is named after the order id (a simple number) and when clicking the node the dishes will be shown together with the ordered amount of each one.

On the right side of the window, the customer’s information is displayed when an order in the tree is selected. Underneath the customer information, there are also two buttons, “Accept” and “Decline”.
Pressing “Accept” prompts the restaurant owner to enter how many minutes the order will take to prepare. An email will then be generated and sent to the address the customer has specified. This mail states that the order is being handled and shows the estimated time of completion.
If “Decline” is pressed, an email will be generated and sent with an apology that the order cannot be processed at this time.

Accepting an order also means the order ends up in the “Orderlist” tab of the restaurant interface.
This tab works in a similar way as the New Orders, but keeps track of orders that the restaurant staff are preparing for delivery.

Orders are again displayed in a tree to the left. When selecting an order with the mouse, the customer information is displayed on the right hand side.

After it has been done all the active orders are sent to the order list in chronological order. When the order is completed and delivered to the customer they can simply click its branch in the tree press “done” to have it removed.

4.3 Admin interface

Upon start a small window pops up and you are prompted to log in by entering a username and password in the two text fields that appear (only accounts with admin privileges are able to log in to this interface).

The user is then greeted by a big scrollable table containing all restaurants in the system and all of their basic information, like phone number, opening hours or street address for example.

This table is not editable (this is to ensure that the user does not change restaurant information by accident). Instead when you click a restaurant with the mouse, its information is displayed in the “insert table” at the top of the window. This only contains a single editable row with information about the selected restaurant: its identification number, restaurant name, general area, phone number, opening hours, street address, postal number and city.

Beneath this editable row are two buttons: “Delete restaurant from DB” and “Update restaurant”. The delete button will prompt the admin with a pop-up window asking “Are you sure you wish to remove restaurant ‘X’?” Pressing yes will remove the restaurant and all associated data from the database.

Pressing the update button will insert the new data into the database.

Up top there is a menu bar with an “Actions” menu. Within are two buttons “Create new user”, which prompts the admin via pop-up to enter the name of the restaurant, the name of the owner, and a new passford for the owner.

The other button is “Refresh list”, this is for monitoring the restaurants in the system and updates the big table with the most recent data in the database.

4.4 The database

The database consists of the following tables:

RestInfo - holds restaurant information

RestID	Primary Key
Area	
RestName	Name of restaurant
PhoneNumber	
OpeningHours	
StreetAddress	
PostalNumber	
City	
Category	e.g. “Chinese” or “Thai”

Menu - stores menu information

RestID	Primary Key
DishNumber	
DishName	
Ingredients	
DishPrice	

Users - stores the user information for restaurant owners and system administrators

UserName	Primary Key
Password	Primary Key
Role	Can be ‘restaurant’ or ‘admin’
RestID	‘Null’ if user is administrator

Orders - connects the order with the customer, contents of order and saves a timestamp for when the order was placed

OrderID	Primary Key
CustomerID	
Timestamp	

OrderContents - holds the contents of a single order

OrderID	Primary Key
RestID	
DishNumber	
Quantity	
DishPrice	

Customers - stores the contact information of the customer placing an order

CustomerID	Primary Key
CustomerName	
CustomerPhone	
CustomerEmail	

CustomerAddress	

5. Work distribution

Our group decided we could work more efficiently by dividing ourselves into three smaller groups of two people. Also our group elected Magnus Johansson as group manager (important distinction, his role was only that of organizer, not leader in any way). Each group made their own tasks and deadlines to complete them by and divided the work as they saw fit themselves. The work was divided as follows, (more details on who did what follows in the chapter Implementation).

Customer interface: Mengjiao Wei and Nicole Musco
Database and admin interface: Magnus Johansson and Henrik Persson
Restaurant interface: Alexander Bartha and Philip Malm

Magnus also implemented the whole real time ordering system for the restaurant interface, including setting up the e-mail client, helped fix the shopping cart in the customer interface, organized and led all of the meetings and kept track of sprint goals and scheduling. He spent approximately around 400 hours actively working on this project.

Alexander also merged the admin interface with the restaurant interface and created a shared login system for these two programs.

For the majority of the semester, the group as a whole met between two and three times per week to discuss what has been accomplished and what should be completed by the upcoming meetings. During the break, group members were spread out all over the world, as we have a diverse group. Meetings with those who remained in Sweden were often and the information was relayed to those who were away and the project continued to move forward with development as normal.

