# ENSE 477
# Capstone Final Project
# Experience Report
# April 10th 2020

**Nickolas Schmidt, Nicolas Achter, Nikolas Lendvoy & Shayan Khan**
**The Ni(C)(K)S**

# Table of Contents

# Table of Figures:

**Introduction:**

For the ENSE 400/477 Capstone project the team consisting of Nickolas Schmidt, Nicolas Achter, Nikolas Lendvoy and Shayan Khan, dubbed The Ni(C)(K)S, chose to work with Eden Care Communities for their final project. This project consisted of creating a mobile application for Android and iOS that allows users to order home care services at any time. This application was later named Care on Demand, and is the topic of the ENSE 400/477 Capstone classes with The Ni(C)(K)S planning, designing and implementing this application for a due date of April 10th, 2020. With the help of Bill Pratt, a member of Eden Care Communities and Tim Maciag, the professor of ENSE 400/477 and mentor this project was conceptualized and a minimum viable product (MVP) was completed for April 10th, 2020.

**Background Information:**

After viewing the project presenter day in September of 2019 the Ni(C)(K)S chose the presentation done by Bill Pratt of Eden Care to develop a mobile application that works alongside their existing services. The team enjoyed the idea of using their software knowledge to help people. An additional reason for choosing this project for the teams capstone was the experience of developing for a mobile environment, which is something no one in the team had done prior. After approaching Bill about joining with him, the team was invited to a planning session in October to get a broader idea of the services that will be offered by the app. This also gave The Ni(C)(K)S the chance to meet more members of the Eden Care company. After that meeting the team had a much better idea on what Eden Care did and how the application would work alongside their system and broaden the customer base by adding a mobile application to the company.

**Technologies:**

Github was utilized immensely as a way to house all data required for this project. The github website and accompanying desktop app allowed all the members of the team access to the project files and make changes on their local devices before pushing changes onto the project. Github allowed The Ni(C)(K)S to work on multiple assets of the project and collaborate the changes each member made by uploading to the Github cloud. Alongside all the code files for the project Github was used to house all the information retained to the documentation of the project as well as the prototypes and other design diagrams. All presentations given during the scrum meetings of ENSE 400/477 are also hosted on Github. All necessary links to the discussed files can be found on Github's README page.

This project was planned as an Android and iOS app. There were a few different ways of development considered by the team. This included making a Progressive Web App (PWA), Native app development, or cross platform app development. PWAs, while being the

latest and most innovative method for app development, are still in their early stages and are missing some native device capabilities that would be required by the application. iOS specially does not have full support for PWAs and we wanted to avoid any potential obstacles that might have arisen because of this. Native development, while being the ideal way to develop, would also have taken more time and has a steeper learning curve which is why it was also not chosen. Ultimately, we decided to go with cross platform app development because of its good support and an active community. With regards to cross platform app development there were also a few options to consider. There are newer Javascript based platforms such as React Native and Ionic as well as older but mature platforms such as Xamarin. We elected to go with Xamarin.Forms, an extension of the Xamarin platform, for a couple reasons; it is C# based which was familiar to the group members already. As well, Xamarin has been in the market longer and there is more support for it with an active community. Therefore, when we do run into issues there is a plethora of knowledge and fixes that we could implement.

The app development framework we chose was Xamarin.Forms. This framework works with both iOS and Android devices and allowed the team to focus our efforts on only learning one new technology. The Xamarin project utilized the C# language which was familiar to the group already, as well as a HTML like language named XAML to handle front end and design work. At this time the group decided to split up into two teams to handle front and back end development. The members Nickolas Schmidt and Nikolas Lendvoy focused on front end development using XAML. The remaining members, Shayan Khan and Nicolas Achter would focus on backend development using C#. Deciding on the Xamarin.Forms App Platform alongside an ASP.NET framework was chosen early on. However, the team also needed to choose a platform to host this project on and ultimately, Amazon Web Services (AWS) was chosen as it was approachable and met the requirements we needed.

AWS was used as the digital infrastructure for our system. A REST API was written to allow our project to communicate with the database stored on Amazon's Relational Database Service (RDS). Likewise, Amazon's Simple Storage Service (S3) was used to store the various images that are used throughout the application so that images could be dynamically loaded. The use of S3 made it easy to include a website link in our code to load up the images when we wanted them. The previously mentioned REST API is hosted using Amazon's Elastic Beanstalk so that the database is always accessible by the application. Login and signup validation is performed using Amazon's Cognito service, this allows us to securely store the user's account data separately from the database as well as performing email verification upon account creation or password changing.

In order to track user stories and tasks, DevOps was used. Using DevOps the team separated the tasks into different users, a Admin, Customer and Care Partner and began planning the tasks and user stories for each class of users. Over time it became apparent that using DevOps as a way to track milestones became very tedious. In February of 2020 the

Ni(C)(K)S transitioned to using Trello, an online kanban board as a way to easily monitor the individual tasks created. The tasks were separated into three groups, design, implementation and testing. This separation made organising the different goals the team had set out for the project simpler. Each group created in Trello had three different states that every task would be in. These states were either, “To Do”, “Doing” or “Done”.  Each task created on trello was given a label with its group name and a colour that matches the group as well as the member who was the individual responsible for that task.

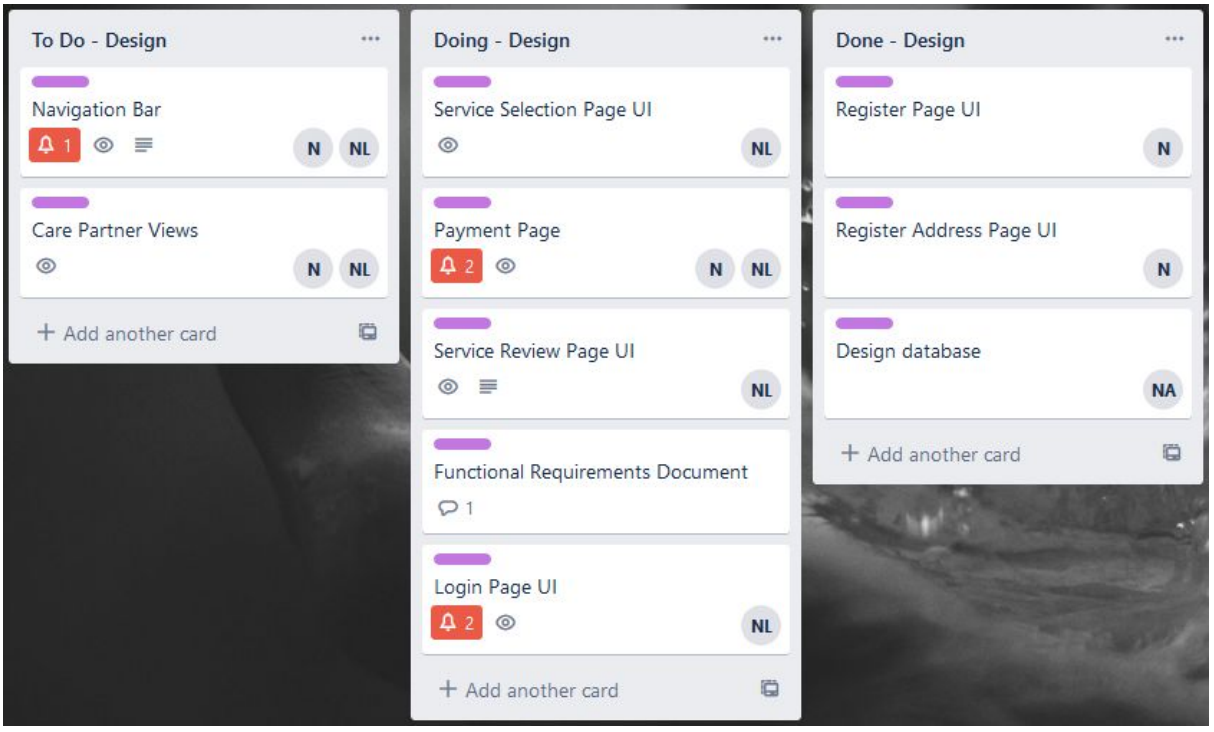


**Figure 1: Trello Design Screenshot**

Finally, Stripe, a payment service for Google and Apple pay, was the planned technology we wanted to implement for covering order payments. However, due to time constraints a payment function is not part of the MVP and will be developed at a later date using the Stripe technology.

**Design:**

Since our application was going to be primarily used by elderly individuals the design we created needed to be easy to use and simplistic to allow less tech savvy people to maneuver through it with ease. We chose a minimalistic design and wanted to stay away from features like scrollable pages and nested lists throughout the application. Original wireframe designs were done in November using the website wireframe.cc.

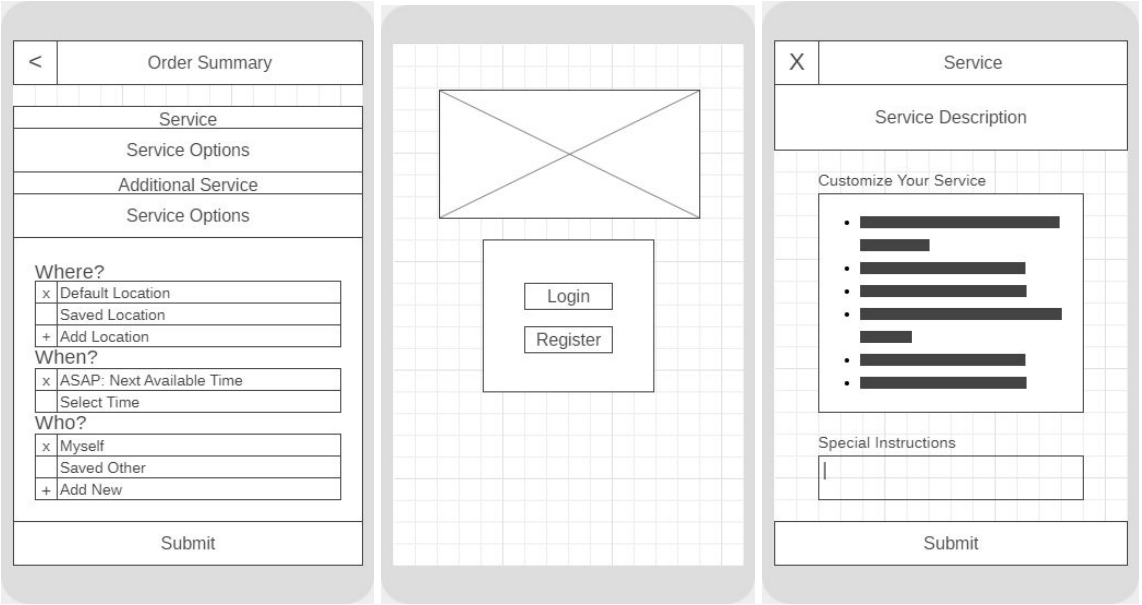


**Figure 2,3,4: Wireframe diagrams (November 2019)**

After reviewing our original design with members of Eden Care, more detailed high-fidelity prototypes were created in January of 2020 using the Balsamiq mockup software. These prototypes continued with our design philosophy of minimalistic pages that don't overwhelm the user with information. On certain pages, including the order details page, this tactic became more difficult. However, we continued to implement these decisions wherever we could. Another design we wanted to implement was a contact button on every page of the app. This button would allow a frustrated user to contact Eden Care to get assistance navigating the app or placing an order. Alongside this button we chose to have all primary buttons of the application span across the entire width of the app and include bold lettering indicating the function of each button. These prototypes were then shown off during a ENSE 477 scrum featuring both Tim & Bill where they were approved at which time development began on these pages

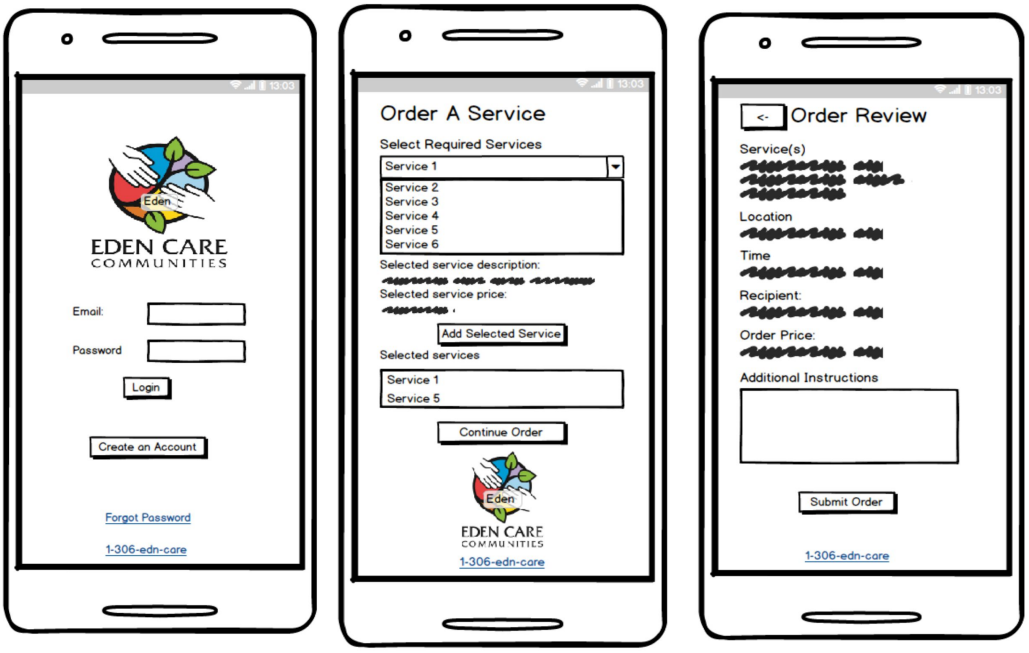


**Figure 5,6,7: Balsamiq High Fidelity Prototypes (January 2020)**

Once the design was approved we started working with XAML to begin designing the initial pages for the customer views. This included a login page, register page and forgot password page. The design philosophies we chose including large buttons and a contact Eden Care button are visible in even the early designs. Blocks of colour or blank spaces were utilized at the top of the pages to grasp the space that would eventually be taken up by the Eden Care logos. All of the labels were designed to be bold and easily readable by our customer base.

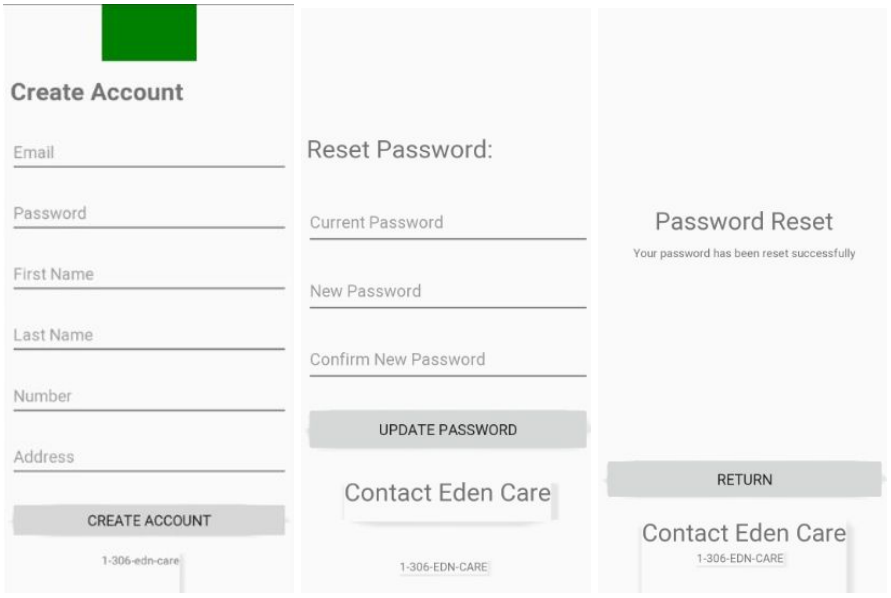


**Figure 8, 9, 10: Care on Demand early screenshots (January 2020)**

The design for our project slowly improved as we became more familiar with the XAML language. The final design utilized multiple page layouts including both a stack and

grid layout. Likewise, features we tried to ignore, including scrollable pages became necessary for locations which help too much information for one screen. This was very noticeable when we tested our application on older devices that contained a smaller screen size than our personal devices that we were using as our primary test devices. We also needed to consider tablets and make sure our application would scale to a readable size on larger screens like that of an Apple iPad.

While new pages were being designed in the front-end the next step was to link the elements on the pages to actual data inside the code and to be able to process that data and perform the essential application tasks. We decided on the Model-View-ViewModel (MVVM) design pattern to structure our code. The MVVM pattern is very similar to the traditional MVC pattern and is highly useful for developing applications with clean code using C#. Each View or page created has its own ViewModel where the data on that page is processed. The result is a lot of different view and ViewModel files which are essentially self contained files and are therefore easily testable. The MVVM pattern follows a strict separation of concerns between the View, ViewModel and Model. The team tried their best to follow all the rules defined by this pattern and to follow the recommended best practices. There are, however, a couple instances where some MVVM rules had to be violated because either the alternative was a very time consuming task or there was no alternative at all.

**Implementation:**

Implementation began in early January with the team learning how to develop in a Xamarin environment as well as learning to use all the features of the various AWS services that were employed. Front end development was done in XAML with most pages being laid out in a grid by defining multiple grid rows and columns on each page. This layout made it easy to place items where we wanted them and allowed us to span buttons and text any width we liked. Initial front end implementation utilized a lot of placeholder text and images which would later be pulled in and displayed from the database. As we had planned out three seperate users, customer, admin and carepartner we decided it was best if we focused on one view at a time. The customer view was implemented initially with login and register functionalities being the first time we connected our project to the database and created something that worked.

After a few initial page designs were complete, work started on the backend side where elements on the page were programmed. As mentioned earlier, the MVVM design pattern was followed to keep the code structure simple and clean. Each view had its own ViewModel file where data from that view was processed. Each ViewModel could interact with any of the Models to store data for processing. Later on, we created several REST services that interacted with the REST API to access data stored in the database as well as other program files that interacted with the AWS services that were used.

The REST API was designed following the MVC design pattern and written using the ASP.NET framework so it would be easily compatible with the Xamarin.Forms project. Database tables were translated into Model files and the database actions, such as GET and

PUT, were implemented in the Controller files. No frontend was developed for the API as testing was performed using Postman, an API development platform, so no View files were created as there were not necessary. As mentioned earlier, the REST API was hosted using AWS's Elastic Beanstalk service for easy scalability and user friendly monitoring; this allowed the application to query the database at any time as the API was continuously available as opposed to if it were hosted locally.

**Reflection:**

As with any project, starting earlier or being more on top of things at an earlier stage is an improvement all groups could make. With our application specifically, we delayed far too long in choosing a cloud service to house our project. This was due to us focusing our time on early designs and documentation instead of beginning the implementation phase of our project. Coding did not begin until January, which included having to set up all of the necessary cloud infrastructure which should have ultimately been done much earlier than it was. This would have allowed for us to speed up the time it took for our application to start communicating with the cloud.

Another improvement we would have made was to stick to our original plan of agile development. As development on our application continued we found that we fell into a waterfall cycle. We would focus on one task and complete and test it before moving onto another. This became apparent when we focused all of our efforts into only working on the customer view before we began developing the other user flows of our application. With a better plan that more resembled the agile development cycle we believed we would have had a more fleshed out MVP that we could have tested with a customer base before the COVID-19 breakout.

Even though we would have liked to start our coding phase earlier we also realized that our initial planning and prototyping was not robust enough and we had to improvise for several aspects of the application. We realized that we needed pages that we never thought of and whose prototypes were never made. We made an initial Entity Relations Diagram but when we actually got to the coding a lot of it was changed and other practices were implemented. Some of the models and database designs we initially came up with needed to be amended due to requirements we failed to consider during our initial planning phase.