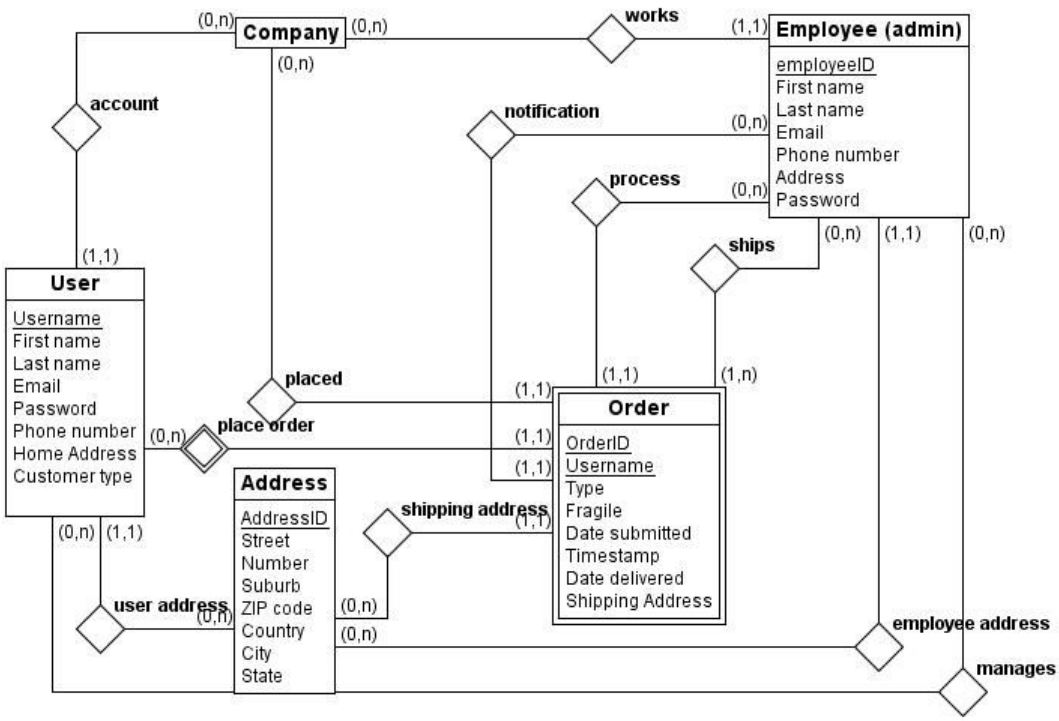
Personal Portfolio

Release 1 - Artifacts

Database design

In the first sprint I helped the group designing the database. I had a lot of objections in how it should be designed. Last year I had a unit in Data modelling, databases and database management, which was very helpful when creating this model. We created an ER-model of the database, which turned out to be pretty good. The model was really helpful when the creation of the actual database started. All relations were defined, and the only changes made from the actual implementation was that Employee-table turned out to be a sub-class of the user table. We did it that because employees and users had a lot in common.