6. Implementation

6.1 The Database

Creating the database architecture:
We started off by creating an Entity/Relationship Diagram using the program “draw.io” The schema for the database was also initially sketched in this program. The ordering system provided us with some headaches. A single order should contain every dish the customer has ordered and the quantity of each dish. Our lack of experience led us to initially try to place the contents of the order inside the order table. This would generate a lot of duplicate rows so we moved the columns with order contents (dishnumber, quantity of dish and dish price) to its own table - "order contents" - along with the OrderID from the order table, which was used for identification as a foreign key in the new table.

Initially we had a lot of information in the same table, such as all customer contact information related to a specific order contained in the order table.
The customer contact information was lifted out to its own separate table, leaving only the customerID field behind in the order table for identification. Our reasoning was that this would be very helpful if we later on got to the point where we would want a feature where customers could actually create an account with personal login so we could save their contact information (however due to time constraints this feature was never implemented, still we feel that it was the correct design decision to make at the time).

Setting up the database:
We set up a DNS-server at Alexander Bartha’s home, using a service from no-ip.com which provided us with an address. To be able to access our work from anywhere was of great benefit and also made it possible to create a “real” functioning ordering system. The tool “phpmyAdmin” was used to set up the database (which was of MySQL type).

As soon as we had made our first working queries using an extremely basic test program we started working with the GUI teams to implement some database functionality; adding the JDBC drivers to java, loading them up in the code, creating the connections and setting up statements and retrieving result sets. The restaurant user was soon able to edit basic information like: restaurant name, street address, general area, city, postal number phone number, opening hours. The customer interface was able to access the same table and retrieve information on which restaurants existed in a certain general area.

6.2 The Customer Interface

This is the interface that will be seen and used by the customers themselves. This GUI's purpose is to help the customers find an Asian restaurant near them and to their particular desire so that they can then order their meal(s) and have a quick pick up when arriving to the desired location. We started with making a frame to browse through various searching possibilities. Possibilities such as general area, category of food, postal code search, and a free text search box. We further created the frames for the list of restaurants with their personal information. Once we had the layout for these frames relatively completed, we moved on to the frame for each individual restaurant with a checkout frame to enter personal information and a confirmation message that the order was completed.

Through the development of this interface, we faced a few difficulties that we have managed to fix and improve. While creating everything, we began to realize there was an organizational issue due to the fact our coding was mostly in just one class. In order to resolve this issue we began to slowly rewrite the program using a separate class for each frame to easily detect errors within the interface. Now, there are six classes which are simpler and easier to manage.

Through the development process, we had some difficulties trying to have a table appear on one of our frames using the 'GridBagLayout' so we decided to try and use just the setBounds and the tables successfully appeared where we needed them and when we wanted them to. SetBounds can set location, width and height. At the end of the class, set every label visible, and it show up on the exactly place in the window. The big problem we met during the programming is ordering system. In the ordering system we have two tables, one is menu table and other is shopping table. In the beginning, we try to click one row then add the add button, this row will appear on the shopping table. However, we found it is so difficult to make that work. So we decide just get the dish name and price. For this part, we use getSelectionModel().addListSelectionListener to get the row which is customer selected, and store them in the SelectionModel as object. Then we add an ActionListener to add button is the default model of shopping table add row. Then the dish name and price which customer selected show up on the shopping table. A big part of this issue was that, before the rearrangement of the code, it was very hard to find which pieces were not compiling. Parts of certain frames were in different places and now that it is all organized we have been able to easily detect where the code works and where it does not finish.

Some difficulties we have had during the development process, that aren't necessarily in our programming skills, are that we were located in different parts of the world at the last stretch of programming and compiling. It had become difficult to communicate to other members of the group when you need something answered quickly. Meetings were not as often but we still had our tasks to finish so we managed as well as we could.

6.3 The Restaurant interface

For the basic GUI, we created panels for the different interfaces, for example, 'Menulist' and 'Restaurant' info. After they were, for the most part, finished we started working on making tabs for the different interfaces. This is where the problems started to occur. The tabs wouldn't show given interface and would cause the program to crash. The problem was when we tried to import the different classes to the 'tabbedpane'. We had to instantiate the classes in the mainframe and add them to the 'tabbedpane' afterwards. When we completed the tabs, we had to add 'scrollpanes' which added more problems. The 'srcollpanes' wouldn't appear at all so we tried several different ways but in the end the solution was adding the scrollpane to a panel which we then added to another panel.

