

Personal portfolio - Eirik Baug

Contributions to release 1:

My contributions to release 1 constitutes of database design and implementation, UI design, framework research suggestion of framework, setting up of the first sprint plan and research on different libraries like Ajax and Vue that allowed med to add some advanced features to the website.

As for the coding in this release I did:

Story 13 – Create a customer account

Story 9 – Editing of customer information

Story 22 – Website home page

Story 20 – Update delivery request form

Story 21 – Special requirements section

Story 23 – Separate website sections

Story 4 – Customer delivery details form

Story 24 – Managing company employees

Story 7 – Delivery request form

Story 8 – Mobile application Functionality

Contributions for release 2:

As for contributions to release 2 I've ensured that the application is mobile scalable in the sections that require such, I've provided meaningful feedback to the developer team and I've gone through the whole code and commented every controller method on the server as well as ensured that the code was properly formatted and structured so that it conforms to the industry standard. In addition, I've written a lot of methods for error handling of forms and database queries and also done a lot of manual testing to check that everything works and fixed any problems I found.

As for coding in this release I did:

Story 1 – Record delivery information

Story 2 – Package information lookup

Story 6 – View daily deliveries

Story 19 – Delivery delay notifications

Story 10 – Package delay notifications

Story 11 – Package delivery status

Story 20 – Update delivery request form (polished, changed and completely finished the one from release 1)

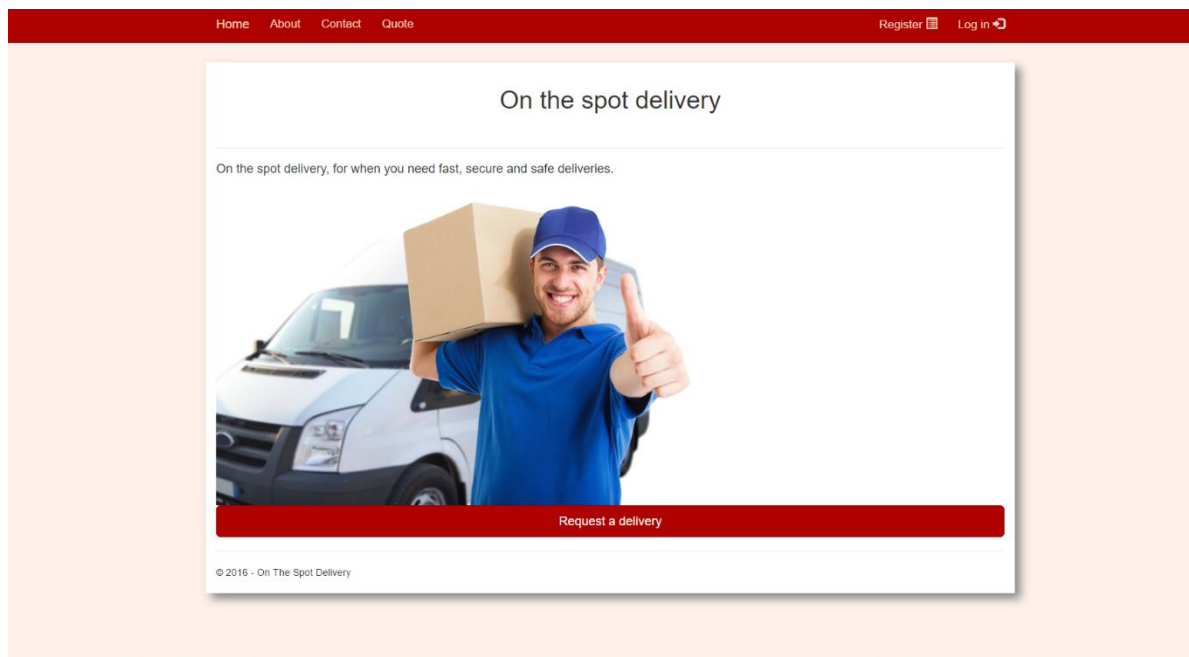
Story 12 – Package deliver cost estimator (Here I only did the part where you transfer the calculated values from the form to the actual delivery request form)

Artefacts release 1:

Artefact 1 release 1 - UI design:

The user interface design is something I put a fair amount of effort into and it's something that I did on the very beginning of release 2. I did some research on how to do the colour combinations and such to make a nice looking website that feels good and warm to look at. I read [this article](#) which states some good practices when choosing colour for your website.

I then chose three distinct main colours that blend nicely together in the application.



Home page of website

As you can see in the picture above the three colours complement each other, and I've also created a uniform layout which is used on every page of the application, where the content

of the page is placed inside the box in the middle of the screen and the box is set to cast shadows to give a bit of a cool 3D effect, making the website welcoming to the user and pleasing to the eye.

I also did choose a bit of a different layout for a couple pages. Mainly pages where scaling to mobile phone was not as necessary, and I was free to implement some more advanced visual features.

The image shows a web application interface with a dark red header. The header contains navigation links: Home, About, Contact, and a user profile section with the email 'owner@qut.edu.au' and a 'Log off' button. The main content area is divided into two parts. On the left, there is a 'Users' section with a search bar and a 'Create employee' button. Below this is a table with columns 'First name' and 'Last name', showing entries like 'Eirik A. Baug', 'Mr. Customer', 'Mr. Employee', and 'Mr. Owner'. On the right, a modal box titled 'Register employee' is open. This modal contains a form with the following fields: First name (Eirik), Last name (Baug), Phone number (97417993), Date of birth (19/10/2016), Car rego (example), Bank account (97417993), Email (eirikbaug@hotmail.com), Street address (Neptunveien, 9), Postcode (4843), Suburb (Arendal), State (Aust-Agder), Password (masked with dots), and Confirm password (masked with dots). A 'Register' button is at the bottom right of the modal.