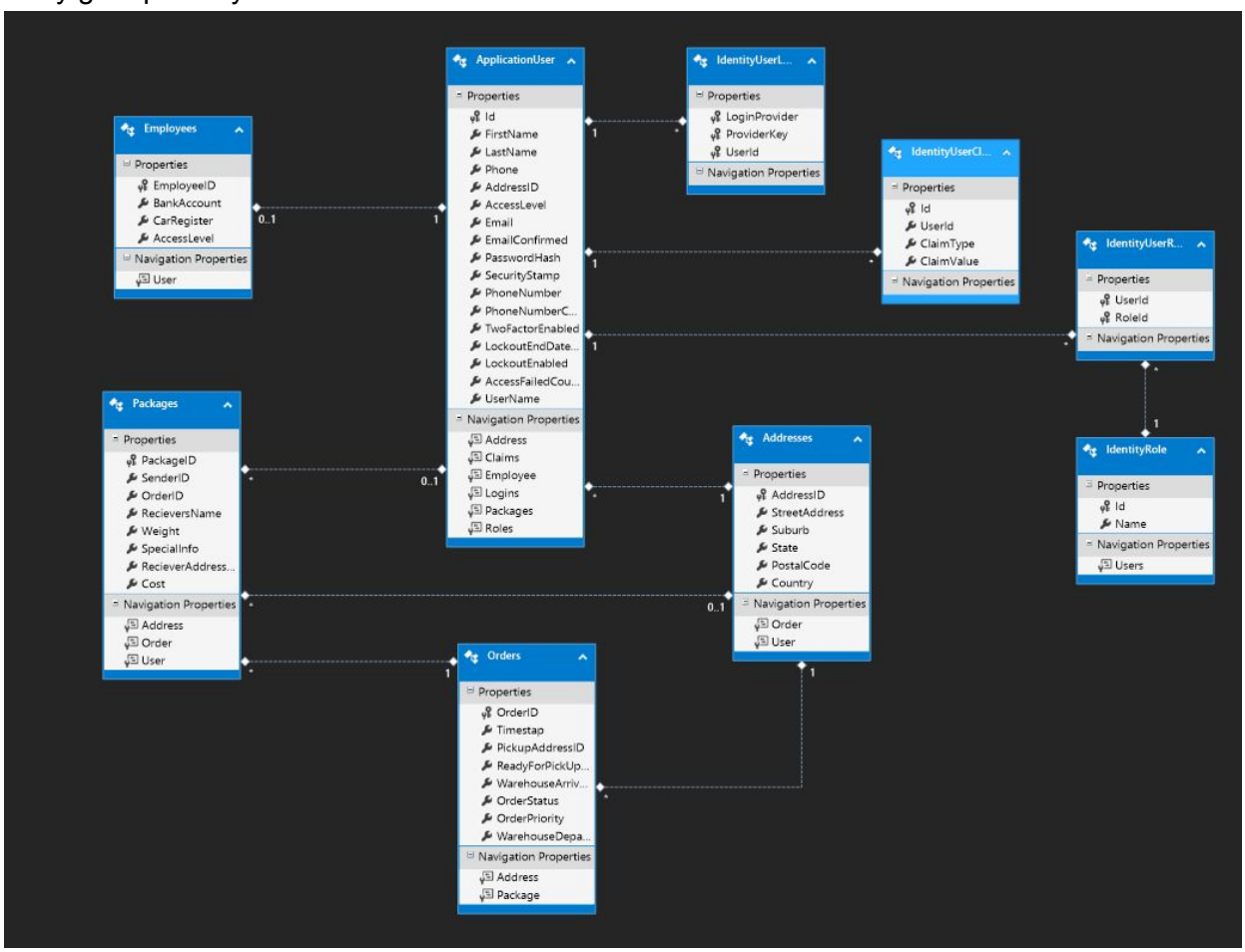
When we developed our model, we used the project description, and started to break it down into smaller pieces. This made it easier to see what kinds of data we needed to store, and how it should be stored. That is how we came up with the different relations, and how they were supposed to fit together. After a while, the entire description was broken into small pieces which led to our data model. By doing it this way, we was fairly sure that we had remember every little detail from the client's problem description.



Database implementation

I ended up implementing most parts of the database. This was initially not my task, it was assigned to a team member that did not contribute to the project. After he stopped answering us in sprint one someone else had to implement the database. The database was crucial for the project, without it would be nearly impossible to do any sensible work. Since the project was based on package delivery, we needed a way to store and organize all our information. I decided to implement the database, using the Entity Framework which is a part of the .NET solutions. By using this I could “Code First Migration”, which means that I could write the entire database in C# as models, and then migrate them to a fully functional relational database. This is an structured way to create a database, and it save you for a lot of SQL-scripting and SQL-syntax errors. It is also safer, considering SQL-injection to the website. All models can be found in our project solution in the files DatabaseTables and IdentityModel.

Final model of how the database turned is shown in the picture below. It has some changes from the original ER-model, but it has all the same key factors. (Bigger picture can be found in my git repository folder.