As our interface worked somewhat okay, we decided to add some functionality to it. We started with the easiest one which was editing the restaurant info in the database via our interface. This went smooth and not many problems occurred besides the update part, which was easily fixed by making the resultset updatable. We also had one major change of the design with the restaurants menu. We started with out with having a JPanel for each dish and this would obviously take up too much space making it difficult for the restaurant owner to get a clear view over the menu. So we decided that an JTable would suit us better because it simplified the process of importing the menu from the database and updating it.

While creating the basic GUI, we first started off by using ‘GridLayout’ but had trouble with it, so we moved on to ‘GridBagLayout’ which allowed us to customize the look of the GUI like we wanted. However, through the classes we have, ‘GridBagLayout’, ‘GridLayout’, and even ‘FlowLayout’ being used. We had a problem in the OrderList class as the JTextFields and JLabels were placed on top of each other, not allowing us to add a certain amounts of rows of fields. This is when we first used ‘GridBagLayout’, but then we consulted with a classmate as he showed us that you could add 2 empty JLabels, but by using ‘GridLayout’. This managed to fix our problem, allowing us to add more rows of fields to the GUI. Overall, the creation of the GUI was not very difficult, just a little bit time consuming because of the repetition of code for the JLabels, JTextFields, etc. This is what the GUI mainly consist of for the most part.

The ordering system was complex and time-consuming. A timer thread was created, it querys the database once every ten seconds for new orders. The results from the database is then sorted into two structures. The customer information is stored in a linked HashMap with the order number as key identifier. The contents of the order is stored in a LinkedListMultiMap from Googles Guava framework, this was a suitable container as it allows the storing of multiple values for the same key, in this case several dishes may be associated with the same order number.
The dish information is stored in a dynamically built JTree which appears under the new orders tab. Each order gets it own tree node or branch, with the dishes working as leaves. This proved a nice slick solution and it was a very nice change from the constant use of Jtables. Clicking a branch or order in the tree displays the customer information on the right side of this split panel. This also allows the user to click accept or decline via their respective buttons. Clicking the button generates an email that is sent to the email address provided by the user. If the order is accepted, the restaurant personnel will enter a time estimation and this is then sent to the customer. A decline tells the user via email that the restaurant is to busy to handle their request.

Accepting also places the order under the active orders tab, in a similar simple Jtree which makes for an interface easy to understand for the restaurant personnal. This Jtree is built in a similar way and the customer information is again stored in a map.
Here there is only one option needed, and that is to click “done” once an order has been prepared and delivered to the arriving customer.

6.4 The admin interface

We were able to have the most important admin functions working for the beta presentation; the restaurants were displayed in a table on startup. By clicking a restaurant in the table it appeared in a textfield on top. Underneath the textfield were two buttons, "delete" and "modify" which performed as expected. There was also a menubar with the option of adding a new restaurant (a pop-up window appears, prompting the user to enter the name of the new restaurant). This made for a very simple interface, but this is more or less all functionality that we expect for the admin; the idea being that all restaurant owners are responsible for their own respective information. We decided to stick with GridLayout for the time being as making the change at this point would prove too time-consuming for us, and we also feel that aesthetics are not a priority when it come to the admin page.

With fixing the 'Add Restaurant' function, the user can cancel at any time, although the restaurant can be named virtually anything at this point. It is appropriate to be able to do so as there are many restaurants with names made up of various signs and numbers.

A custom Java database class was created which was later also used with the restaurant interface. It contains methods for connecting, querying and retrieving resultsets all with explicit exception handling, which made any database error handling easier. We also made our own tablemodel for our JTables, which had built in methods for saving a MySQL JDBC resultset directly into a Jtable and the method to listen for changes and update the table accordingly.

When waiting for the other teams to finish some work we added extra functionality to the admin interface, the admin may now add or change all the basic information of a restaurant, such as contact information and address (the original requirements only stated that the admin should be able to add and remove restaurants). We also switched to GridBagLayout for a better looking GUI.