Register employee modal box in admin panel

Above is an example of how the owner register an employee in the admin panel; by simply pulling up a form in a modal box. This looks and feels a lot better than the standard form which requires a full page refresh to load, and I've added pretty looking CSS animations that displays as you close and open the boxes. All CSS I wrote can be found in the site.css file which I will attach with this submission (can also be found [here](#)). These things are not very friendly to mobile scaling so I've only implemented them where I assume the user will use a computer to access.

[Home](#) [About](#) [Contact](#) [Quote](#) [Admin](#) You are logged in as Owner@qut.edu.au [Log off](#)

Orders

Apply filter:

From:
dd.mm.aaaa -- --

To:
dd.mm.aaaa -- --

Search:
Searchword...

Apply

Package id	Order id	Reciever	
1	1	Bob Bird	<div>Details Sticker</div>
2	2	Jayman	<div>Details Sticker</div>
3	3	Mr. Owner	<div>Details Sticker</div>
4	4	Mr. Owner	<div>Details Sticker</div>

© 2016 - On The Spot Delivery

Order history page with filter

[Home](#) [About](#) [Contact](#) [Quote](#) [Admin](#) You are logged in as Owner@qut.edu.au [Log off](#)

Order

Place an order.

Delivery address:

Street address

Streets 1:122

Postcode

2342

Suburb

Some suburb i don't even know

State

Queensland

Pickup Address:

Street address

Postcode

Suburb

State

Package info:

Weight(kg)

0

Length(mm)

0

Width(mm)

0

Height(mm)

0

Reciever name

Mr. Owner

Priority

Low

Payment type

Cash

Ready for pickup time

dd.mm.aaaa -- --

Special instructions

Place order

© 2016 - On The Spot Delivery

Example of how a normal form is designed on the website when not using modal boxes.

The user interface design makes the website more accessible and welcoming to the user. A good looking front page with a good looking design around it invites business and makes the customer more likely to use your website than a website with a sloppy design. It also makes the website easier to use and navigating around looks good and is easy to do, making both the client team and the customer happy.

I chose this story because it's extremely vital for a business to have a good design, and it really helps business making it an important artefact.

Artefact 2 release 1 - Database

I also created the database for us, modelled it in visual studio and implemented it on a code first basis. Referring to a commit on [4th of September](#) where I committed the database code. The files for the database code is split into 3 parts, [the database initializer](#), [the identity model](#), and [the database tables](#). These all exist under the branch of 4th of September ([initializer](#), [database tables](#) and [identity models](#)) in release 1, but I've included the linked files from the last submission since they have comments to properly explain the methods.

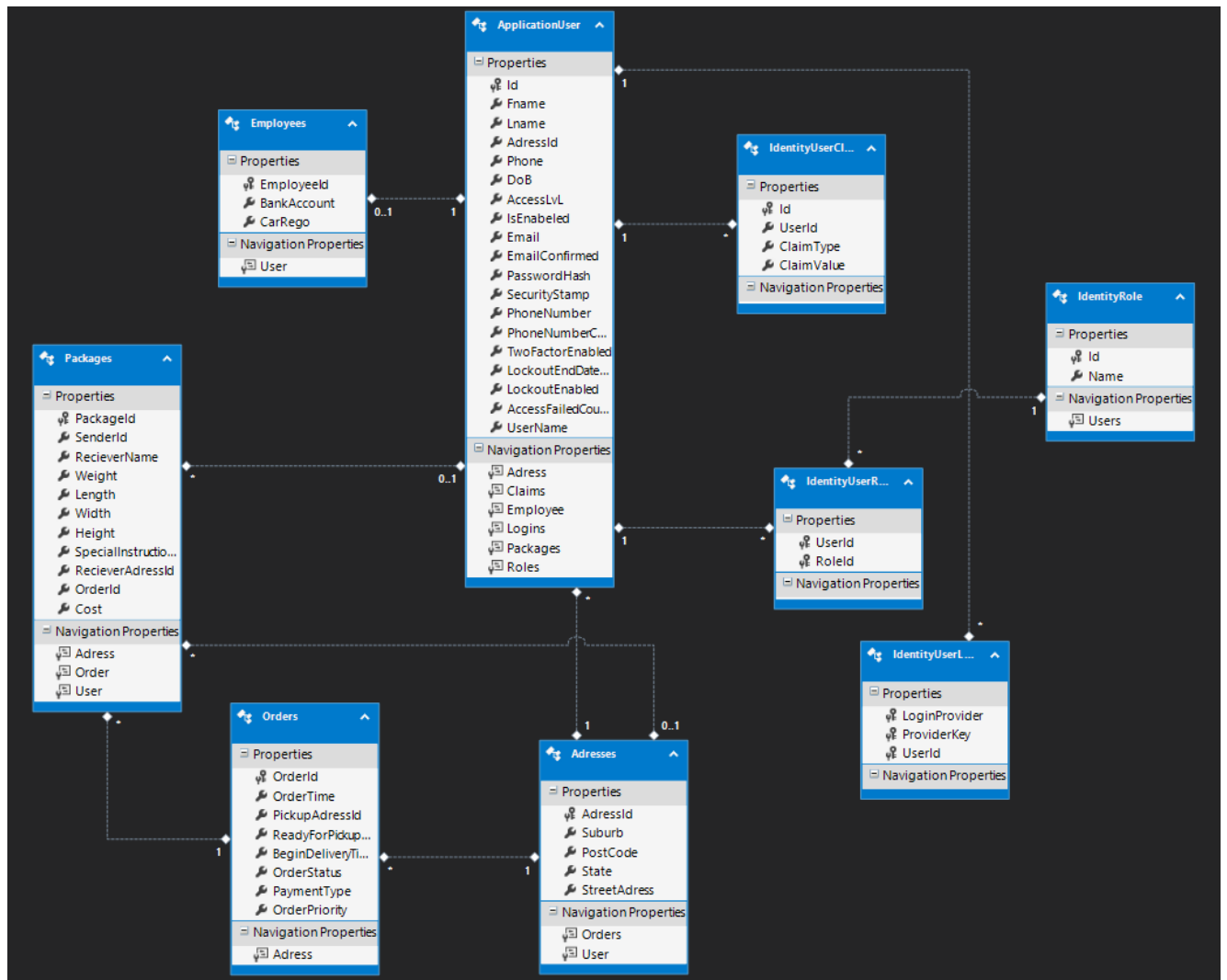
The database initializer is what's initializing the database when it's first created, where a seed method is ran to add test data to the database. In the final submission the database is set to DropIfModelChanges, which means that the database is dropped and seeded anew only if one or more of the database tables change. During development I had this set to DropCreateDatabaseAlways which made the development process easier as the database was dropped and created again every time the code compiles and runs. In the initializer I also added test data like customers, employees, user roles, orders, packages and addresses so that one could easily debug and show the application without having to add everything manually all the time.

In the Identity models file is where I add all the tables to the database and also create the user table with all its values which were not already added by the framework.

The DBtables class is the one where I create all the database tables aside from the user table.

The framework I used for the database was the [Entity framework](#), developed and published by Microsoft. After some research I found that this was the best framework for working with databases in the .NET framework, allowing for easy access to the database from the server. The Entity framework uses a method for querying called [Linq](#) which is an abstractification of SQL, making it really easy to work with and integrate with code.

I chose this artefact because the initializer and the Entity framework speeds up the development process making us more effective as developers and allows us to deliver features faster because the database integration is mostly handled by the framework and the querying of the database is very easy.

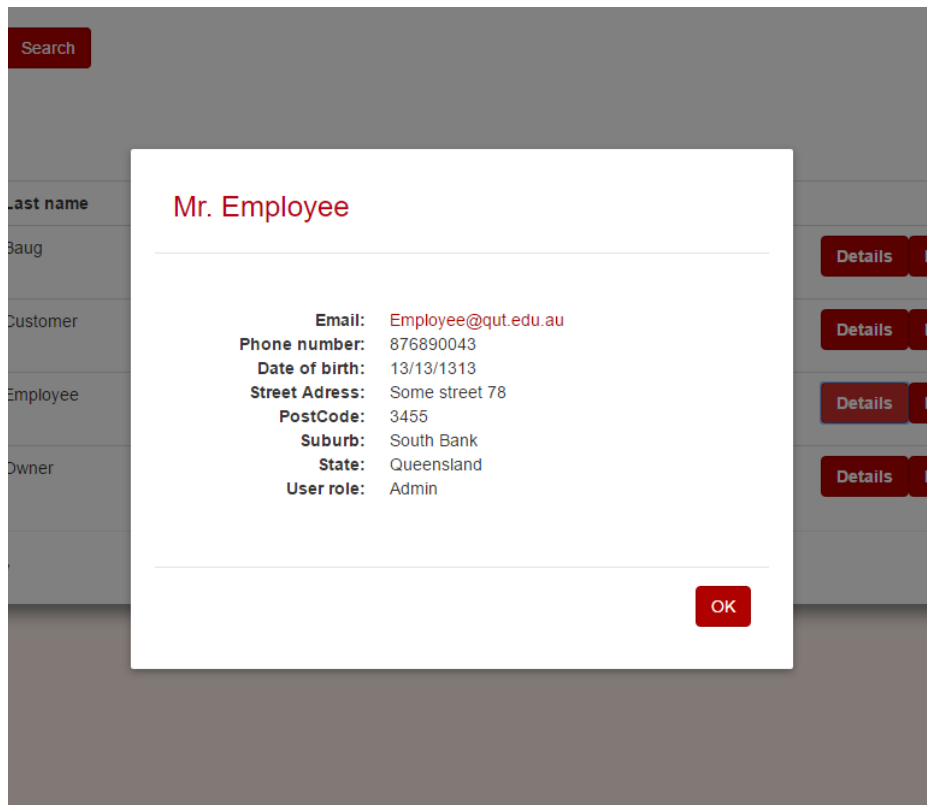


Visual representation of the database used in the application.

Artefact 3, release 1 – The user management panel

Referring to the commit as of [5th of September](#) where I committed the user management page.

The user management panel is one two panels which contain more advanced features for visual display and interaction with the database. Here I researched two libraries, [Ajax](#) and [Vue](#). The ajax library allows me to asynchronously access the database from the browser without actually having to refresh the page to display the information, and with the [.NET framework's partial views](#) I could do an ajax request for information about a user, then update the page with the partial view (which means basically inserting some html with database values on the page you're on) and finally display it with Vue in one of Vue's fancy looking modal boxes.



Example of a details modal box taken from the user management page

As you see above it is a modal box with the user Mr. Employee's profile information. What happens when you click on the details button is that a request is sent to the server to display the modal box. The modal box is then fetched as a partial view, and displayed with the information about the user corresponding to the user of the row you clicked on. All done with Ajax to fetch the modal box and the database values and displayed in a nice manner with Vue and my [custom made CSS](#).

The ajax library allows me to also pull up modal boxes with forms that lets us asynchronously update the user's information and refresh the list all without actually having to refresh the page. When a user is edited correctly and the list is refreshed a green message will appear saying the editing was successful and the edited row will change to green, indicating what was successfully edited. If an error occurs I use the colour red instead and an error message.

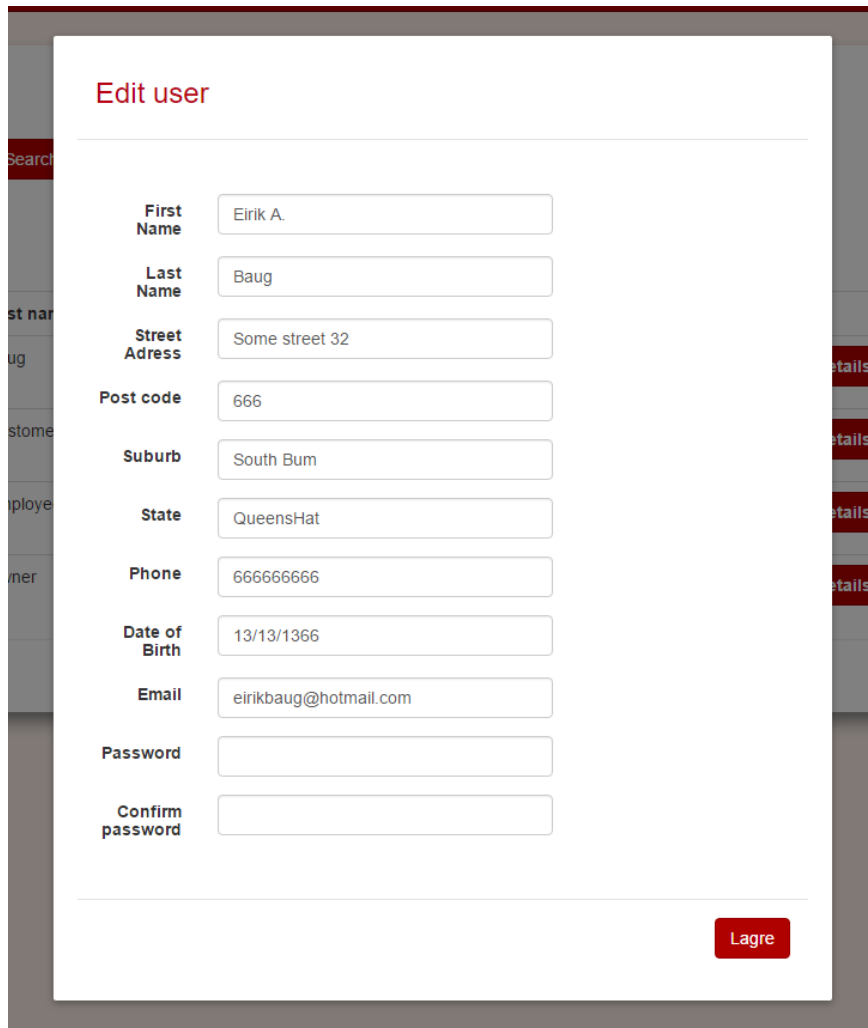
This functionality makes the application much more interactive and easier to use. No need for annoying and time consuming page refreshes, you can just update dynamically which makes the application very pleasing to look at and it feels good to use it, creating a fair amount of business value by making the process more effective, look better and feel better.

As for the code, there is quite a lot since every box modal box has to be its own partial view. All the partial views for this page can be found [here](#).

The controller method is quite long and complicated and can be found [here](#).

As this controller method can be confusing the commented version was added in a later commit and can be found in the final product [here](#).

Moreover, the here are the tutorials I used to learn [Ajax](#) and [Vue](#).

A screenshot of a modal edit box titled "Edit user" in red text. The modal is centered on a dark gray background. It contains a form with the following fields: "First Name" (Eirik A.), "Last Name" (Baug), "Street Address" (Some street 32), "Post code" (666), "Suburb" (South Bum), "State" (QueensHat), "Phone" (666666666), "Date of Birth" (13/13/1366), "Email" (eirikbaug@hotmail.com), "Password", and "Confirm password". A red "Lagre" button is located at the bottom right of the modal.

Example of modal edit box

This artefact is important because it shows some of the more advanced visual features which makes the application so easy and pleasing to use, implementing the required features in a stylish manner.

Artefact 4, release 1 – Research and choosing of framework

We did our development in the .NET MVC 5 framework, which is just one of many frameworks, and at the start we were unsure as to what framework to use. So I did some research figuring out what framework is would best suit our needs. I came up with either node.js with express server – a JavaScript framework, Laravel - a php framework and .NET – a C# based framework. All are frameworks I have worked briefly with in the past and that I knew were good. Different articles [suggesting what to look for](#) in a framework and other articles praising .Net for its strengths like [this](#) article made it so I thought maybe that was the best choice. Of course, the other frameworks had articles like [this](#) and [this](#) which made a compelling case to use those frameworks, but still .Net MVC 5 seemed like the correct choice because of the easy integration with databases and how easy it was to create websites that display information about a specific database item.

Our project is the package delivery one and that means that a lot needs to be saved to a database and a lot needs to be pulled from the database. Since .Net, as mentioned before, has such good integration with the [Entity framework](#) it made these tasks very easy and pleasant to do. The framework also has [scaffolding](#) which allows the user to create automatically generated views, models and controller methods that gets a database item, fills the fields into the model and displays the information on an automatically generated html page, saving us a lot of work when just displaying database information. [Identity framework](#) also allows for easy setup of a login system when you create your application and is also something that's part of the .NET framework, meaning that we don't have to write the difficult methods for securely logging in and creating accounts, we only have to tailor the methods provided by the framework to our needs.

I suggested the framework to my team and they had no problems with it, so that's what we chose.

Looking back, I believe the .NET framework was the correct decision making us effective by using a lot of the help provided by the framework meaning that we didn't have to spend too much time writing similar methods over and over again and use effort to access and retrieve information from the database. This allowed us to complete everything we set out to do, delivering a better product to our client.

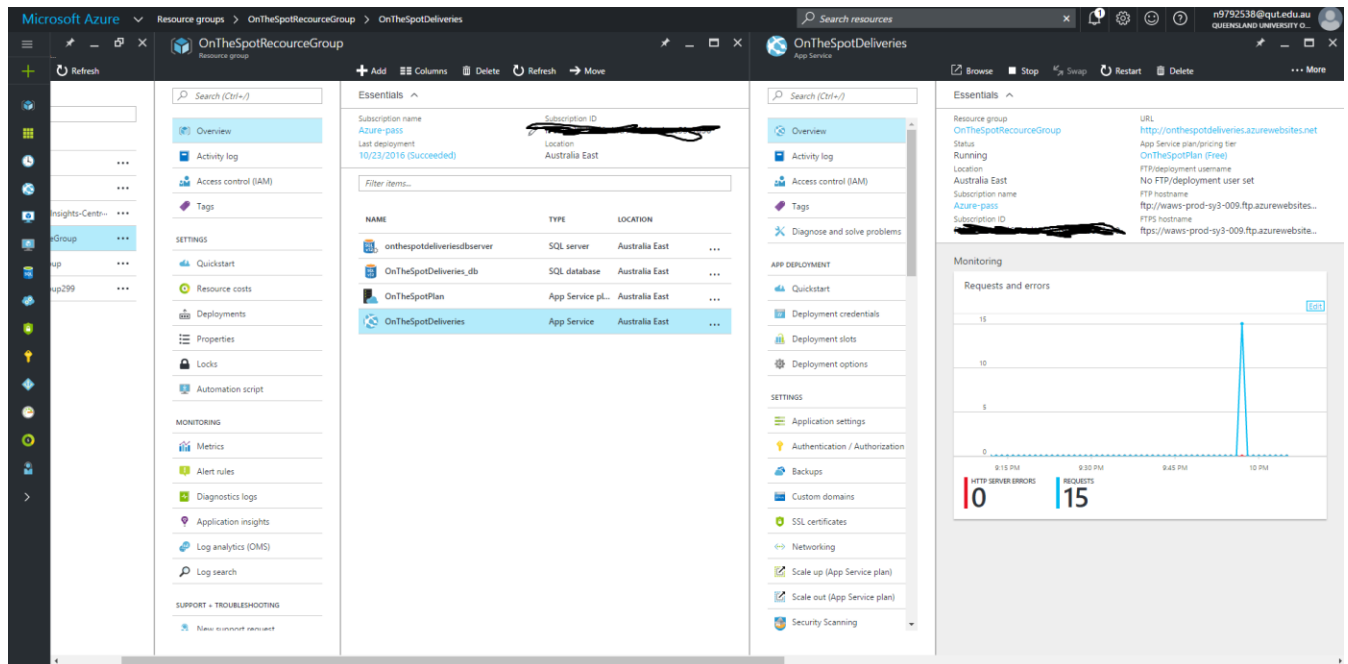
Artefact 5, release 1 – Hosting of website

The website need to be hosted somewhere so that we could display it to the client team during presentation and allow them to test and use the website as much as they wished during the development process. Thus I needed to figure out how to host the website and where. After some research I stumbled upon [this article](#) which suggested I could easily host the website on Microsoft Azure and that visual studio had its own tool for publishing .NET websites via the IDE.

This meant that I need an Azure account since hosting is not free. I got one after talking with Jim Hogan, a lecturer at the university which works a lot with Azure. This allowed me to get a student account with 130 AUD of credit per month, more than enough when we choose the minimum CPU capacity.

I then opened the hosting client in visual studio, followed the steps in the tutorial and successfully hosted the website, which can be accessed through [this link](#). If you wish to test that it works you can log in with Username: "owner@qut.edu.au" and password: "Password1."

Having the website hosted, even though it started creating troubles after a while which I managed to fix before final release, was very vital for the development process and the development of the product. Now we could get more detailed and thorough feedback from the client team allowing us to know what needed our attention more than other things.



This picture shows the dashboard for hosting in Azure, where you can see the application, “OnTheSpotDeliveries”, and also the corresponding SQL server and database. To the far right you can see that the application is up and running.

Artefact is important because it allows us to keep close in touch with the client team’s feedback during the development process.

Artefact 1, release 2 – Git

I have been, throughout the whole project, the one to handle everything with git. The other guys managed to create and push their own branches after I had a little introductory course for them, but aside from that I handled everything. I set up the repository, created the read me and git ignore files and added all my team member to the repository. I also handled all merges of my team member’s branches and solved all merge conflicts that arose.

Since this was an artefact that extended both release 1 and 2 I’ve decided to just put it in release 2 since there after all a lot of the merging was done in the last weeks.

Here is the [link](#) to the repository showing that the repository is managed by my Github account, eiriba14.

Here are 3 examples of merge conflicts I solved and pushed up after merging some branches into master:

[Example 1](#)

[Example 2](#)

[Example 3](#)

This contributed to the project by making the process effective and letting us complete our stories faster, by not having everyone worry about merging and fixing conflicts. The other team members could right away focus on the next task and leave the difficult process of merging to me since I’ve properly researched and know how to fix conflicts and merge

branches. There is a saying which goes; “more cooks spoil the broth” so letting everyone work on merging branches and conflicts is likely not necessary and probably creates more problems than it solves. In short; by leaving it to me I believe the project ran smoother. We used GitBash as our terminal for pushing and creating branches. The merge conflicts themselves were solved by opening the conflicting files in NotePad++ and manually figuring out what was to be kept and what was to be discarded.

This artefact is important because without it we wouldn't really be able to work as a team and deliver the code together, making the project very hard to complete.

Artefact 2, release 2 – Order history page

I created an order history page in the second release which is also scalable to mobile phones.

Referring to a commit as of [6th of October](#) where I added the order history page to the website.

The order history page is a page that allows the user to search through the whole database of orders, completed or not, view details about them and print out a sticker with order information. I will not take credit for the sticker because that I did not do.

When entering the order history page, the list of orders will by default be empty, but if you hit apply without specifying any filter you will display all the orders in the database. You can of course apply a filter which consists of a to and from date field as well as a search word.

Package id	Order id	Receiver	Details	Sticker
1	1	Bob Bird	Details	Sticker
2	2	Jayman	Details	Sticker
3	3	Mr. Owner	Details	Sticker
4	4	Mr. Owner	Details	Sticker

Example of showing all orders from the database (only 4 in this case).

On the server an SQL query is ran based on the request from the view, listing out the orders corresponding to the date and search filter. The queries can be seen in the code from the commit [here](#). The newer commented version can be found [here](#) and the views are [here](#) and

[here](#). In addition, the model used to display the order information can be found [here](#). In my submission it will include only the mentioned controller method from that file.

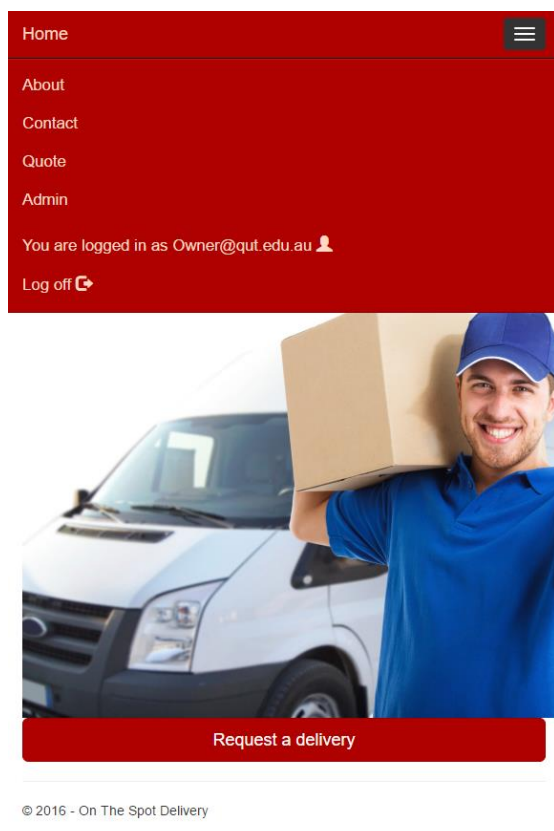
The mentioned controller method is the one called “Orders” and features a number of SQL queries on [Linq](#) format allowing us to extract the relevant database information and reload the page showing what we asked for.

This fit into the project and is important because it was a story requested by our client team and delivered by us in the way they wanted, showing an easy to use order history page which is also scalable to mobile phones which allows for the delivery driver to easily check packages in the database upon request from, for example, the customer.

Artefact 3, Release 2 – Mobile scalability

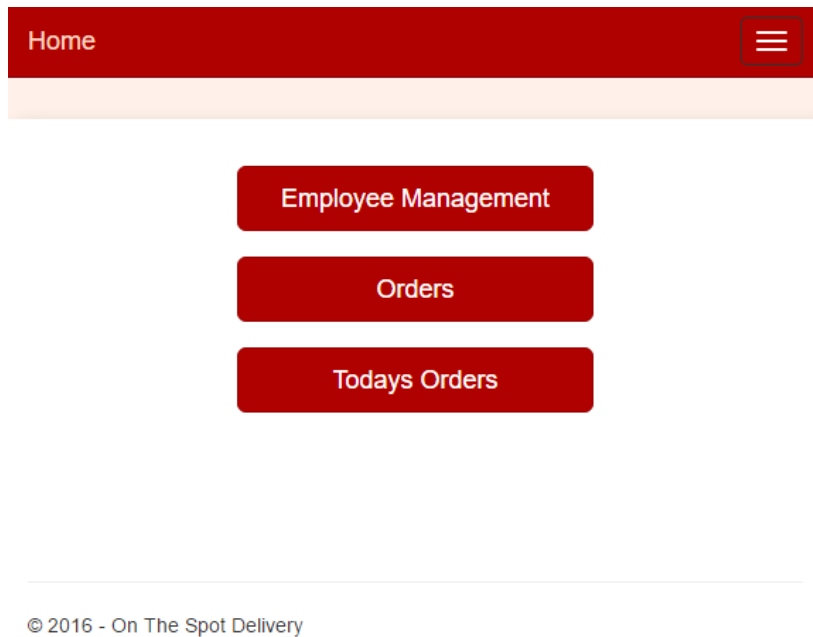
Referring to a number of different commits like [this](#). Creating mobile scalable sites.

This is an artefact which was started in release one and partially completed in that release, at least for the pages it was intended for in the first round, but a lot of it was done in release 2 since that is when I created pages like order history and today’s order’s which according to the client’s wishes absolutely had to be scalable to a mobile phone.

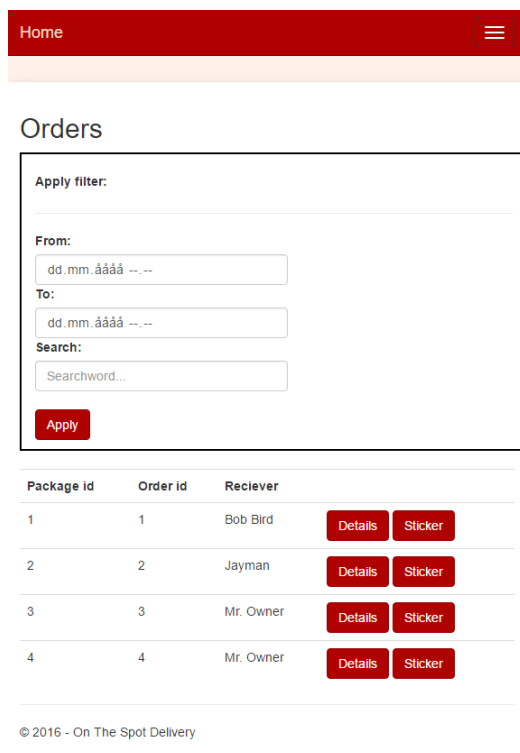


Example of how the application is scaled to a small screen, here displayed with the navigation bar button pressed.

This meant that the application would be easy to use from different devices, and can be done using bootstrap CSS classes creating a modular and dynamic website.



Example of how the application is scaled to a small screen.



Example of how the order history page scales to a smaller device

The website being scalable to a smaller device allows for easy use when the delivery driver is out on the road and only has the phone or tablet with him to start, complete and view

details about deliveries, just as the client team had requested. Some pages are not as well scalable to smaller screens, and those are pages with more advanced visual features allowing the user interaction with the website to be a lot better at the cost of not scaling as well. This is however intended since those pages wasn't meant to be used on a mobile phone anyway (Not a requirement from the client team).

I chose this artefact because it is very vital to a business such as a package delivery service, making delivery almost impossible without some way of accessing the database and website from the phone/tablet while on the road. With this feature the application becomes richer and also feels a lot better to use.

Artefact 4, Release 2 – Code commenting, formatting and standardizing

Referring to the commit as of [October 24th](#).

In the real world the appearance of the code is very important, both for the client team who will be receiving the code and the maintenance of that code in the years to come. It is, over a long period of time, very important that the code is easy to read and understand, because there are likely many different coders who will maintain it, without having originally created the system.

Therefore, I believe this artefact is very important. As I went over and structured the code, creating more business value for the project since the cost of maintaining well-structured and commented code will not be as high. I commented all controller methods that was written by us (did not comment the framework methods) and also every model class we wrote. I refrained from commenting too much in the views as this is not common practice.

I commented inside the controller method on noteworthy things and also create docs above every method explaining what the method does, what the parameters are and what the returns values are. Generally, you can check any controller method in the current master to see examples of docs and code structure, but I can't link to it all as that would be too much. I can however provide a [sample](#).

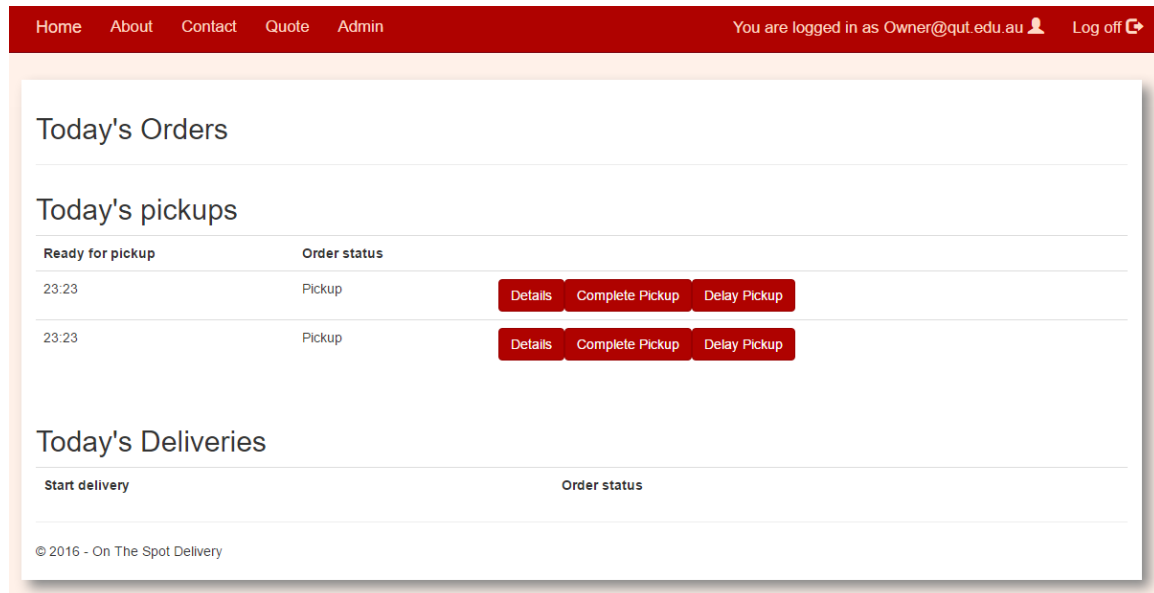
I also researched code structure and formatting to find what was the best practices for doing these things, and found out that [JetBrain's ReShaper](#) was an easy and well developed tool to use for this kind of thing, completely automating the process and conforming the code to industry standard. With this I could download the ReShaper visual studio plugin which was free for students and run the code cleaner to clean up the code, fixing the structure, variable names, the layout and making fields with only need to be "readonly" actually be readonly, writing everything on the [ReShaper standard](#).

With this the project should be easier maintained and understood creating making it easier for the client team to handle in the future.

Artefact 5, Release 2 – Today's orders panel

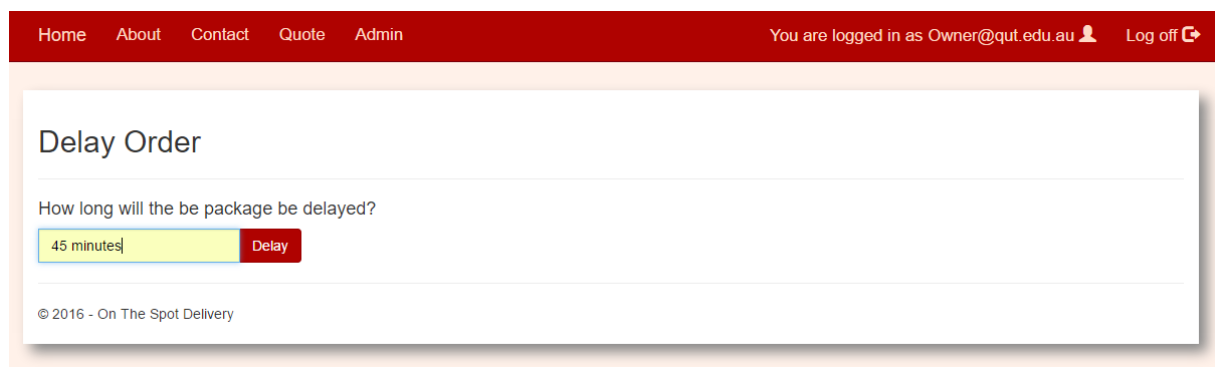
Reffering to the commit as of [5th of October](#).

For the last artefact I chose the Today's order page in the admin panel. This panel is crucial for the process to run properly and allows delivery drivers to pick deliveries and pickups and start them by changing the order status in the database and sending an email notification to the customer notifying them that the package is on the way.



Panel showing pickups and deliveries to do today.

The delivery driver may also use the delay pickup and delay delivery feature which sends an email notifying the user that the package has been delayed by the time input by the delivery driver.



Shows the delivery delayfeature, sending an email to the user about the delay.

noreply@OnTheSpotDelivery.com	Pickup is underway! • A driver picking up you package at Neptunveien is underway Kind regards, On the spot delivery	ma 24.10
noreply@OnTheSpotDelivery.com	Pickup is underway! • A driver picking up you package at Neptunveien is underway Kind regards, On the spot delivery	ma 24.10
noreply@OnTheSpotDelivery.com	Package delayed • Your order being delivered to Some street 32 has been delayed by 45 minutes. Sorry for the inconvenience, On the spot delivery	ma 24.10
noreply@OnTheSpotDelivery.com	Package is underway! • Your package for Some street 32 is underway Kind regards, On the spot delivery	so 23.10
noreply@OnTheSpotDelivery.com	Pickup is underway! • A driver picking up you package at Streets 1:12 is underway Kind regards, On the spot delivery	so 23.10
noreply@OnTheSpotDelivery.com	Package is underway! • A driver picking up you package at Neptunveien is underway Kind regards, On the spot delivery	so 23.10

Image shows notification emails sent from the website.

When the pickup has been completed the package will transfer to the delivery table if it is to be delivered today where the delivery driver can start the delivery and then complete it. The delivery driver can also view all details about the package, like address to pick up, address to deliver, weight, cost and who to deliver to, making the process easy and fluent. When a delivery has been completed it will be removed from the list, but can still be accessed from the order history page.

Code for the controller methods can be found [here](#). Also, here are the views for [order details](#), [delay order](#) and [today's orders](#), as well as the [package model](#) in the Dbtables file.

This page is, as stated before crucial for making the process go around and actually doing so that delivery drivers can find the packages' destination and deliver them safely as well as keeping the user updated on the progress. Without this feature we wouldn't have much of a package delivery service.