# System specification

The system will be developed using the XP process. Since continuous communication with the client is not possible within this case study, feedback will be simulated; however the business cases will be tackled from an XP context. The Pair programming metaphor is also required with respect to XP, however since I am working individually; this aspect will also be simulated. Ultimately the application of the principles of the programming methodology will illustrate the points required effectively.

The initial phase of the XP software engineering is the description of the system as a metaphor (Kent Beck n.d.). By applying a correct metaphor to the system, communications channels and the required information may be correctly abstracted from the system and a successful software product may be developed.

In this case, the metaphor to be generated is to be based on various different ‘stories’ regarding the use-cases the scenario is tackling. The scenario is therefore broken down into a number of stories, which may be seen below:

As a client entering the website, I want to be able to see a list of car categories and select the type of car I like on the front page.

As a client entering the website, I want to be able to search for a car based on the parameters of colour and description.

As a client, I would like to view multiple pictures of the car in question, as well as download information regarding the car in PDF format to print out or view later.

As a client, I would like to generate a number of reports of cars I like which I can print out and bring with me when I come to actually view the cars.

As a client, I would like easy access to all the pertinent servicing information regarding the car that I am viewing. I would also like to be able to save this information for later retrieval.

As a client, I would like to have a login with the ability to save cars I am interested in viewing.

As a client, I would like to be able to send the list of cars that I am interested in viewing before I arrive so they can be prepared for me to view.

As a content manager, I want to be able to view the articles I submit for the front page first as I enter the website.

As a content manager, I would like to be able to flag a particular article or a number of featured articles so that they shop up highest on the front page.

As a content manager, I would like to be able to edit my articles online and save them frequently. I would like to have a drafting facility which would allow me to save unfinished articles in my content management system which I would then continue to publish online as I finished them off.

As a content manager, I would like to view the last article I was editing when I login back into the system having used it.

As dealership manager, I would like to have a categorised list of all the cars that I have in my dealership in such a way as to have as many cars as possible displayed.

As dealership manager, I would like to search cars by registration number for editing.

As dealership manager, I would like a simple way of adding multiple cars in an easy sequence, so I can add a number of cars after each other when a new shipment of cars comes in.

As dealership manager, I would like a way for my clients to indicate which cars they are interested in before they arrive, so I can prepare them for a viewing.

As dealership manager, I would like to be able to view which cars have the most requests for viewing to understand which cars are most popular.

As dealership manager, I would like a simple way of adding multiple photos of different cars in the adding process.

As dealership manager, I would like to be able to add photos of new cars later in the process by searching and editing each car.

With these user interface stories, a clearer idea of the user features may be derived.

## Development Schedule

The development schedule chosen is based on the generation of functional tests, as well as the code required for these tests to pass and new code to be generated. The schedule is therefore laid out as follows:

1st – 7th Feb 2013: Generation of CW1 document with user acceptance stories and choice of process model.

10th - 14th Feb 2013: Generation of prototype implementing screens for the different user interface stories as required.

15th – 20th Feb 2013: Generation of screenshots as required for the submission of CW1 prototypes.

21st Feb – 5th March 2013: Generation of code to pass unit tests, starting work on the generation of CW2 document. Generation of quality standards as required for the development of CW1 prototypes.

5th – 15th March 2013: Architecture of the system built with the MVC design pattern in mind. Database model and queries built, and object relationship system constructed.

15th – 20th March 2013: Generation of CW2 tests as required for section b of CW2. Explain the MVC architectural pattern in the document as required.

20th – 25th March 2013: Implementation of CVS based system based on Git, generation of screenshots necessary to give evidence of versioning and the use of the git push command shown.

25th March – 15th April 2013: Building the system to the extent that all tests are passed. Demo the tests and record the working system using screen recording system.

15th April – 20th April 2013: Implementation of the Bugzilla reporting system and Bugzilla Reports reporting tool with screenshots to be done here. Post mortem analysis of the development would also be finished.

# Process model chosen

The development of complex software systems is a difficult design activity, not because of the complexity of technical problems but because of the social interaction when users and system developers learn to communicate (Greenbaum 1991)

There is an important social dimension in the development of software processes, as even at the lowest level; software design is a communicative activity. The main challenge is communicating the requirements of the users effectively to the designers. Technical hurdles are then worked on by the designers, but it is important for the users to communicate their requirements accurately.

Process is an important part of any software development project. Processes are important because they force the developer to ensure that the project is completed correctly, by checking that all the necessary tasks are completed, in the right order and to specification.

Milestones and business goals are therefore emphasized, as the client and the developer can work in unison to understand how to deliver a final project within the time constraints necessary. By using a process, the developer and the client know where the goalposts are.

Before selecting the correct process model to develop this application, it is important to define the client parameters. Analysing the case study gives insights into the information that is available to the developers. In this case it is clear that the client as defined is a small scale business which requires a small scale project. A clear understanding of the scope of the project is not available, as the client is expecting the expert to suggest necessary features to define the business scope of the application.

Since the features as required are not listed, it is under the remit of the developer to define what features are required and build a system that fulfils the client requirements in a way that is reasonably clear to the client. Ultimately the client needs to have a clear idea of what the software product he/she is getting contains.

The lack of information in the case study indicates that client requirements are not completely clear, therefore one may regard the earlier, linear sequential software process engineering methods as being unsuitable in this context.

There are several different software process models, which may be split into several categories:

## Linear Sequential Models

The earliest software models are based on a sequence of documents based on the generation of given documents (Royce 1970). One of the main Linear Sequential models is the Waterfall model, which is based on a series of stages, each of which has a document as a deliverable.

The system allows for certainty at every step. Once the analysis is complete, we are certain that the document pertaining to the analysis of the problem is correct. This implies that there cannot be steps back up through the lifecycle, as once the analysis is complete; the design phase begins, and is completely dependent on the analysis phase.

Enriched waterfall models do include feedback from one phase to the next and back, however there is a risk of the project stalling in the requirements specification and analysis phase as the users and the designers keep feeding back into the system, holding the project back.

Another sequential model is the V lifecycle, where testing is planned before any coding starts. A high level design phase is followed by the generation of general system architecture and a final integration test plan. The software product is generated as a number of specific small modules which are tested independently and integrated at a late stage.

In this case, since building a system out of independent components without an easily visible deliverable would not be suitable for a family run business with a fast versioning cycle. The time needed to build full system architecture would be too long for an internet based release cycle.

## Prototyping Models

Prototyping models are a good solution to dealing with requirements that are not completely confirmed. By creating prototypes, clients are able to visualize the software product in their business context, and would therefore be able to understand how the software can help their business process.

There are two main types of prototyping process models, depending on the type of prototype to be generated. Throwaway prototypes would be the focus of discussions between the client and the designer and are refined in these discussions. The system architecture is not contained within these throwaway prototypes. Non-functional mock-ups of screens and report layouts may be used, with the client getting an idea of what the system would look like quickly and inexpensively.

Throwaway prototypes (Smith 1991) could be as simple as quick sketches on a piece of paper or screens generated by simple authoring software such as Macromedia Director or Macromedia Flash, designed to quickly sketch out the interface design. By giving the user an idea of how the system will really work, a clearer understanding of the project may be derived, and the user may suggest new features as required.

Evolutionary prototyping, on the other hand, is more efficient of resources in that the initial prototype is built into the final system. On the other hand, compromises in the early design may remain in the finished system, and code shortcuts designed to build the prototype quickly may remain in the final build of the application.

## RAD

RAD is based on a constant lifecycle of development and feedback, going through the following four steps of requirements planning, where users’ input is sought with a focus on solving business problems, to user design, where UI prototypes are generated, to construction, where code is generated from the prototypes and users validate the system, to final cutover, where the system is tested and introduced. The focus is again on getting the system out as quickly as possible, but testing is handled at the end of the cycle, and quality may be an issue. (Whitten 2003)

### Agile Scrum

These problems are tackled in Agile and Extreme programming methodologies, where the principles of timeboxing and tested code are paramount. With respect to Agile programming, the idea is that an incremental prototype is built, with regular feedback from the client. In an Agile scrum meeting, phases overlap, as the design phase merges into code in a similar way to a rugby team advancing in a scrum. (Schwaber 1996) Regular scrum meetings which include the product owner are organized. The development team are constantly improving the prototype by highlighting what work has been done on the prototype since the last scrum meeting and what tasks are scheduled next for the programming team.

### The Crystal Method

The Crystal clear method (Cockburn 2004), highlights different workflow levels depending on how complex the system to be developed is. A simple (Crystal clear) system would require a Sponsor (the client), a Senior Designer and a Programmer, whereas more complex system would require the implementation of a Crystal Orange process, which would involve an Architect, Sponsor, Business analyst and a Project manager.

### DSDM

Another agile approach that may be of interest in the quest to choose the right process to deal with this case study is the DSDM approach. In DSDM (Rietmann 2001), cost and quality standards are set at the beginning, so the project is guaranteed delivery in a specific time frame with a specific set of resources. There are three main phases in DSDM, with the pre-project phase followed by a project lifecycle phase, which includes a feasibility study, business study and two main iterations which are repeated until the product is ready to be released, but there is a clear focus on feature priority. The first iteration is known as the functional model iteration, and it aims to generate the model on which the system must be built. The second iteration is the design and build iteration, where code is built to reflect the model. This model would primarily be useful where requirements and specification is clearly outlined, and time / resource constraints are tight, however, in this case, the resource constraints are not directly affecting the development of the software in this case study.

### Feature Driven Development

Another agile methodology to be examined is Feature Driven Development. In this case, a high level overall walkthrough of the system is built and the features required are listed. Once the list of features is compiled, planning begins as different programmers take on the different features available. Each feature is developed independently and integrated into the whole of the program. In this case, since the overall walkthrough of the system is not strongly defined, Feature Driven Development does not seem ideal in this context.

### Extreme Programming

The Extreme Programming technique is a widely used agile method. It involves several key practices, including Customer involvement in the development and management of a project, the use of test driven development, pair programming and an emphasis on elegant simplicity and constant small releases, taking advantage of extensive time-boxing to maximize productivity.

An extreme programming project is based around the construction of a metaphor of the business case of the company involved, very similar to the case study as given above. This may be a complex metaphor that would need to be broken down into different stories. Each story is used to estimate the size and complexity of the project, and the number of stories indicates a cost estimate of the amount of hours required to complete the project.

The customer then prioritises which stories are to be developed first, however the main advantage of the user interface stories is that storyboards can be developed which are easily understandable both to the developer and to the client, giving a clear idea of the software to be used to the clients as required.

XP is based on iterations, similar to the other processes reviewed. Programmers select the stories they wish to implement at the Planning phase. In the Doing phase, the customer creates user acceptance tests, which are the final tests that are passed to the developers. Developers then start their work by writing unit tests first. These simple unit tests are used to generate assertions as to what the desired behaviour of the system code is to be. Once the tests are generated, the developer writes the system code. It is generally felt that the time involved writing test code and system code is the same as the time dedicated simply to write the system code.