We stumbled upon the problem of how we should store each of the pending orders (orders that have not been confirmed by the restaurant) locally or in the database. There are a lot of safety aspects to consider when building this sort of system. What if the power goes out at the restaurant or the operating system crashes? We implemented some code in the restaurant interface that saves data to a text file on the last received order, but if this was a real world application, we would need a lot more safety measures, for example saving all the order information to file once in a while.

7. Outcome and results

We fulfilled all the goals stated in our project plan. The fact that we during one semester went from total beginners to actually creating a real time ordering system, with several search functions, browse-able menus and lots of administrative functions, could be considered quite an achievement.

7.1 Changes that were made to requirements

We implemented the category food search (sushi, Indian, Thai etc.) fairly late in the project. However to allow the restaurants to belong to more than one category would have meant major changes to the database and/or code and at this point we didn’t have the time to do so. While our original requirements didn’t specifically state that a restaurant should be able to have more than one category, this was clearly an oversight on our part.

7.2 Changes done to database

The database team realized that we might as well somewhere along the way create a table with order history (it would contain dates/times, quantity, which types of dishes that were ordered and the prices of previously placed orders), this would not require very much extra work for us and could benefit the company/the restaurants immensely with statistical data on the consumer habits if we had time somewhere to implement this feature later. It was interesting to see how you along the way suddenly can realize completely new potential for your system (although some would probably argue that this was a feature we should have predicted and planned for all the way from the start?).

7.3 Non-functional requirements

Some thoughts on where we stand after the project regarding our non-functional requirements from the project plan.

Maintainability: The code may not be well structured everywhere, simply because we were not at all proficient with Java when we began, however comments in the code should help with the maintainability if the system was ever upgraded/revised.

Performance: It's difficult to write high performing code as a beginner. For the restaurant interface, we should have used SwingWorker for background tasks such as checking for new orders and sending e-mails, unfortunately there was not enough time and a simpler timer with it's own thread had to be used.

Efficiency & emotional factors: It should be both fun and simple to use the system. We've strived for simplicity and making good looking GUI:s despite the limitations of Java compared to for example html5. We never used any drag and drop programs like NetBeans, all code has been made "by hand".

Security: We don't store any unsafe information like credit card numbers in the database. Important data like passwords might not be safe enough from skilled hackers if this was a real world business application, but we've done our best regarding this.

8. Lessons learned

8.1 Some thoughts on developing in Java

Having never programmed before, it's hard to say anything insightful about the language you're using for your first project. However using Java as a language we realized it's a very well documented language, and most of the time you could find help online with the problems you were having. Searching forums we could also re-use some components from forums like StackOverFlow, this really showed us the power of both open source solutions and component based programming.

8.2 Version control

We really need to be better in using version control properly. In our defense this has not been taught during any of the lectures yet. Using Dropbox and uploading the java project folders without naming conventions made it difficult to know which version we were using. We named them after the date upon they were created, but this was far from optimal.

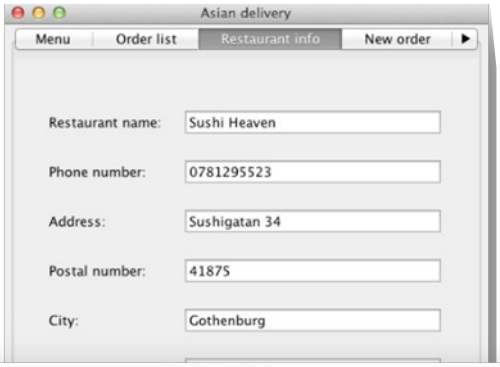
8.3 Scheduling and keeping track of progress

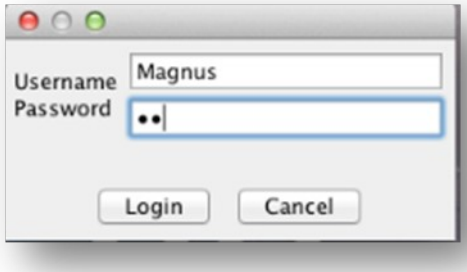
While I'm glad we did an initial schedule and find it very useful, we had to push several deadlines forward, simply because programming took a lot longer than expected. It's difficult to measure progress during software development and setting up sprints and goals really helps to get you an idea of where you are in the project.