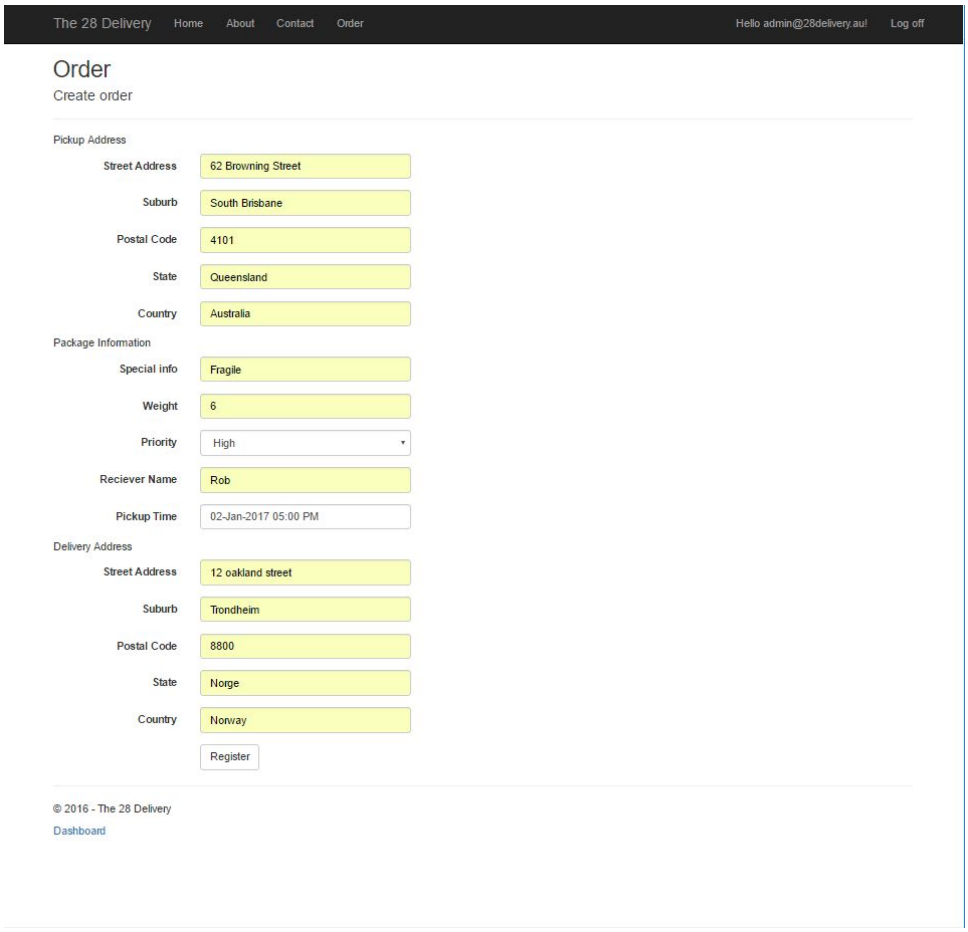


Create and submit orders

I've created and implemented the page where users can place an order on the website. This is an important part of the product, since the company actually deliver packages for people. To deliver a package it is important to first of all be able to create an order for the package that should be delivered. Code can be found in our GitHub repository, in these files: OrderModel, DatabaseTables, Controller , View

As the code clearly shows, the standard MVC model is used to create the page. ASP .NET makes this a to a simple process. I had never done any C# before this project, so that was something I had to learn. Previous experiences with object oriented programming and Java development, made learning C# a lot easier. The page is just a simple form, where users can fill out all information needed for a delivery to be fulfilled.

Picture below shows the actual page. Bigger picture will be in my branch on the Git repo as well.



GitHub

I created our GitHub repository, and I was responsible for teaching the others about Git. By teaching Git, I mean how to create a new branch, add, commit and push work, pull newest version of the source and I tried to teach merging and how to solve conflicts. Since I was the one most comfortable with Git, I mostly handled all commits that were merged with master. I always tried to merge things with the person who had done some work on another branch than the master branch. This way the other team members could learn how to merge two branches together. If conflicts arose during a merge, it was also easier to solve, with the person who had made changes to the files that were conflicting.

Using a distributed version control system (DVCS) like GitHub was crucial for our development. The Git repository helped us sharing source code, and was a perfect tool to backup our work. Without Git, we wouldn't have the same opportunities with code sharing. DVCS is used in nearly every professional companies, and to use that as much as possible will certainly be of great advantage in the future.

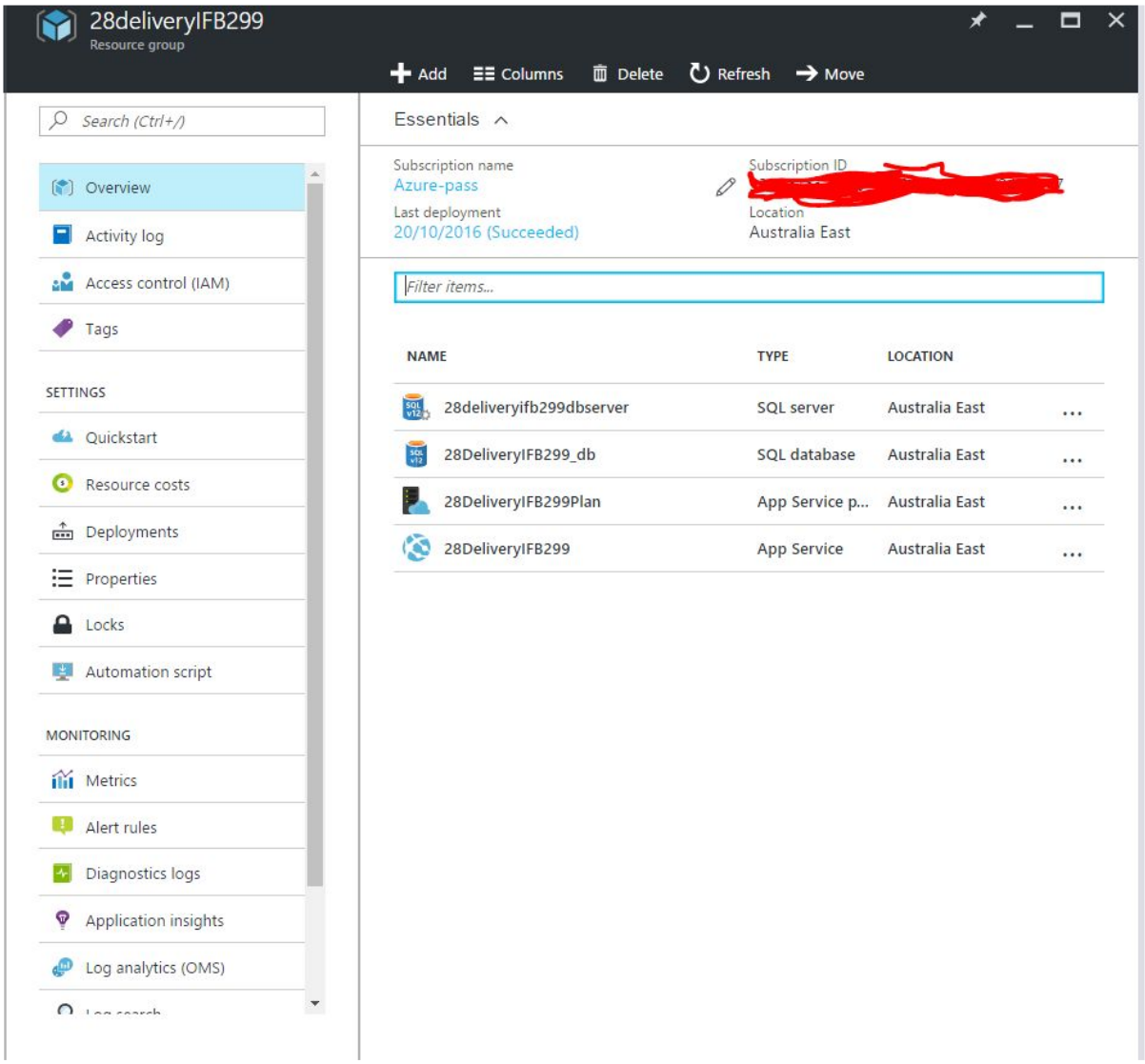
Git repository can be found here.

Deployment of website on server

I was responsible deploying our website. Deployment was done on an Azure server with an Azure SQL database. This was important for actually being able to "hand in" our product. This was done so the client didn't have to clone the repository and install programs on his personal computer to check it. To have the website online also was a great way to confirm that our product would work in production, something our client team appreciated. The webpage are currently hosted at http://28deliveryifb299.azurewebsites.net/. The domain has been changed a couple of times during the project, that is simply because Azure has free trials on their databases, and I had to create new accounts every now and then, so I wouldn't get any bills.

I chose to use Azure for two reasons. Number one was that Visual Studio has great integration of Azure's services, which made the whole process of deploying the webiste easy. The second reason for choosing Azure is that they are one of the biggest "player" in this field, and it will therefore (most likely) be reliable.

Picture below shows the current server, database and app service running in the Azure portal.Bigger picture will also be provided in my personal Git repository branch.


28deliveryIFB299
Resource group
Add
Columns
Delete
Refresh
Move
Search (Ctrl+/)
Overview
Activity log
Access control (IAM)
Tags
SETTINGS
Quickstart
Resource costs
Deployments
Properties
Locks
Automation script
MONITORING
Metrics
Alert rules
Diagnostics logs
Application insights
Log analytics (OMS)
Essentials
Subscription name
Azure-pass
Last deployment
20/10/2016 (Succeeded)
Subscription ID
Location
Australia East
Filter items...
NAME
TYPE
LOCATION
28deliveryifb299dbserver
SQL server
Australia East
28DeliveryIFB299_db
SQL database
Australia East
28DeliveryIFB299Plan
App Service p...
Australia East
28DeliveryIFB299
App Service
Australia East

Release 2 - Artefacts

Order Management

I created and developed pages for managing orders for employees and administrators. This implies creating roles in the system, that separates normal users/customers from other employees and administrators.It also implies that employeesa and administrators can view, search, edit and delete all orders that have been placed on the website. They can finalize and order and change status of an order. This is to ensure that all orders will be satisfied.

The code for all of this can be found at these links:
Controllers
Views

- Delete order
- Order Details
- Edit Orders
- Index page of order management

Models used are the same as mentioned in Release 1, uses the same database tables.

When I created this, I first created a new area for this code. I figured this was a clean way to keep the code tidy. In this area, all code for order management is placed, and this is also an area that only are accessible for employees, and that is why the area is called Employee. The MVC structure is also used for all the order management.
The different sites can be accessed here. To reach the page you can use username "admin@28delivery.au" and password "#Password1".

Sprint and release planning

I have contributed to planning all sprints and releases. This implies redefining acceptance criterias, manage and create user stories, discussing with client which stories that should be implemented in the different sprints. This has been an important process, to ensure we always had just enough workload to finish of each sprint. After each sprint planning meeting we always made sure that we had a clear sprint goal, and a sprint backlog with user stories that should be finished during the sprint.

After release one we had to demote the team's total velocity, because one team member dropped out. That was quite unfortunate, because it led to that we were only three people doing the entire project.

After redefining the team's total velocity, each team member's velocity actually got way better. We moved one story from release two into release three. This was a big story, but it made it possible for us to finish sprint 2, without any big problems.

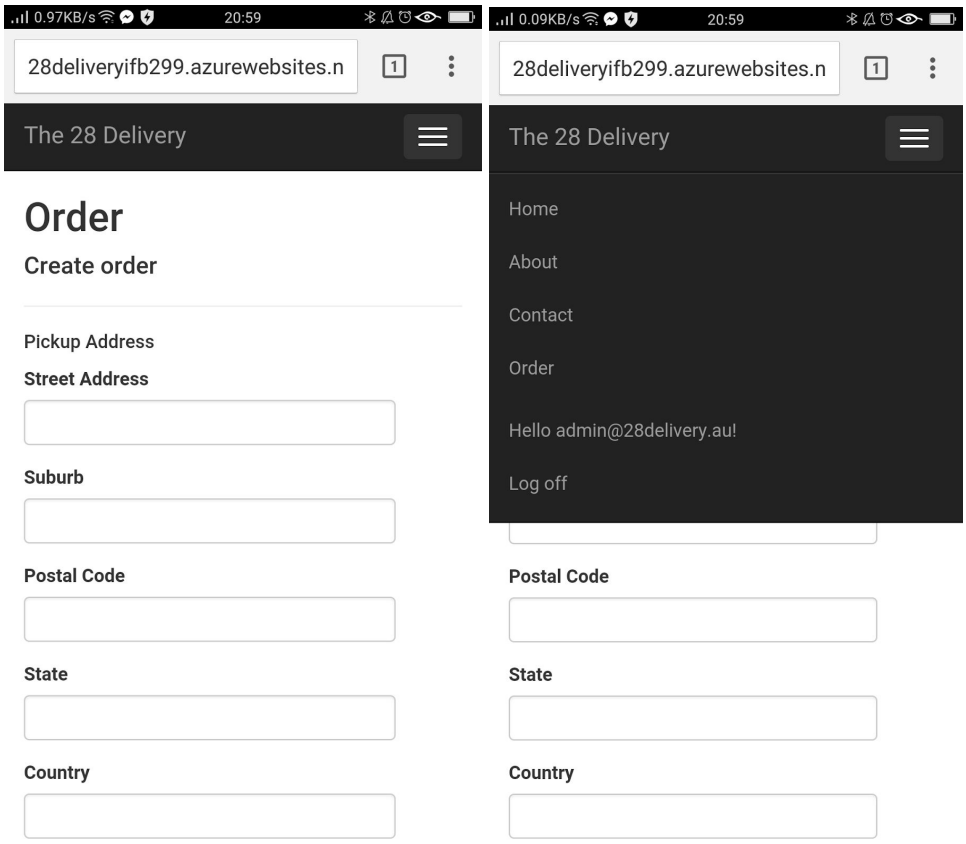
Sprint and release plans can be found in the team's repository, or by following this link.

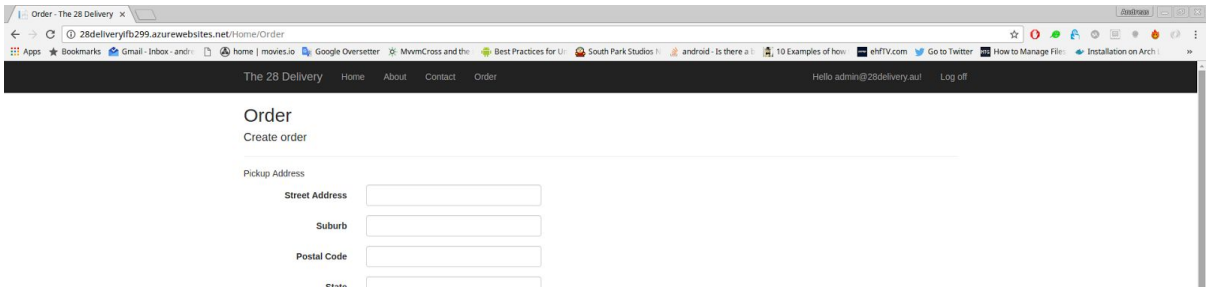
User Interface

I've helped design the user interface of the website. We created it in a simple, easy to use way, so users could use previous experiences from other websites on our website. This has led to an user friendly website. The website does automatically scale to different screen sizes, which means that if someone are using it on a tablet or mobile phone, it will still be fully functional.

We made it this way, because it should be easy to use for employees that were out on a delivery. This feature made it possible for them to check a delivery as delivered straight away, check up customer addresses and where the next delivery should go, without bringing anything else then a mobile phone or a tablet.

See pictures for how the design / user interface changes on different screen sizes.
The first two pictures are on a mobile phone. The one on the bottom is on a desktop computer.





Acceptance testing

I helped create all the acceptance test for the application. We created these tests before we started the actual coding. This made it very clear for what each feature of the application should do, and made coding the application far more straightforwardly. When I started coding a new feature, I could look at the criteria for that specific feature, and start implementing it. When I felt the feature was done, I could easily check it with the acceptance tests, and if they passed it was done.

Acceptance test are also important to encourage closer collaboration with customers. Since customer s are contributing to create these test, they can work as a “contract” between developer and client for which functionalities that are accepted, and when a feature can be considered done. This is the main reason that it was great having these test when I was developing. I always knew that when theses tests were passed, our client would be satisfied.

These test can be accessed in our Git repository in the file ‘tests’, or by following this link. Acceptance test are allocated from page 1-10.

User management

I have also created and developed all user management in the application. This implies that users can create their own account, manage all their details and have full overview of only their current and past orders. Users that are registered as employees also have full access to all users registered in the application. They can view and manage user details, which can be handy if a user for example calls customer service and asks for help. It is also convenient when employees are out on deliveries, to check the user’s profile before they pick up a package. Users registered as an administrator also have the possibility to “hire” people, by making a regular user into a employee. Then that specific user, will get access to all the employee restricted pages.

These pages are also created in the MVC structure, which have made the code very clearly structured, and easy to navigate through. Code for these page can be found in the links below:

Controllers:

- AdminController
- UserController(user managing their own account)

Views:

- Overview over all users(index file)
- Delete user
- Edit users
- Details about selected user
- ViewYourOwnDetails
- EditYourOwnDetails
- Index page for your own account
- View order details for specific order
- Overview of your own orders

Models:

- Database tables used (same as referenced earlier)