If requirements change, it is often as simple as writing a new test to meet the new requirement, and code is then refactored to meet that requirement as well. A corollary of the XP paradigm is the fact that pair programming is involved, with one programmer sat at the desk coding with the other programmer thinking about the code written with a view to integrating that code in the overall system. Although pair programming will not be used in this scenario, it is an interesting way of spreading programming knowledge in an enterprise. The test first adage of XP also emphasizes quality and ensures that all the code developed meets the standards as required (James Shore 2007).

The Extreme Programming process model is ideal in this scenario given that a family run business is liable to have loosely defined requirements which will be refined in view of the prototypes produced in the system spec. Since the development project will be very customer focused, a UI metaphor will be developed, and a number of prototypes will be implemented to be checked by the customer.

# Prototype

A horizontal, high fidelity, interactive prototype gives the client an idea of the work flow of the system by showing detailed screens of what the application is going to look like, based on the user interface stories derived:

## Story 1

As a client entering the website, I want to be able to see a list of car categories and select the type of car I like on the front page.

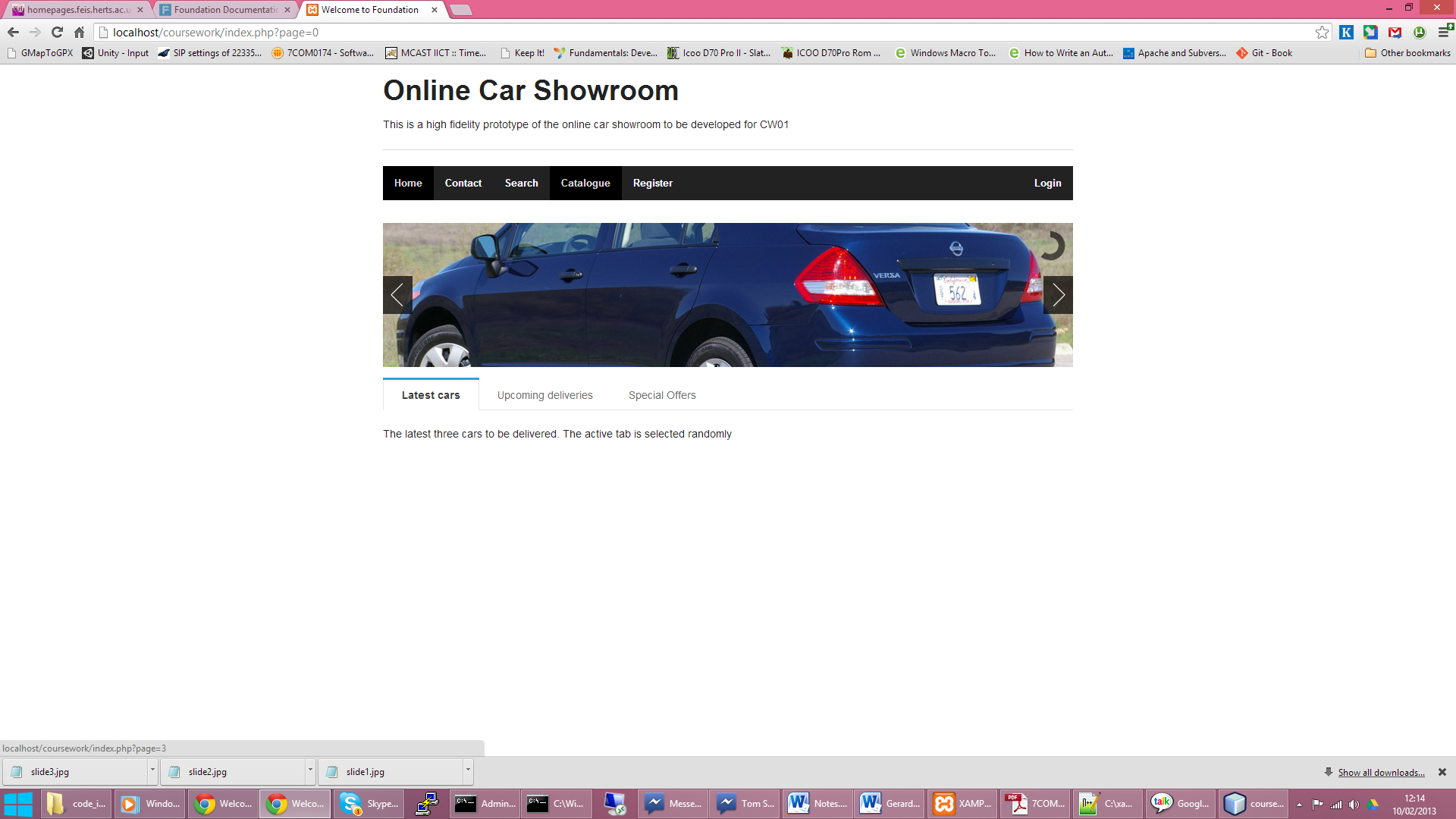


Figure 1

Client selects the ‘catalogue’ page to view a list of car categories in the top navigation bar.

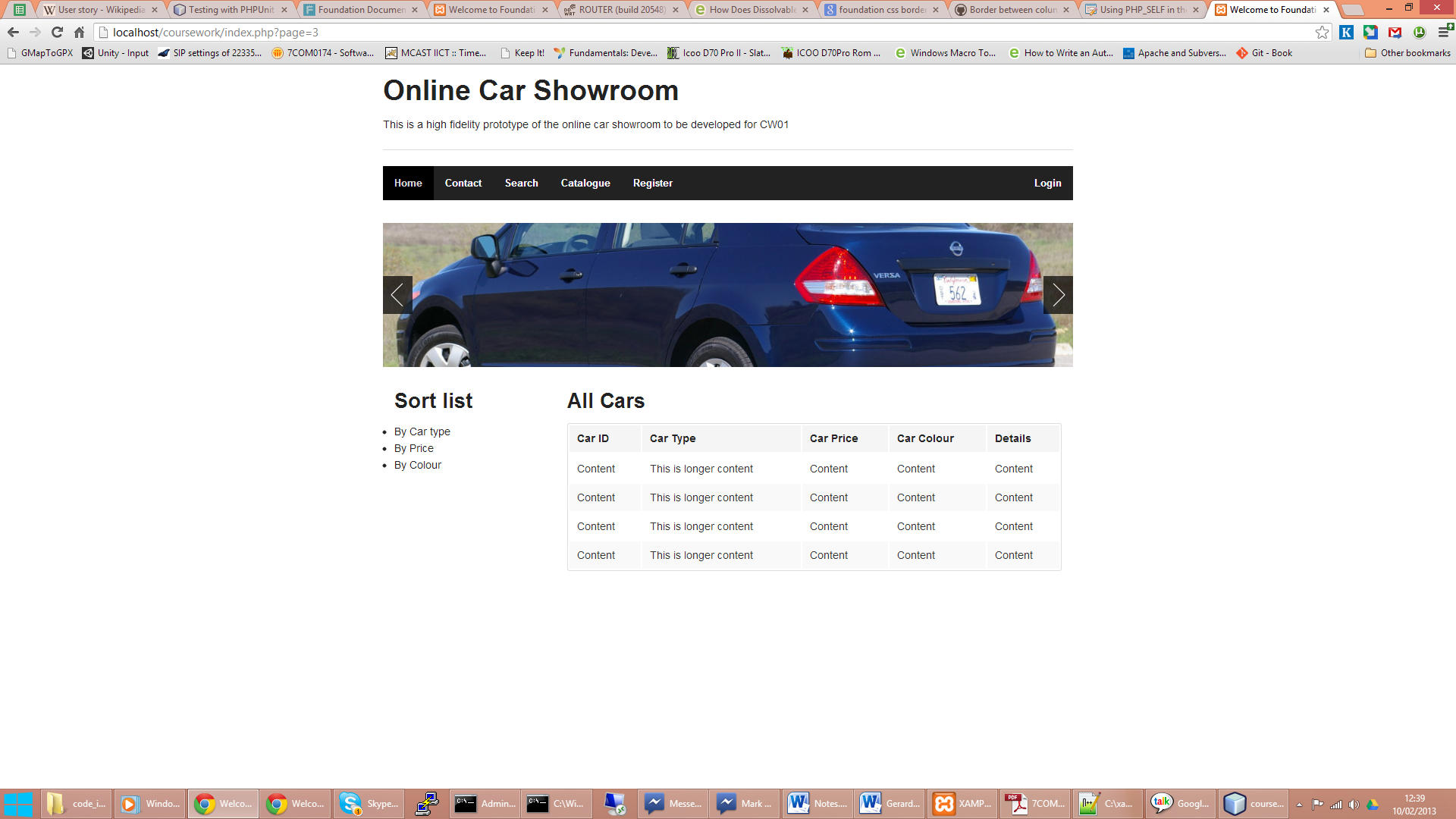


Figure 2

Client can sort list by car category and view the cars as required.

## Story 2

As a client entering the website, I want to be able to search for a car based on the parameters of colour and description.

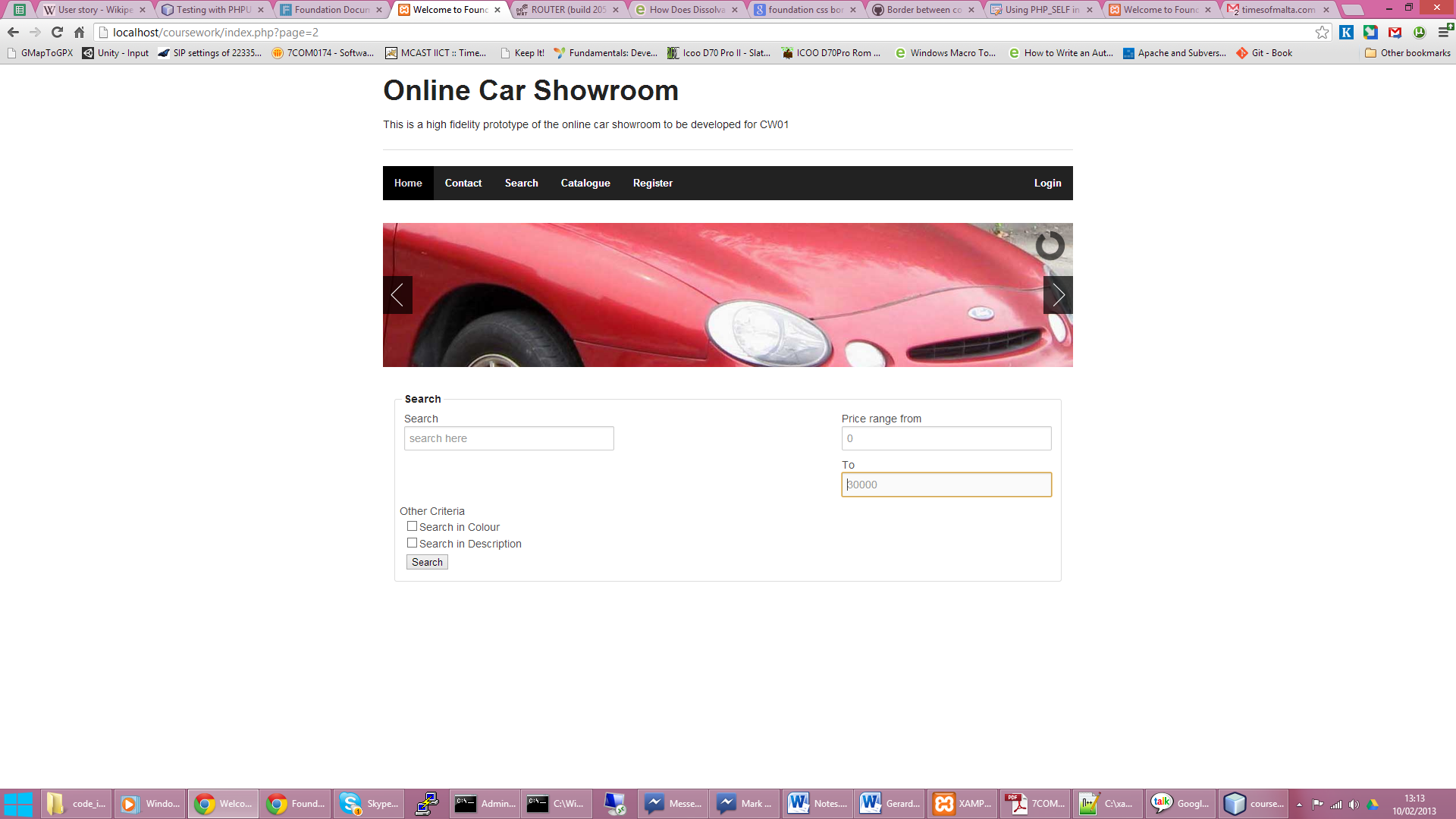


Figure 3

The client can view the search page with the categories as required by clicking ‘search’ in the top navigation bar and bringing up the search page. Results are displayed in a tabular format as above.

## Story 3

As a client, I would like to view multiple pictures of the car in question, as well as download information regarding the car in PDF format to print out or view later.

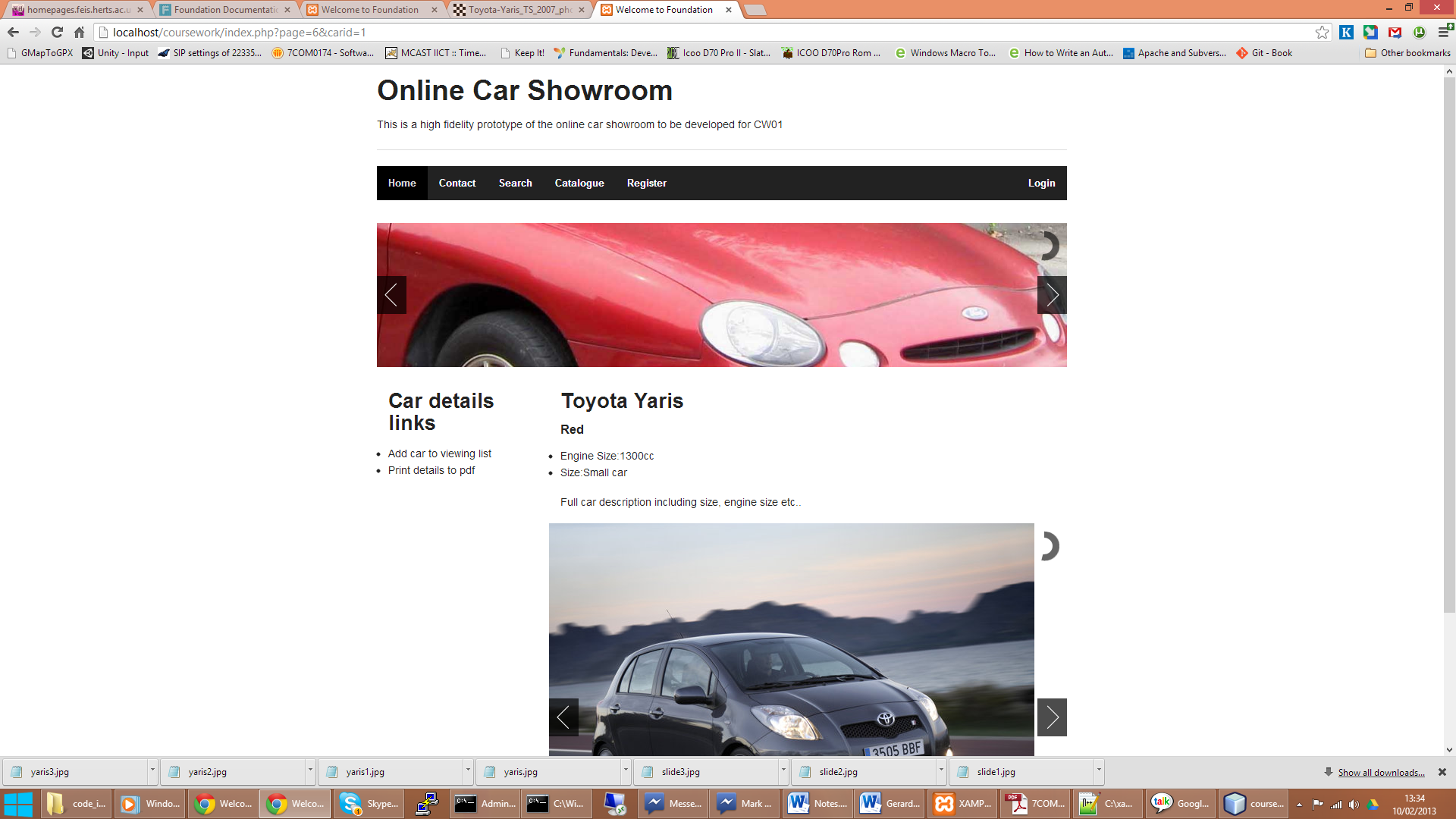


Figure 4

The cars detail screen contains the information about the cars, as well as links on the side of the screen giving the client the option to add the car to his / her favourite list and print details to pdf.

## Story 4

As a client, I would like to generate a number of reports of cars I like which I can print out and bring with me when I come to actually view the cars.

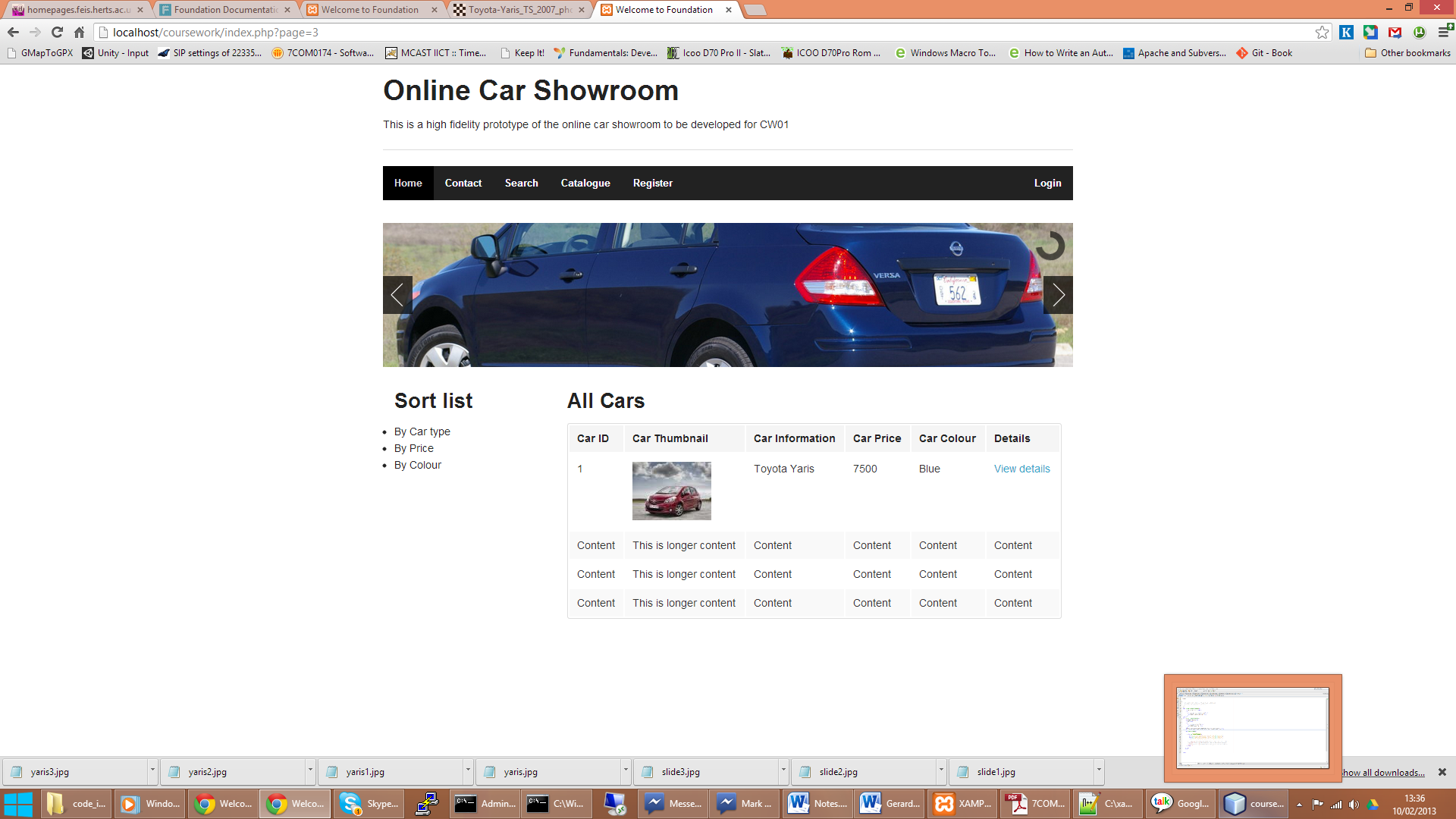


Figure 5

When the client is logged in, he/she is presented with a profile screen which allows him/her to view the list of cars he/she have added to the viewing list

## Story 5

As a client, I would like easy access to all the pertinent servicing information regarding the car that I am viewing. I would also like to be able to save this information for later retrieval.

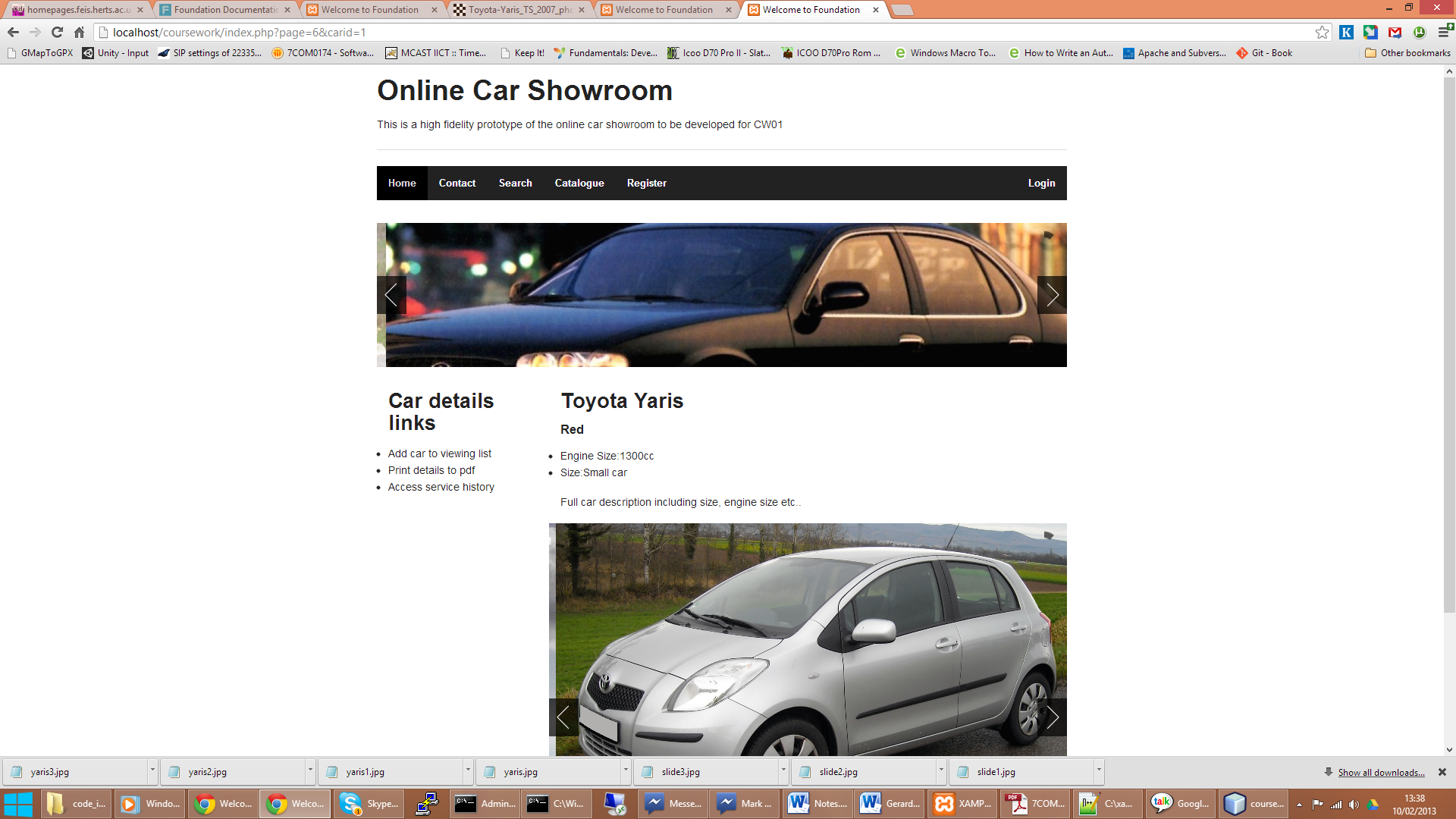


Figure 6

One of the car details links on the left is a link to the car’s full service history. Each car can be added to the client’s favourite cars, as described in a later user interface story.

## Story 6

As a client, I would like to have a login with the ability to save cars I am interested in viewing. In this case, this sequence of events is carried out as follows. The first page is registration, followed by the creation of a login, followed by the creation of a client profile:

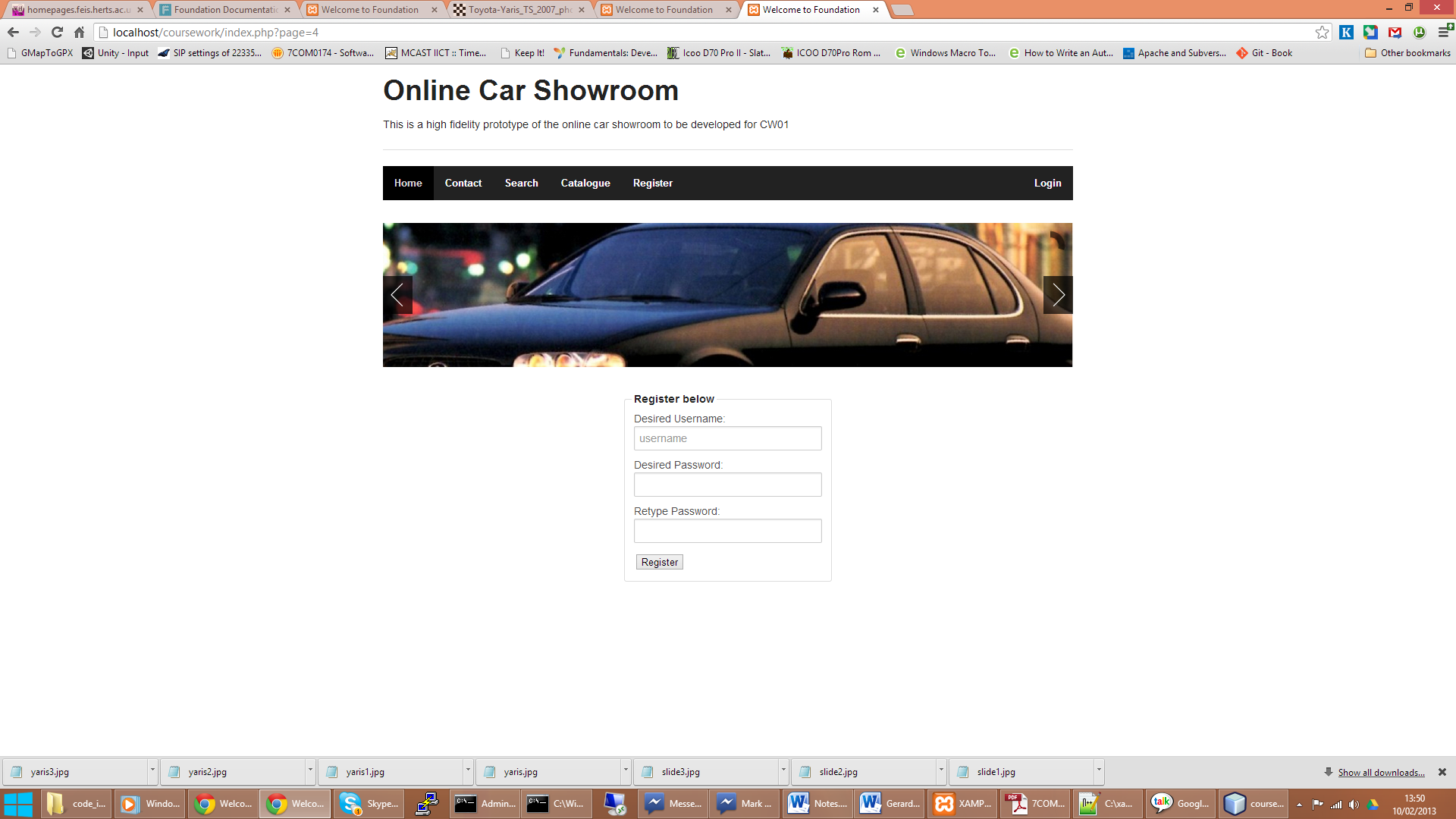


Figure 7

The client registers a username and password within the system. He/she is then taken to a client profile page, and his username is listed as being logged in:

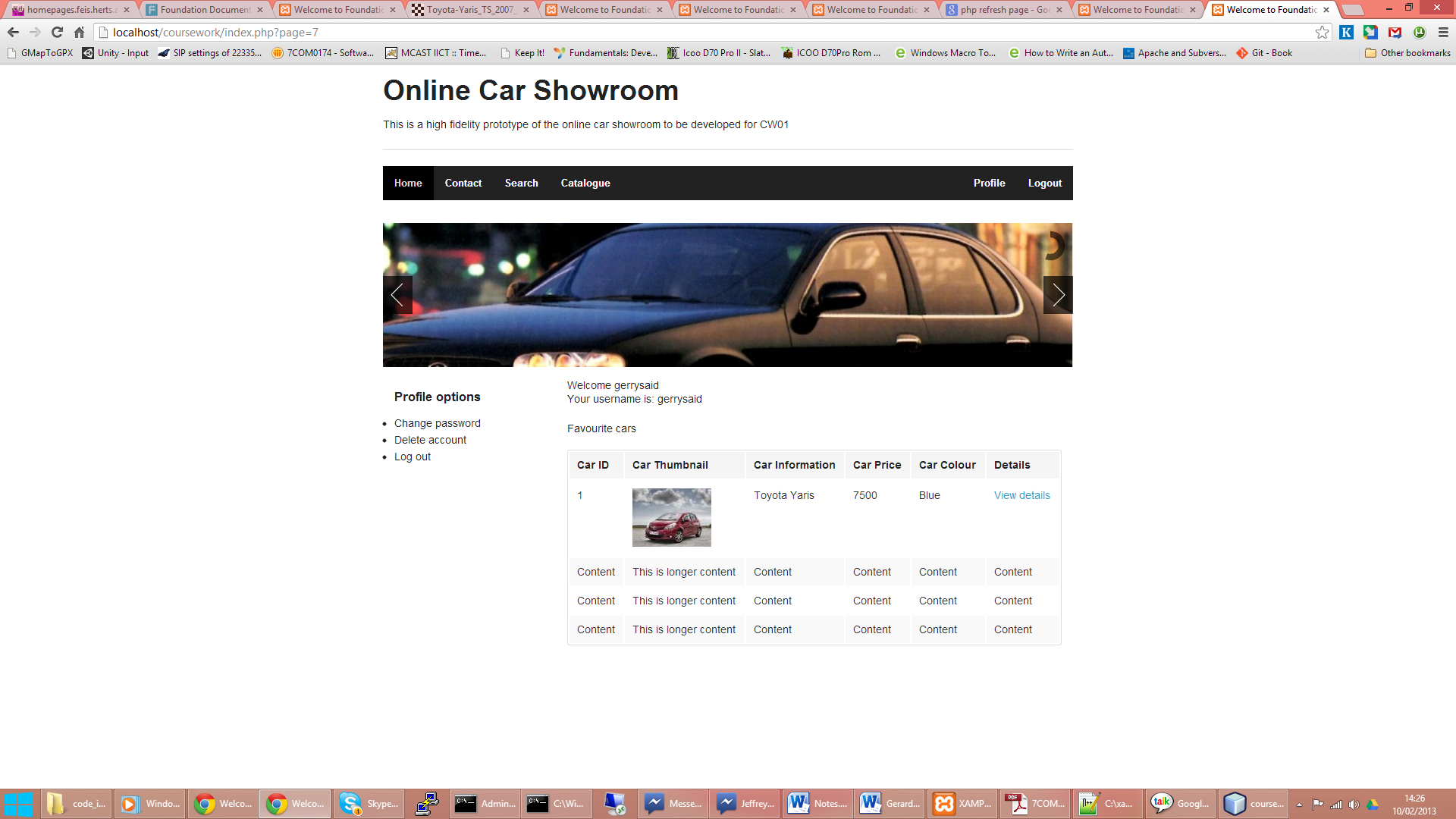


Figure 8

The client profile page, as seen in Figure 8, contains the cars that are listed as favourites by the user in question. The client also has the option to change password and to close his/her account as necessary.

Each client’s favourite list is accessible to the dealership, so the dealer can get an idea of what the client has been looking at on the website.

This favourites list when saved is accessible to the dealership to give the dealership an idea of how many viewings are being prepared for a specific client.

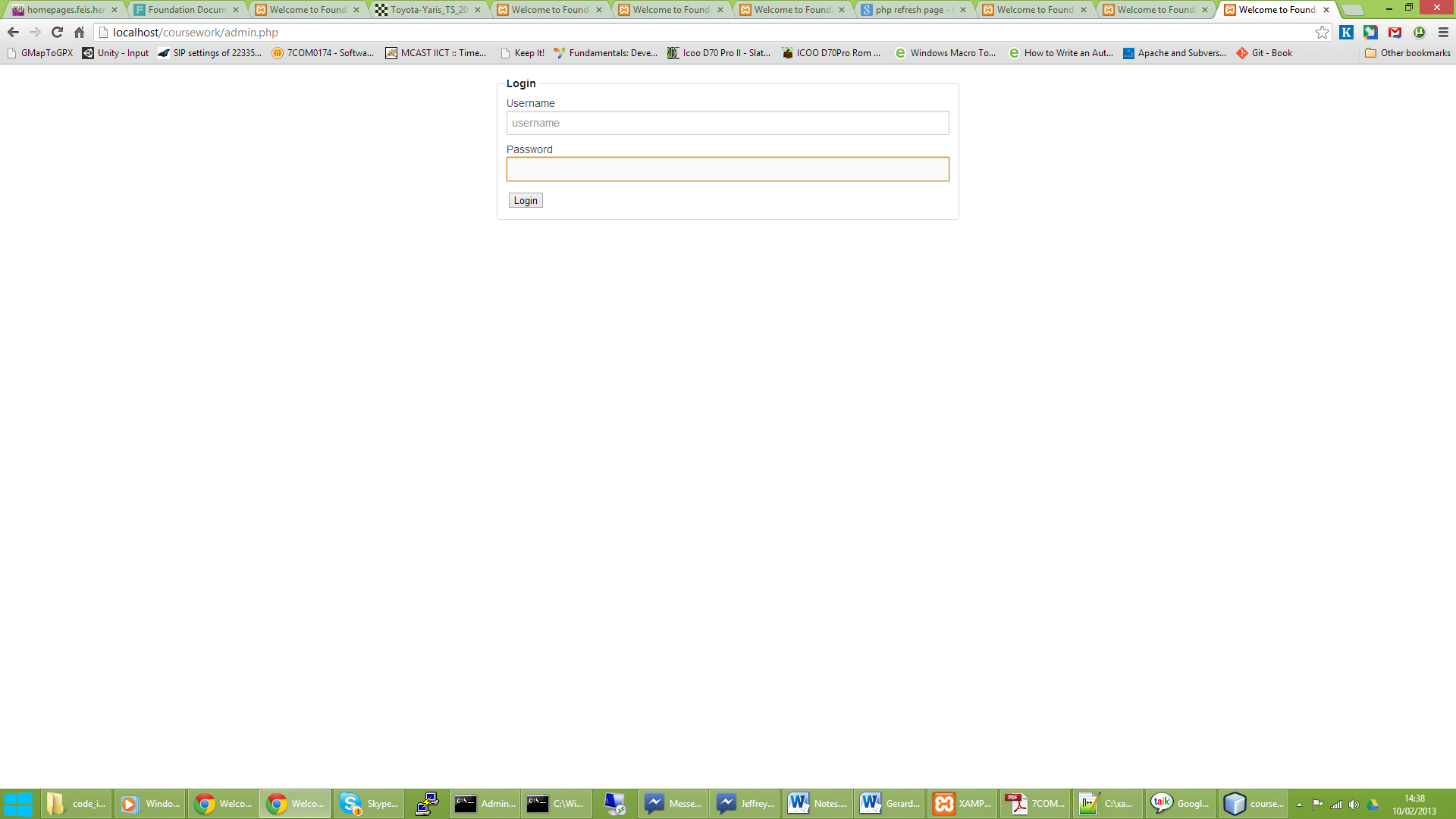


Figure 9

The DMS user logs in using a specific login page. Access is given to the specific DMS users. Once the DMS user logs in, he/she is presented with the following screen:

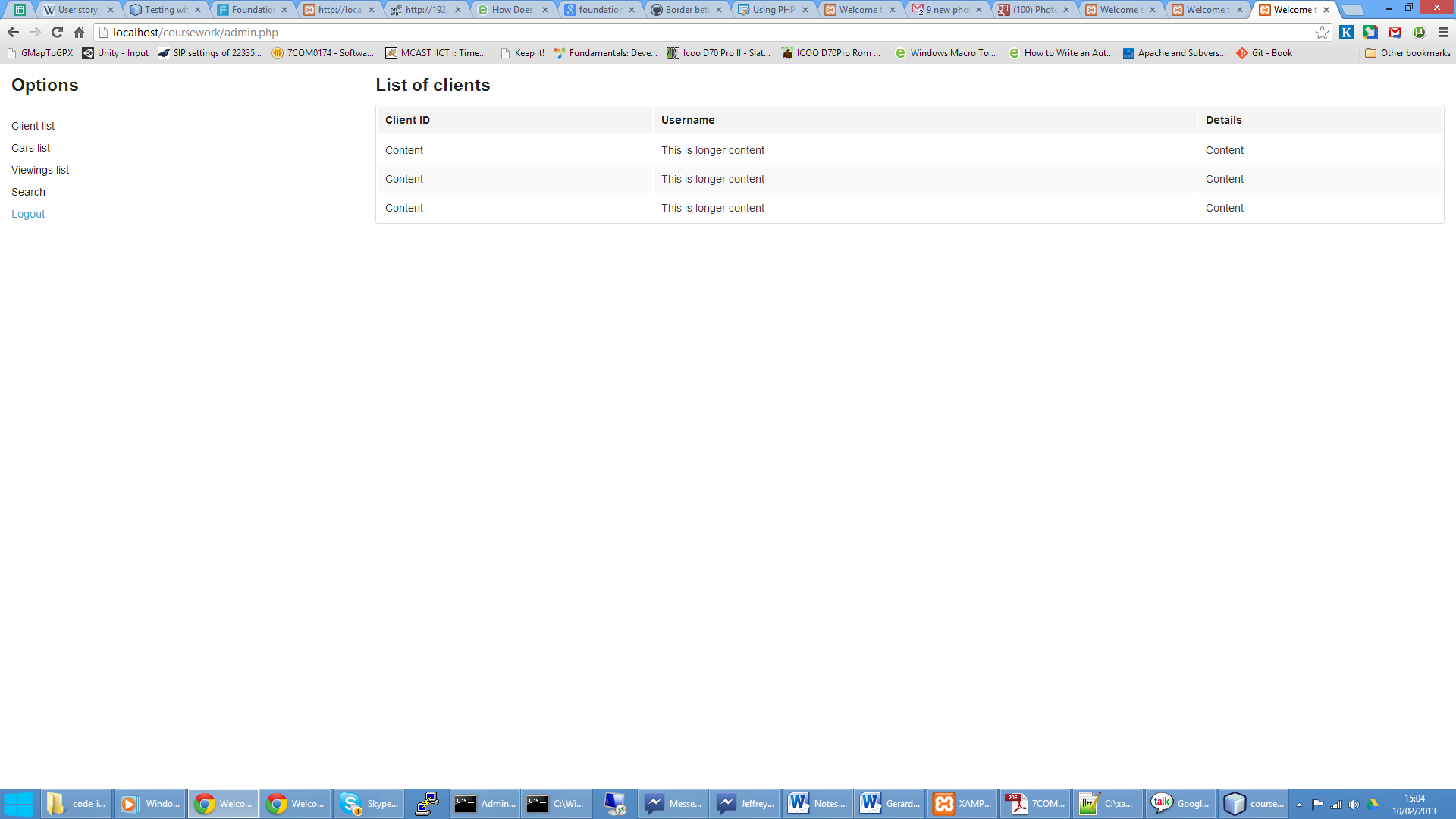


Figure 10

The DMS user can choose to view a list of clients, list of cars, or a list of viewings, with an option to search any of these lists in the search page.

## Story 7

As dealership manager, I would like to have a categorised list of all the cars that I have in my dealership in such a way as to have as many cars as possible displayed.

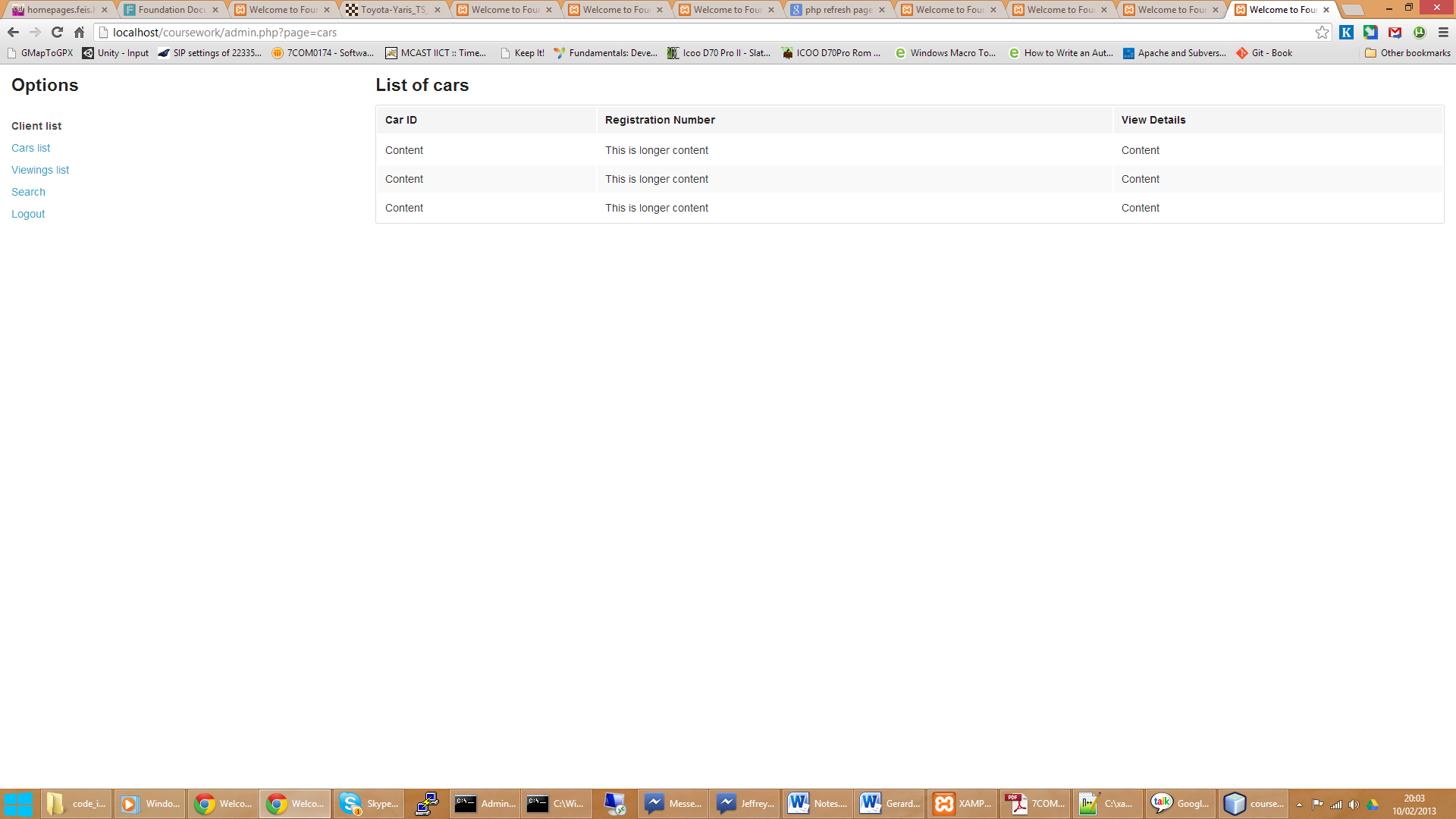


Figure 11

## Story 8

As dealership manager, I would like to search cars by registration number for editing.

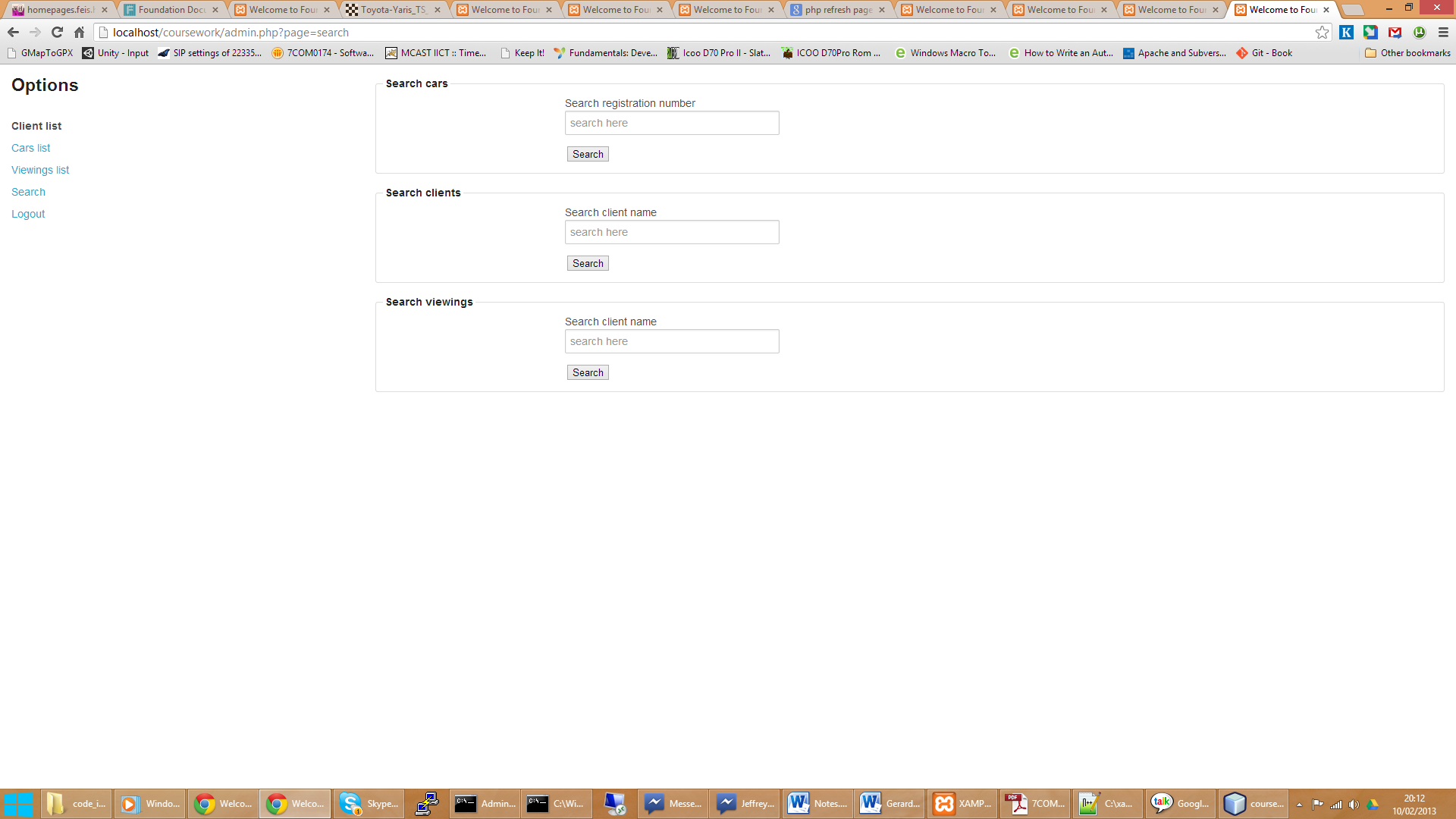


Figure 12

The search option in the admin interface allows for searching multiple criteria with the search results being displayed in a standard grid.

## Story 9

As dealership manager, I would like a simple way of adding multiple cars in an easy sequence, so I can add a number of cars after each other when a new shipment of cars comes in.

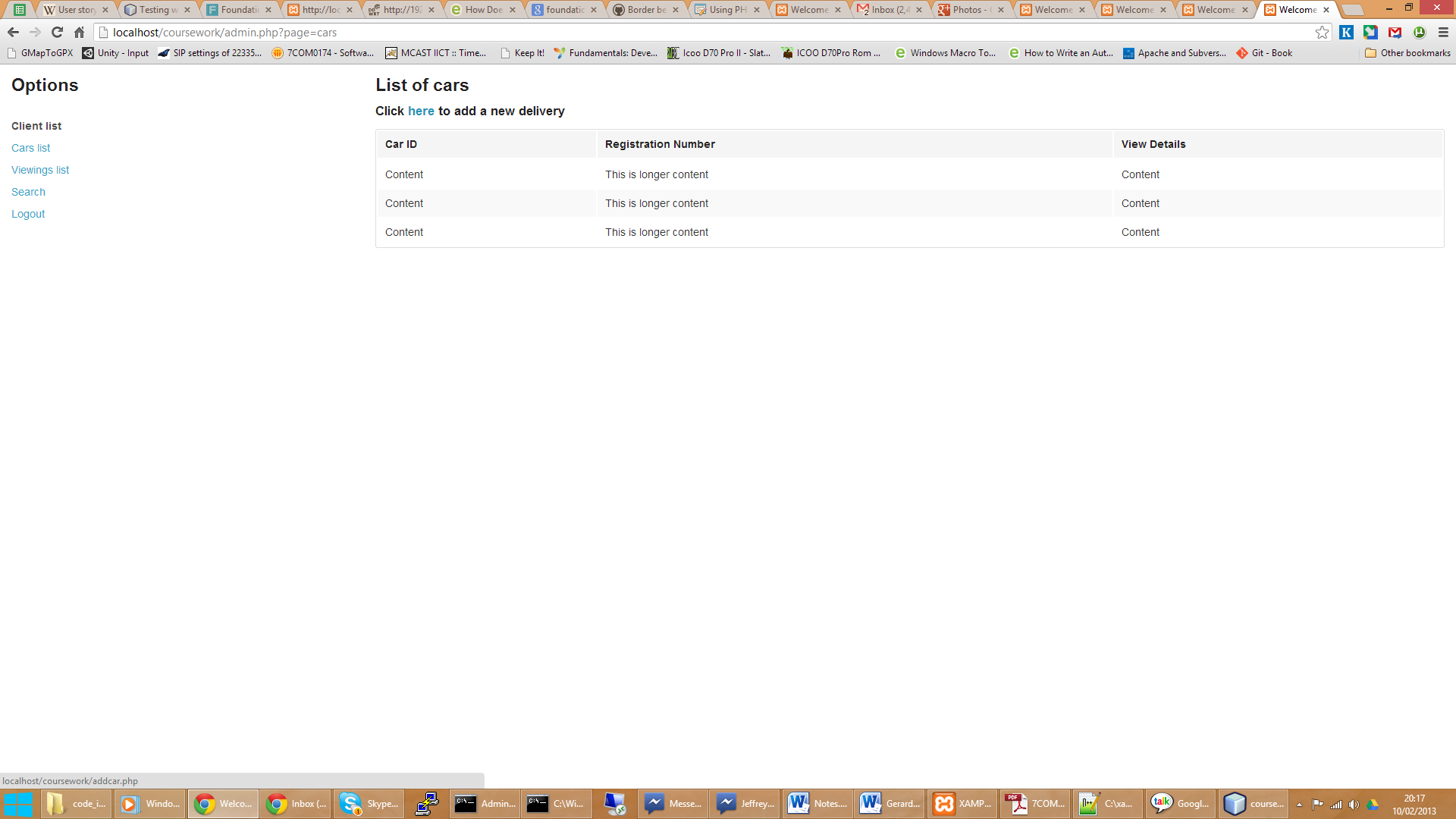


Figure 13

The dealer clicks the list of cars, and can click to add a new delivery.

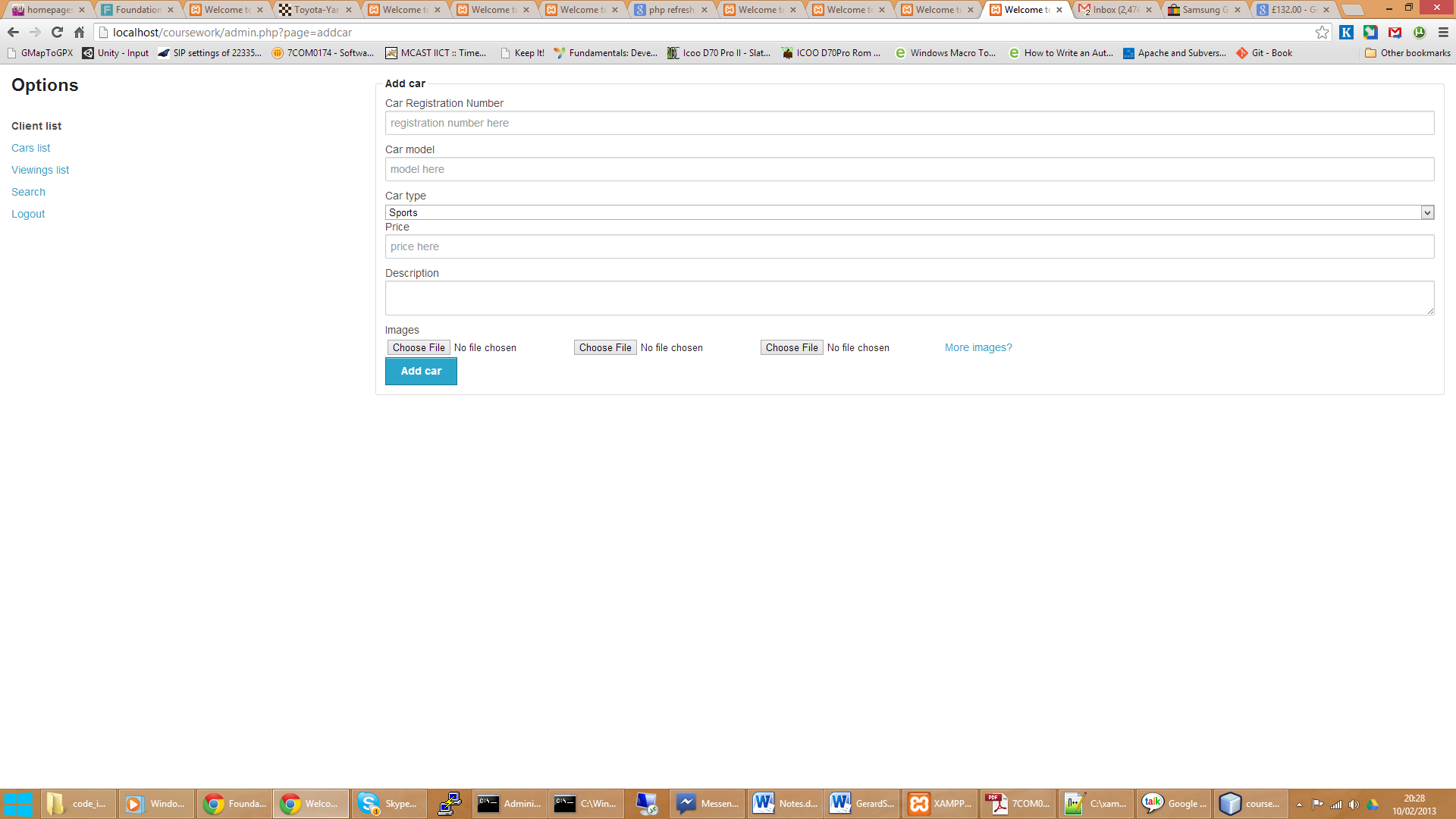


Figure 14

The dealer can add all the car information to add a new car. All of these forms will needless to say be validated.

## Story 10

As dealership manager, I would like a way for my clients to indicate which cars they are interested in before they arrive, so I can prepare them for a viewing.

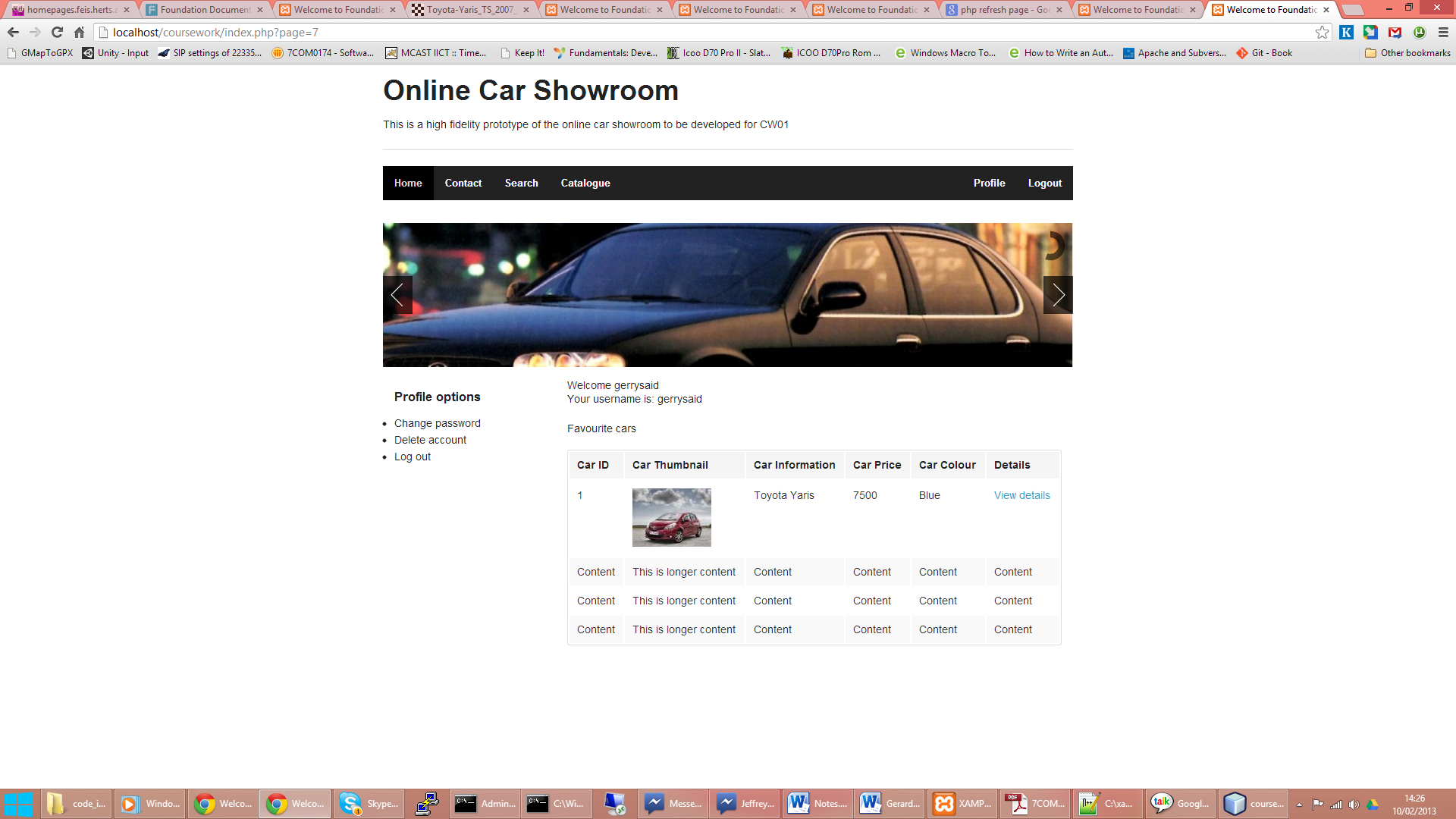


Figure 15

When the client adds the cars to the favourites list, he has the option to book a viewing for all or some of the cars as required by selecting the car in the favourites list and clicking the viewing button under the car details page

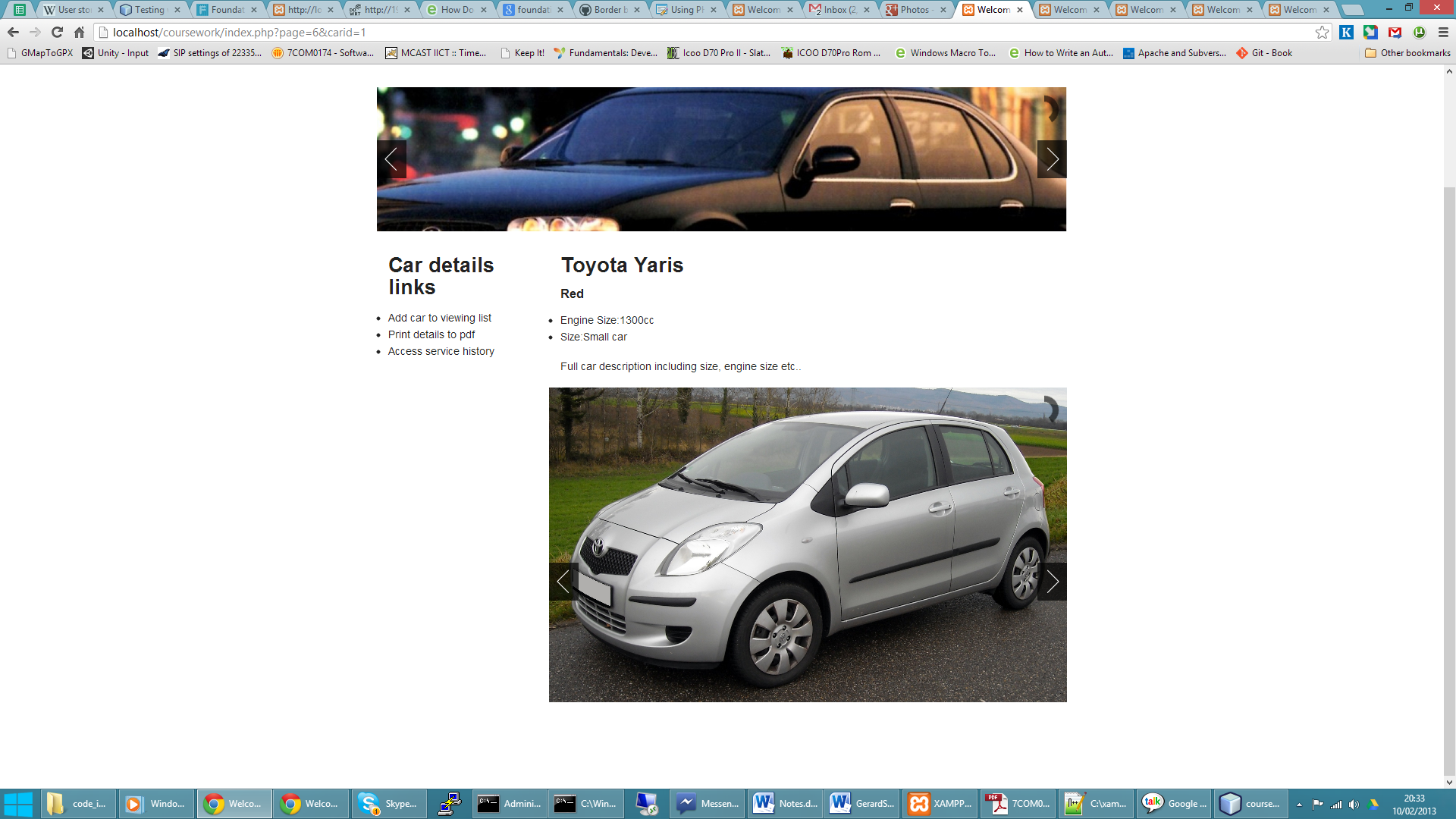


Figure 16

The client can then view his/her viewing list from the profile page

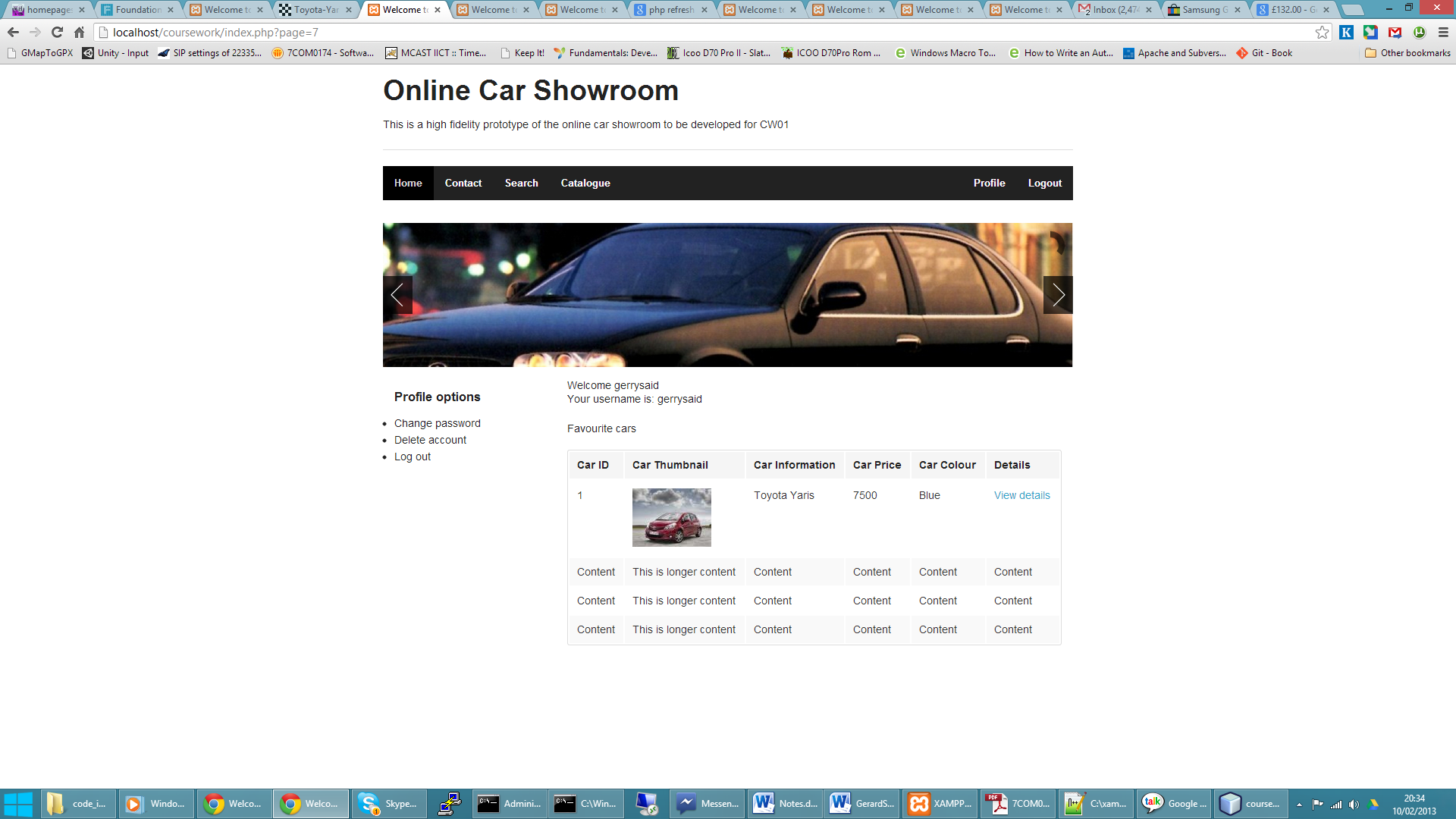


Figure 17

The client can select ‘book viewing’ at the bottom of this page and submit a viewing date for the list of cars required.



Figure 18

The date picker allows the user to select/remove the cars on the viewing list and book a viewing at a convenient date.

## Story 11

As dealership manager, I would like to be able to view which cars have the most requests for viewing to understand which cars are most popular.

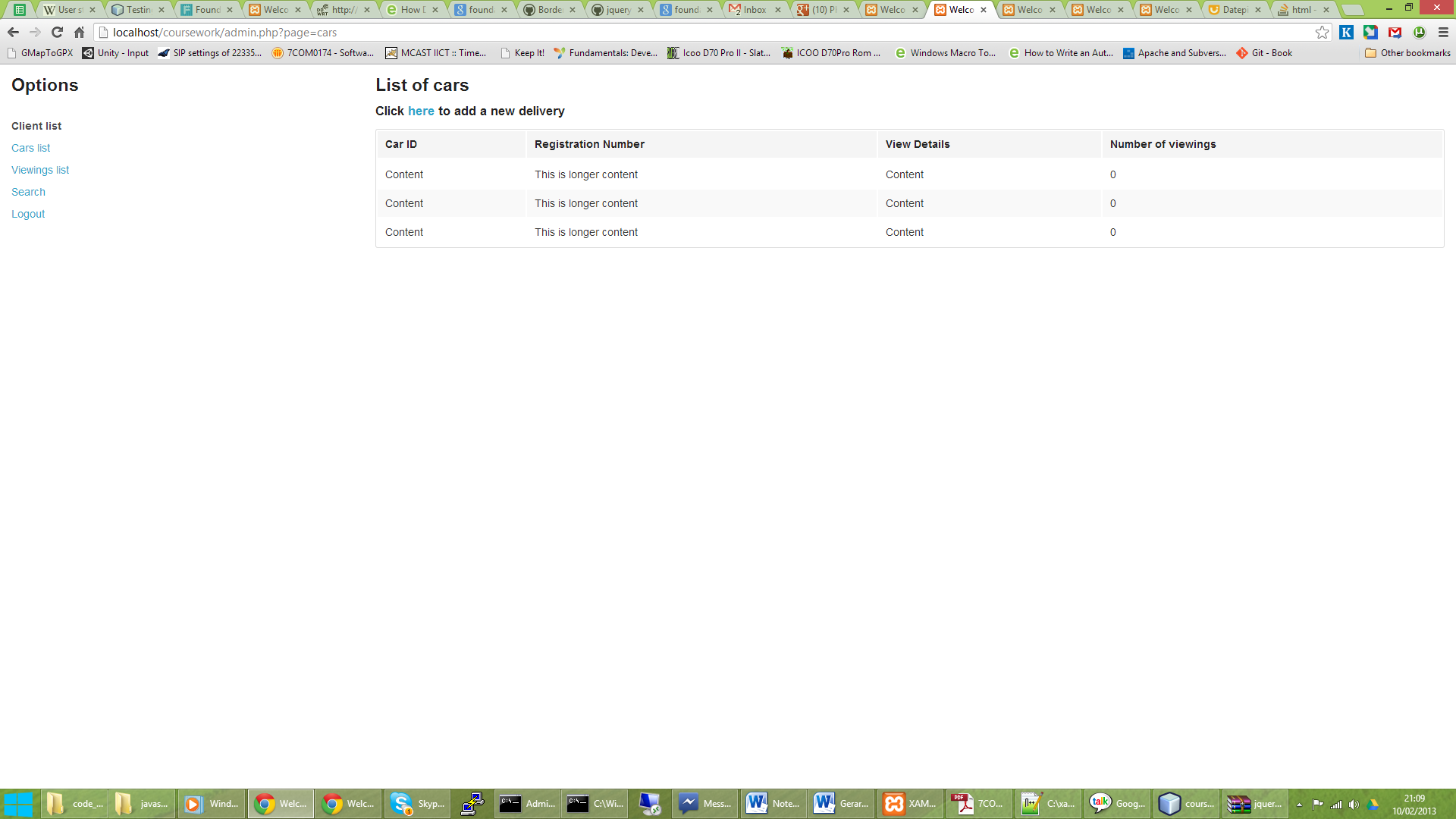


Figure 19

The dealer can examine the most viewed cars by re-sorting the cars list by number of viewings, a piece of data which is available in the cars list screen and is calculated dynamically depending on the number of viewings booked on a specific car ID.

## Story 12

As dealership manager, I would like a simple way of adding multiple photos of different cars in the adding process.

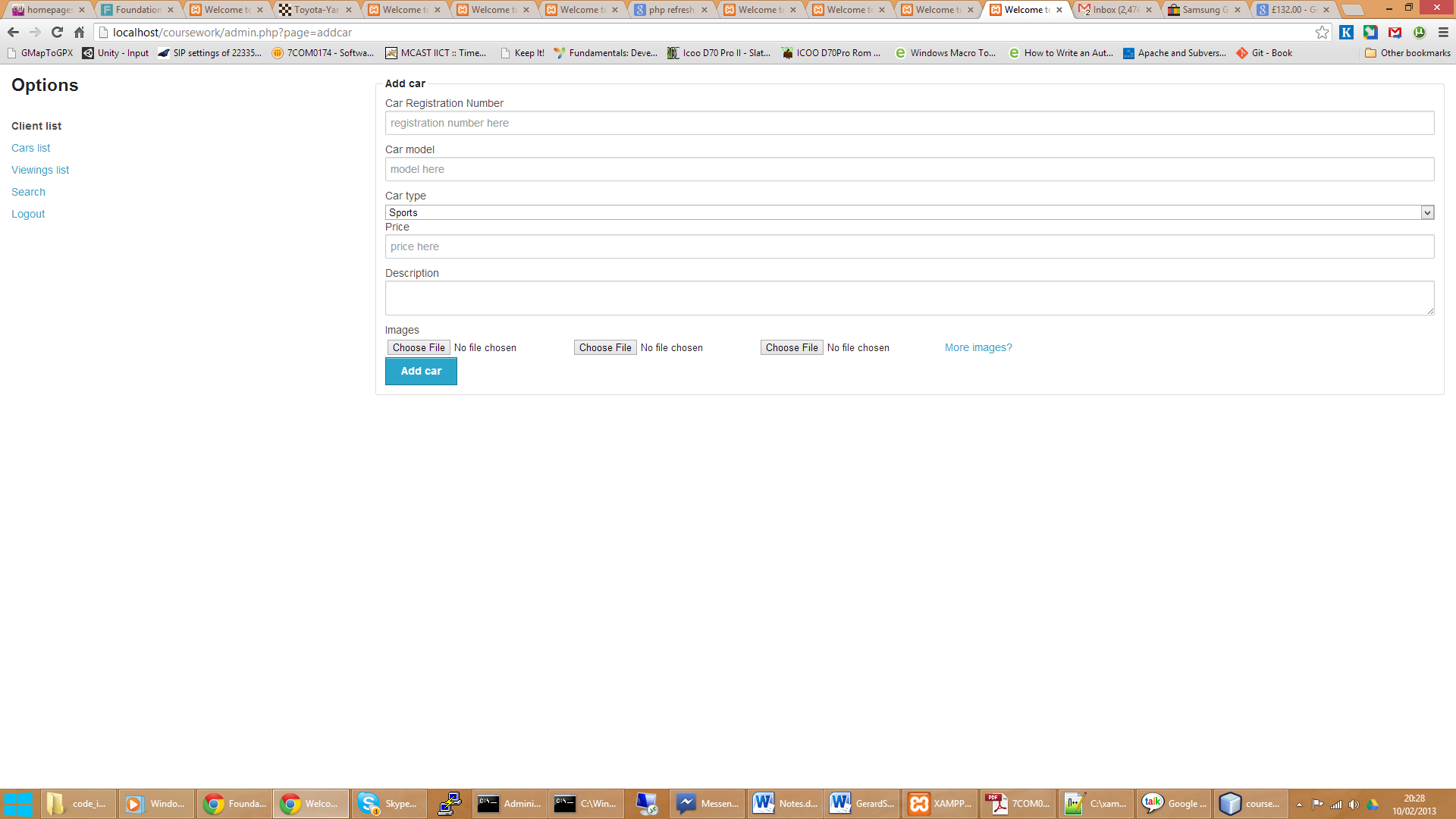


Figure 20

Multiple photos may be added. By clicking the ‘another photo’ link, the dealership manager can add additional photographs to the cars as required.