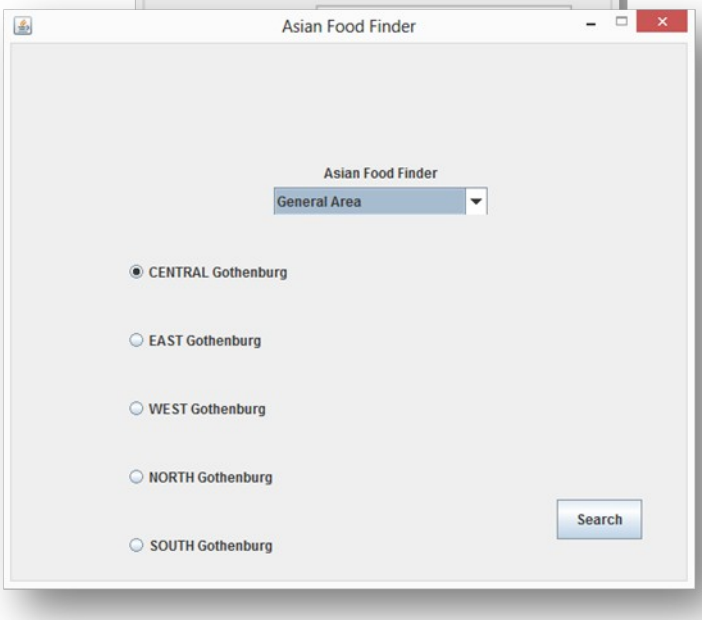
Using a collaboration program like Teamwork PM can be really useful; however it's required that all members actually use the program continuously. Storing our notes on the work here was really clever, as you could check up on other teams' progress and in best case scenarios learn something from their notes. For the final stage of the project it wasn't used actively enough so the manager just put up a to-do list as a text file on Dropbox.

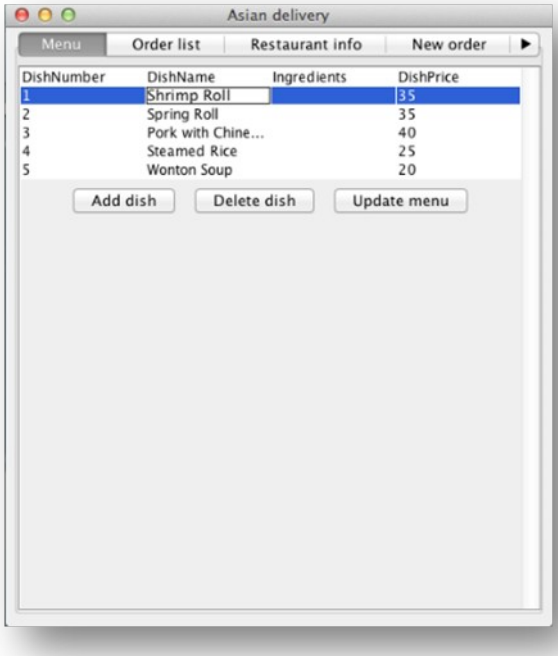
Appendix A: The system in use

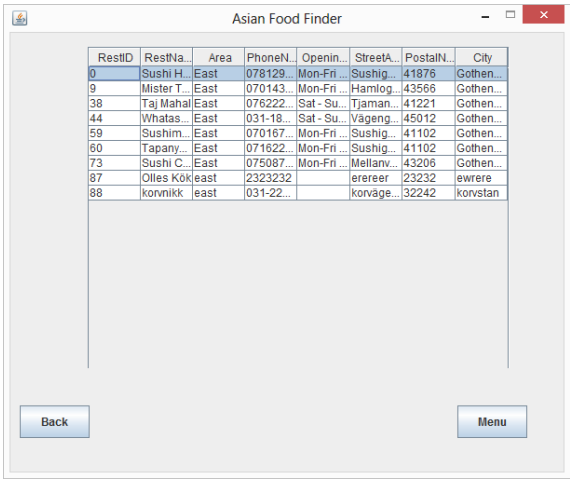
Customer Interface:



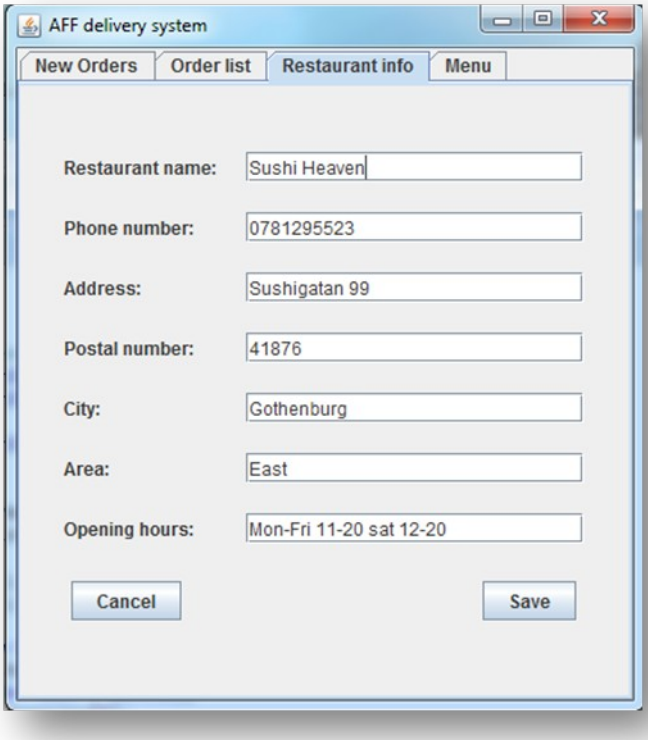


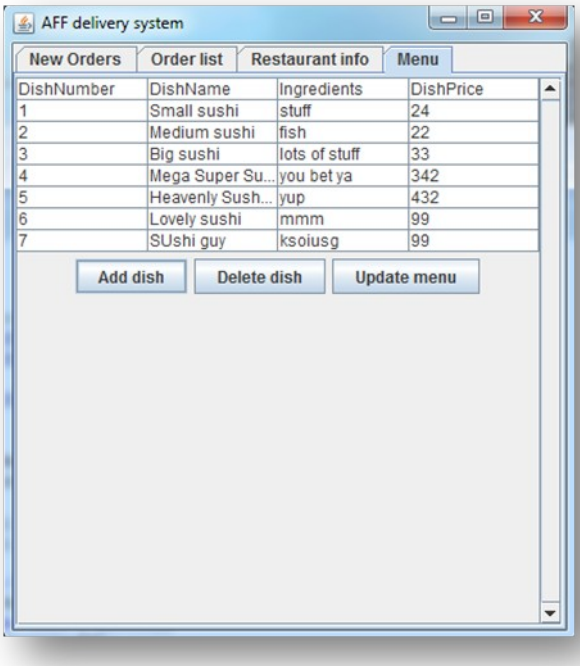


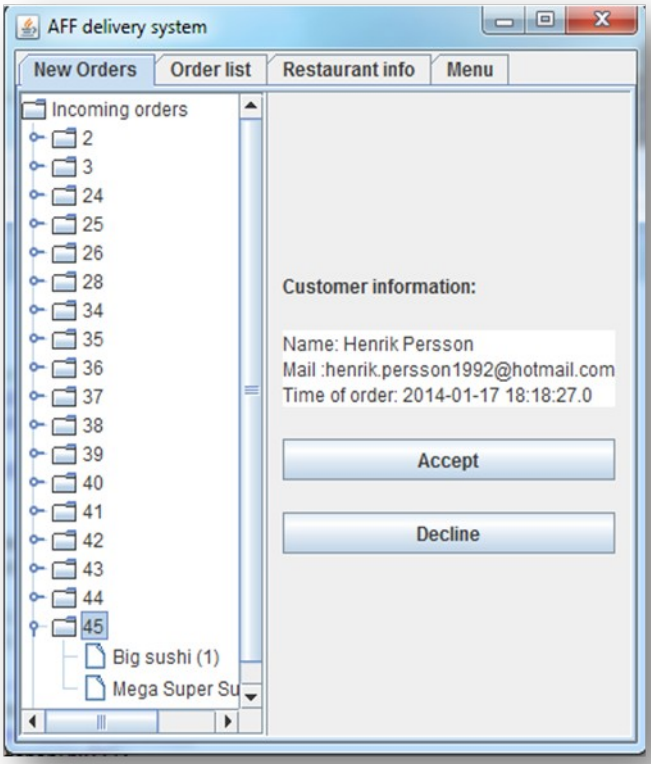




Restaurant Interface:







Admin Interface: