

Group 1

One-Time Two-Factor Authentication

Amy Pierce – 17330305 - <https://github.com/amy-pierce>

Jamison Engels - 17300599 - <https://github.com/engelsj>

Ciara O'Sullivan – 17321934 – <https://github.com/ciaraos>

Liam Collins – 17301097 - <https://github.com/lpc477>

CSU33BC1
Blog Post

BACKGROUND AND PROBLEM

Users generally log into their online accounts with a username and a predefined password. For the sake of convenience, many people often use the same credentials across multiple accounts. However, this creates an extra security risk. Hackers can install malicious software on a device such as a keystroke tracker to save these credentials. Not only does this give the hacker access to the account currently being logged into. They can also gain access to all other accounts with the same login information.

Two-factor authentication and one-time passwords (OTP) were introduced to combat this security risk. Two-factor authentication works by sending a code by text or email to the user after they enter their username and password. This code must be entered into the website to gain access to the account. One-time password works similarly. Rather than entering the entire predefined password, the user can request a one-time password. This will be sent to either their phone or email. With both of these methods, the hacker must have two things in order to gain access to the account, the user's login details and access to their phone or email.

The problem with these methods is that they don't prevent the hacker from being able to phish the data. While the hacker may not have access to accounts protected by two-factor authentication, they can use the phished data on any other account with the same credentials. In the event of two-factor authentication being deactivated, the hacker can gain access to the account. One-time password may stop hackers from phishing the predefined password, but it removes the need to know some private information (the password). If the hacker gains access to the user's phone, then they will have complete access to their account.

Reference to the original problem statement can be found at:

<https://joeranbeel.atlassian.net/browse/STUD-20> (requires authentication)

GOAL

Our goal was to develop a program where a user can login without entering their full password. Instead, the user enters only part of their password, for example, they could be asked to enter the 3rd, 6th, and 7th indexes of their predefined password. This method is more secure as even if a hacker phished this information, they will not know the entire password. They will also be sent an OTP by text. This must be entered into the website and verified in order to gain access to the account. This provides additional security by checking if the user has knowledge beyond access to the phone, and all the benefits of two-factor authentication.

NON-TECHNICAL SOLUTION

Our project allows users to choose between a "secure" and "unsecure" login. The unsecure page is just the normal login process with a username and password. Secure can be used for extra security if the user does not fully trust the website. This is where the user will be asked to enter their mobile number so they can be sent their one-time password. After the OTP is entered, the user will be asked for random indexes of their predefined password. This provides all the benefits of one-time two-factor authentication with the additional security of a partial password, preventing hackers to be able to phish all the private information. The person logging into the

account must not only know the whole password to be able to enter the random indexes, they must also have access to the account owner's phone.

DESIGN CHOICES

We decided to change the proposed order of events for the login solution to one we felt was more secure. Instead of having the user enter their partial password and then their OTP, our application asks for the OTP first before asking for the partial password. We felt this was more secure as by sending the OTP the user should get a text with the validation code and this will alert them that someone is trying to access their account. Twilio also has a request limit to prevent spamming. This provides an extra layer of security, as if the hacker does not guess the password within the given limit, they will not be able to receive any more OTPs. In short, we changed up the order of events in the login process to make hacking the account by brute force more difficult.

Another design decision we made was to integrate our own functionality into a premade react app by GitHub user, giorgi-m, the repository can be found here:

<https://github.com/giorgi-m/online-shop?fbclid=IwAR3LIAWiNSmhDrND2gkn0J8DgDUS9wG8azRLAf46fqVd29fSMTXhUK1Ezx0>.

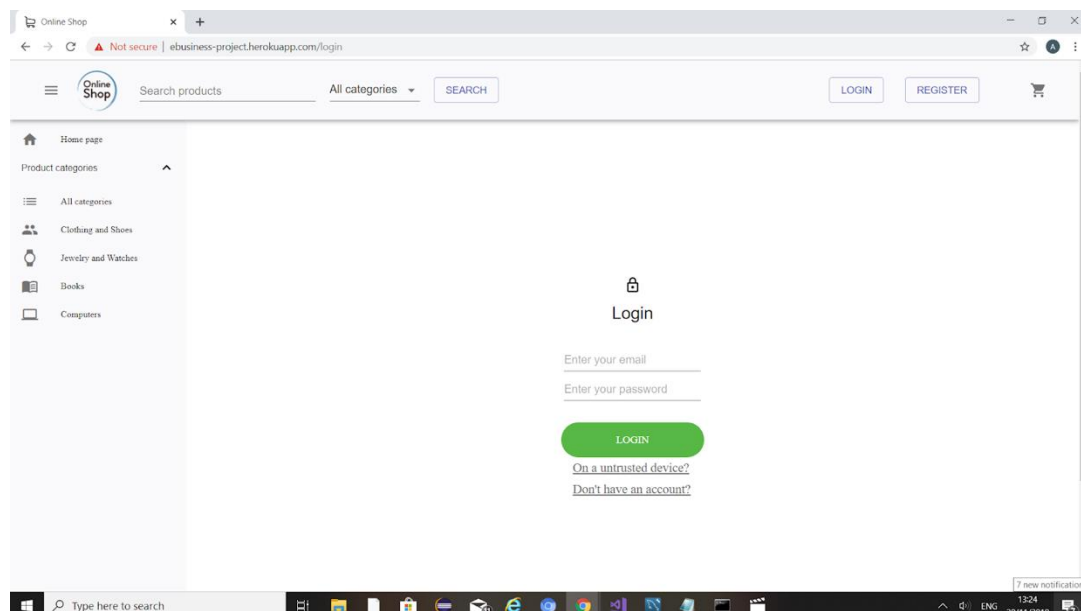
We did this because we didn't want to reinvent the wheel. Rather than spending time developing a whole new, fully functioning website, we felt it was more important to spend our time on the problem at hand. Our choice to do this was influenced by Professor Beel's comments after our prototype demonstration. We followed the principles of MVP (lecture notes week 1 slide 25).

FUNCTIONS

The following screenshots and their descriptions outline the functionality of our project:

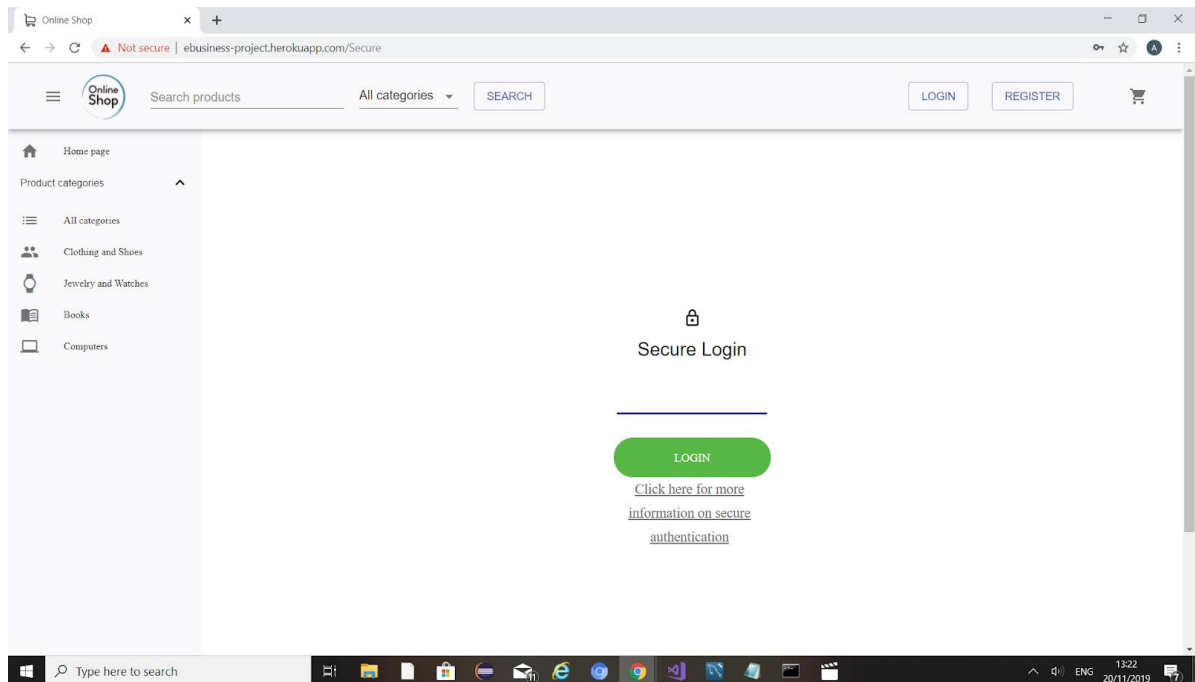
UNSECURE LOGIN

When the user clicks login they are initially brought to this page, as can be seen in the following screenshot. They are given the option to login as normal or click one of the links below to go register or choose the extra secure login for untrusted computers.



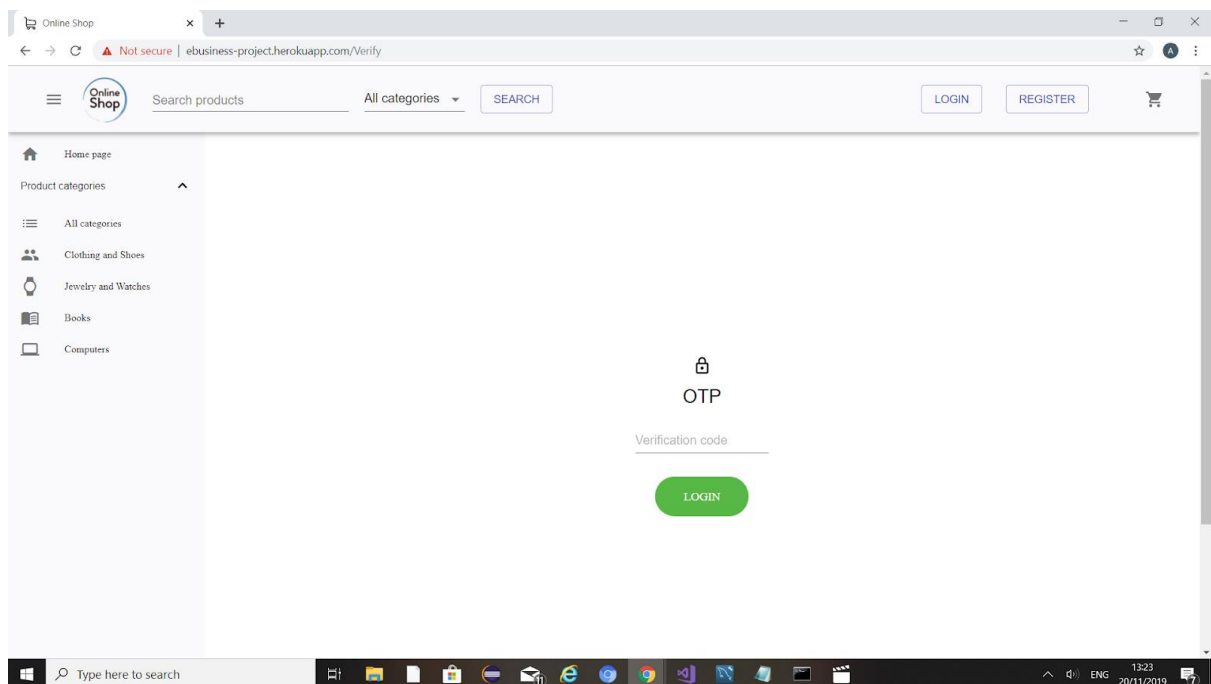
SECURE LOGIN

If the user opts to go for secure login, they are pushed to this page where they are required to enter just their email. They can also follow the link to find out more about secure logins.



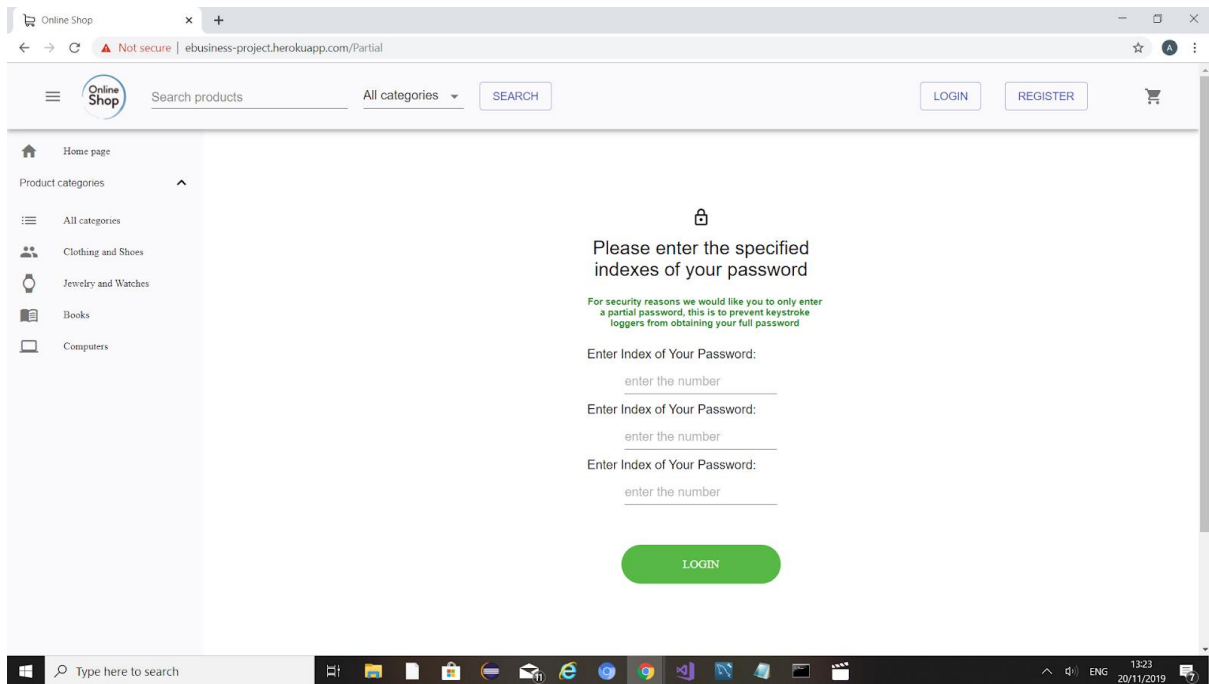
OTP

Once the user enters their email and it is validated, an OTP is sent. They are then brought to this page where they enter the OTP received on their mobile device.



PARTIAL PASSWORD

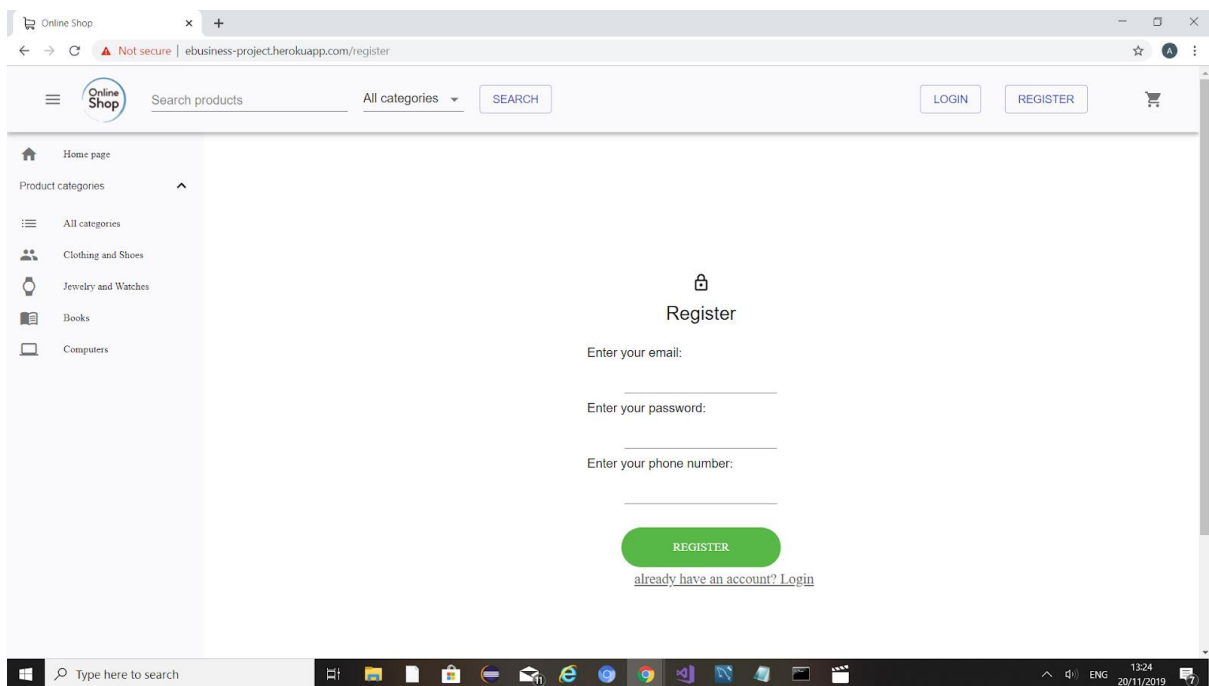
If the OTP is valid they are then pushed to this partial password page where they are prompted to enter specific indexes of their password.



The screenshot shows a web browser window with the URL `ebusiness-project.herokuapp.com/Partial`. The page has a header with a search bar, a category dropdown, and 'LOGIN' and 'REGISTER' buttons. A left sidebar lists product categories: Home page, All categories, Clothing and Shoes, Jewelry and Watches, Books, and Computers. The main content area is titled 'Please enter the specified indexes of your password' and includes a security note: 'For security reasons we would like you to only enter a partial password, this is to prevent keystroke loggers from obtaining your full password'. Below this, there are three identical input prompts: 'Enter Index of Your Password:' followed by a text box containing the placeholder 'enter the number'. A green 'LOGIN' button is at the bottom.

REGISTERED

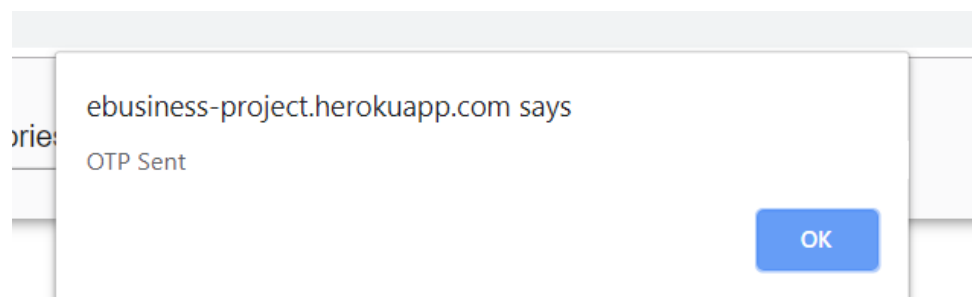
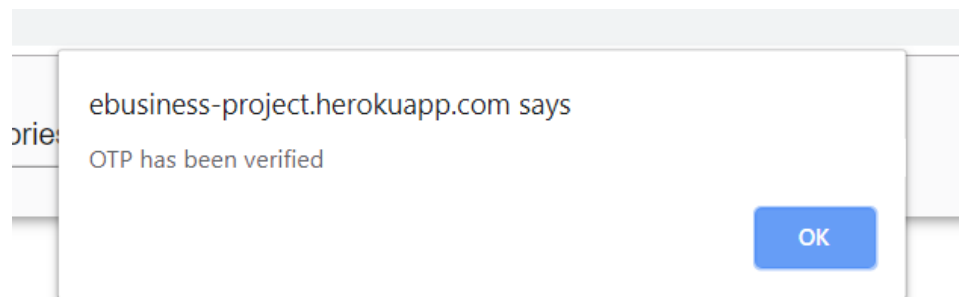
If the user chooses to register, they are brought to this page and asked to enter their email, password and phone number.



The screenshot shows a web browser window with the URL `ebusiness-project.herokuapp.com/register`. The page layout is identical to the previous one, but the main content area is titled 'Register'. It contains three input prompts: 'Enter your email:', 'Enter your password:', and 'Enter your phone number:', each followed by a text box. A green 'REGISTER' button is at the bottom, with a link below it that says 'already have an account? Login'.

ALERTS

We made use of alerts to let the user know their status for the current stage of the login process: e.g. that the OTP has been sent or login failed etc.



ERROR CASES

We accounted for error cases on login fields such as leaving a field blank or not entering a valid email. Our application displays these errors for the user to give them a hint as to why they could not move forward in the login process. We implemented this through a series of if statements and a call to a function that validates the login form.

A screenshot of a login form. At the top is a lock icon and the word "Login". Below it is a text input field with the placeholder "Enter your email". A red error message "Email can't be blank" is displayed below the field. Below the email field is another text input field with the placeholder "Enter your password". At the bottom is a green rounded button with the text "LOGIN". Below the button are two links: "On a untrusted device?" and "Don't have an account?".

A screenshot of a login form. At the top is a lock icon and the word "Login". Below it is a text input field containing the email "student@tcd.ie". Below the email field is another text input field with the placeholder "Enter your password". A red error message "Password can't be blank" is displayed below the field. At the bottom is a green rounded button with the text "LOGIN". Below the button are two links: "On a untrusted device?" and "Don't have an account?".

DEMO

Our project was published under the MIT license. All source code can be found within the GitHub page:

<https://github.com/engelsj/Ebusiness-Project>

A live demo of our project can be found at:

<https://ebusiness-project.herokuapp.com/>

A video demo of our project working can be found on the following YouTube link:

<https://www.youtube.com/watch?v=TT4H4aYJW3A&feature=youtu.be>

TECHNICAL SOLUTION

KEY DESIGN CHOICES

Initially we started our project with the intention of a monolithic structure. As time went on, we saw that a microservice structure would be better. We used a react app, Micronaut, Twilio, Kubernetes, and MySQL to complete our project. The react app is written in JavaScript, CSS and HTML. This is the front end of our project where the user can register an account or login. We used Micronaut as the framework for our microservices. Our microservice handles generating and verifying the partial password. It is also used to call Twilio to generate and verify an OTP. Kubernetes is the platform we used for deployment to Google cloud so we can have our code run off local. Finally, we used MySQL to create and query our database. The database stores the users' email, password and phone number.

TECHNOLOGIES USED AND LECTURE REFERENCES

React App – Front end → Week 5/6/8 – HTML, CSS, WikiMarkup, Emails

Micronaut – Microservice framework → Week 10 - Conor Gallagher Guest Lecture

Twilio – OTP Service → Week 10/11 - Web services

Kubernetes – Deployment Method → Week 2 – Entities, Servers, Connections

MySQL – Database management → Week 9/10 – Structured Data, XML, Encoding, HTTP

Heroku – Webhosting → Week 2 – Entities, Servers, Connections

SMALL FEATURES ADDED FROM LECTURE CONTENT

CSS to stylize our webpage → Week 5/6/8 – HTML, CSS, WikiMarkup, Emails

POST/GET Requests in our Microservice → Week 10/11 - Web services – Slide 21

Use of Micronaut and Basic Naming Conventions → Week 10 – Conor Gallagher Guest Lecture – Slide 15

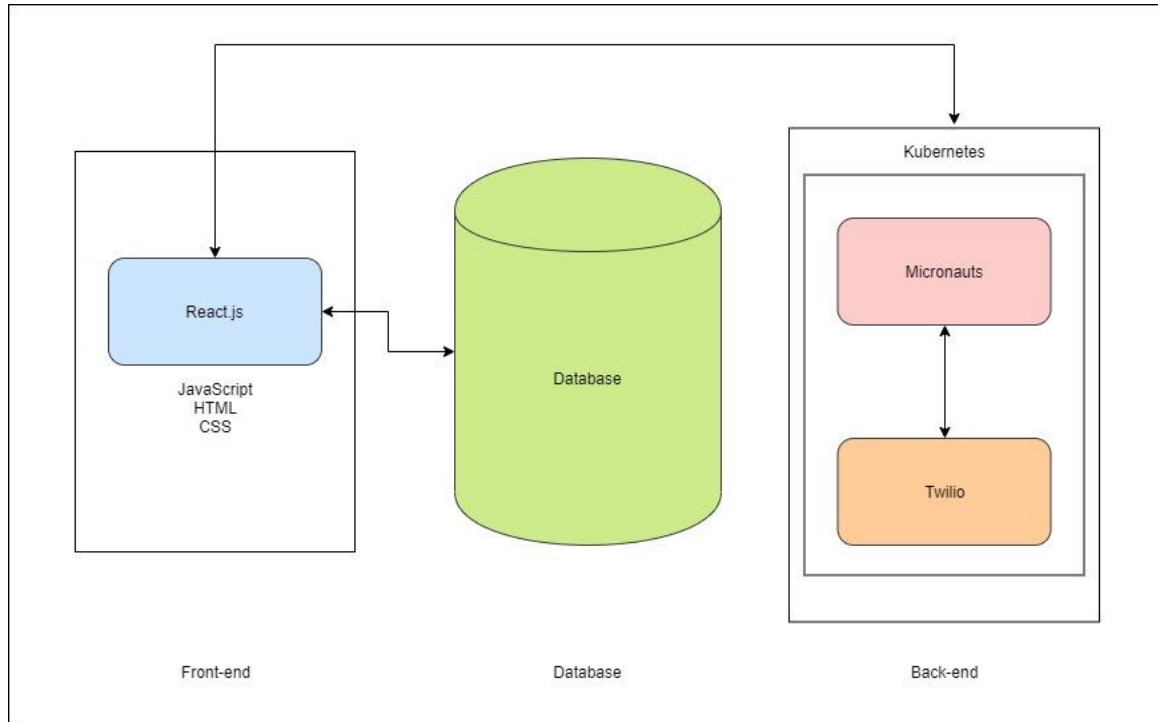
Semi Structured JSON Responses from our Microservice → Week 9/10 – Structured Data, XML, Encoding, HTTP – Slide 8

Sub Domains for Different Parts of our Website → Week 3 – Domain Names and Domain Name System – Slide – 16

Proxy Server to Circumvent CORs Issue (Not in final Prototype) → Week 4 – Routing and Protocols – Slide 17

OVERVIEW

ARCHITECTURE DIAGRAM

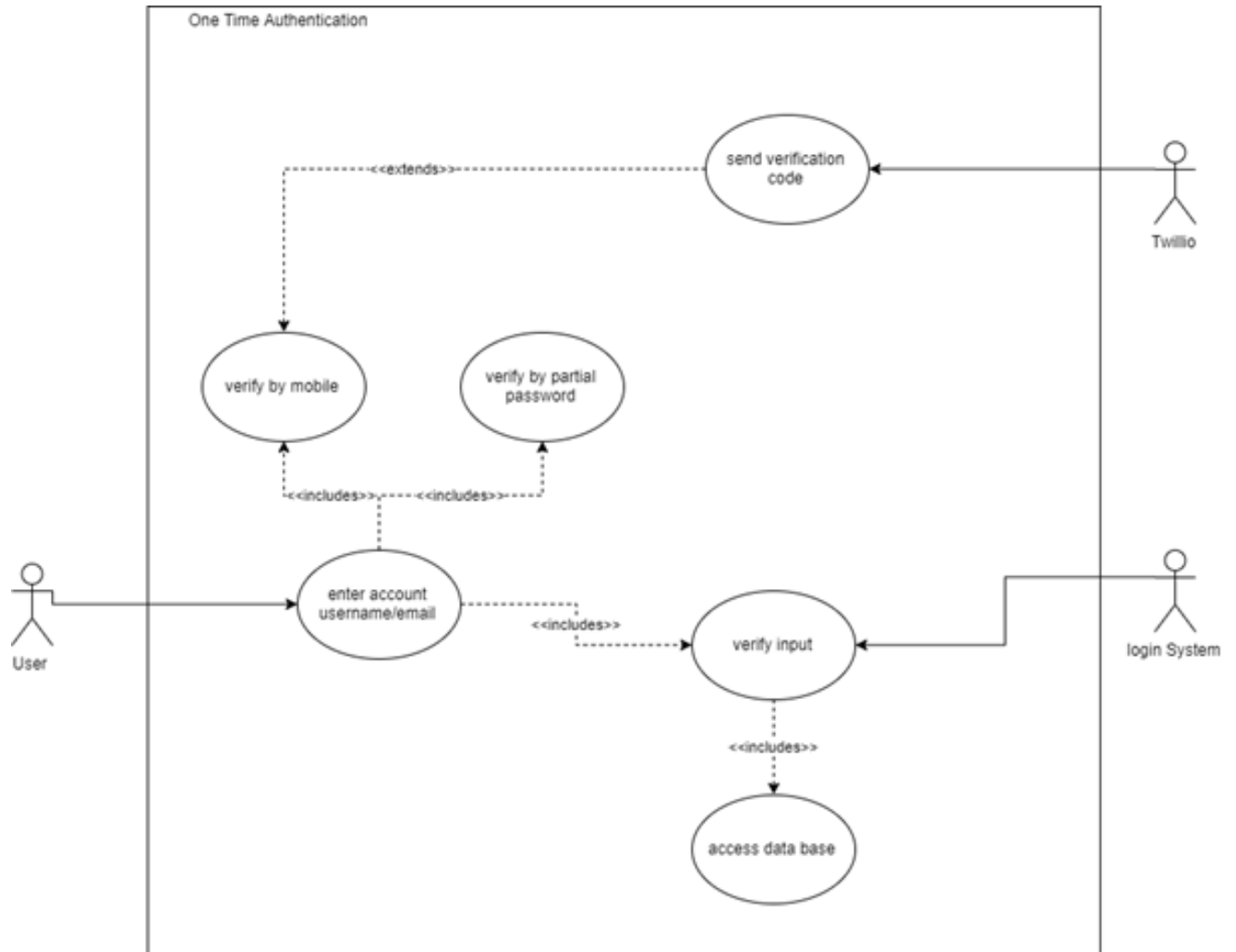


This diagram shows how our platform works in its current form. The Kubernetes, Mircoservice, and Twilio on the right side shows the backend processes that are operating. In the ideal scenario, the Microservice (operating on a Kubernetes cluster) would receive the field information from the frontend, make the HTTP requests to the database, and also administer and check the OTP passwords. This would be an incredibly powerful tool for any E-Business, as it would give an added security buffer of the services and allow for simple implementation of an OTP protocol.

In reality, this demo shows the functionality of the OTP administration services. All parts function as designed, but the database is linked to the frontend. We realize that this exposes many security risks and other issues. This iteration is not designed for public use/deployment. Our version does show what we think is the best way to administer two-factor authentication with a one-time password via the designed microservice system. A database hook-up would make this deployment ready and should not be too difficult to implement. In truth it is probably better to allow programmers to design a database to their own specifications anyway. If given more time, we would work on a fully deployment ready version of this system, with full database integration.

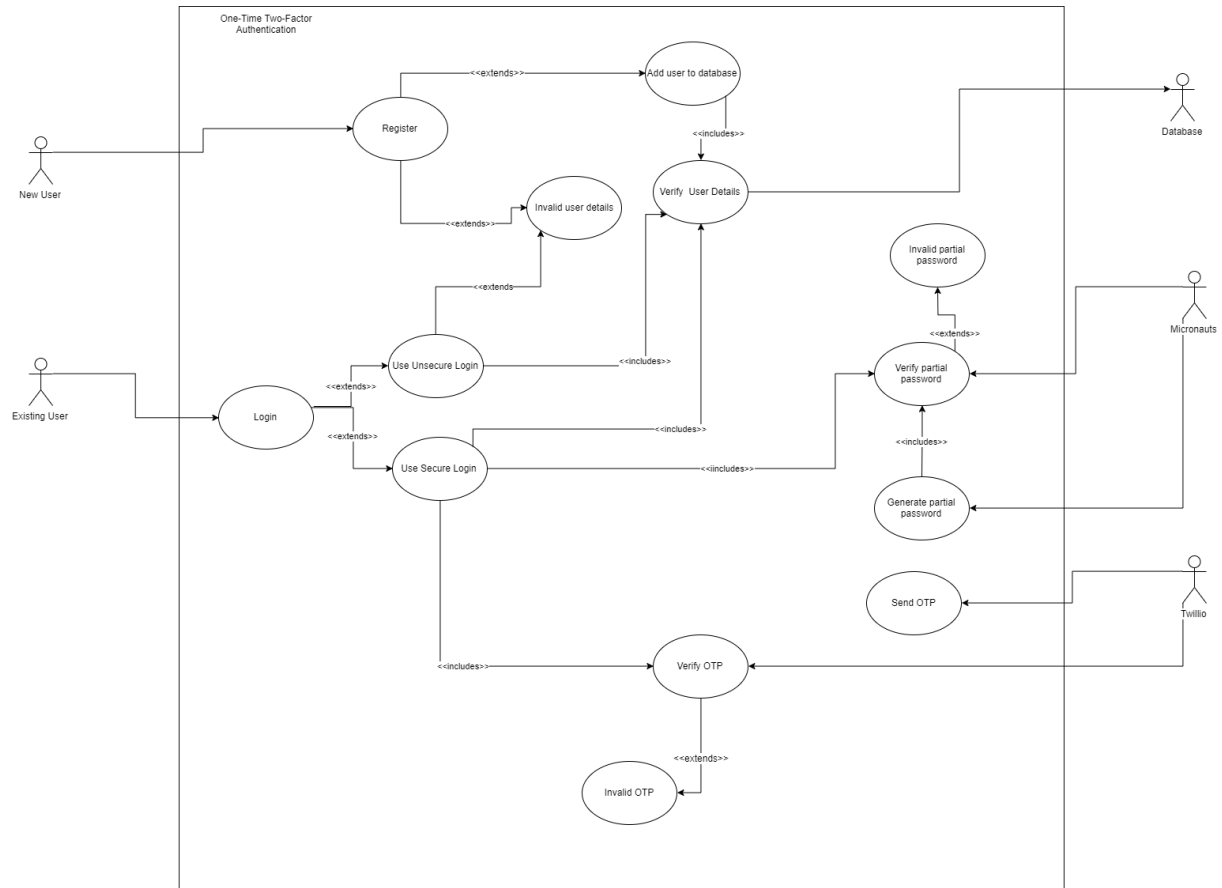
USE CASE DIAGRAM

INITIAL PROTOTYPE DIAGRAM



This was our original use case. When we presented our initial prototype, we worked from this diagram. However, as we got further into the development process, we realised that this no longer fit our design and we had to adjust or diagram accordingly. Our new use case better represents the flow of our application.

FINAL PROTOTYPE DIAGRAM



USE CASE TEXTUAL DESCRIPTIONS:

Name: Register

Participating actors: New User, Database

Entry condition: User is on the correct website and enters valid details

Exit condition: User has successfully registered

Normal scenario:

1. User opens the website and clicks to register
2. User enters credentials
3. Database adds user details

Error scenario:

- User enters invalid credential and unable to register

Name: Unsecure login

Participating actors: Existing User, Micronauts

Entry condition: User is on the correct website with the correct login details

Exit condition: User has successfully logged in

Normal scenario:

1. User opens the website and clicks to login
2. User chooses unsecure login
3. User enters credentials
4. Micronauts verifies user credentials

Error scenario:

- User enters invalid details

Name: Secure login

Participating actors: Existing User, Micronauts, Twilio

Entry condition: User is on the correct website with the correct login details and access to their mobile phone

Exit condition: User has successfully logged in

Normal scenario:

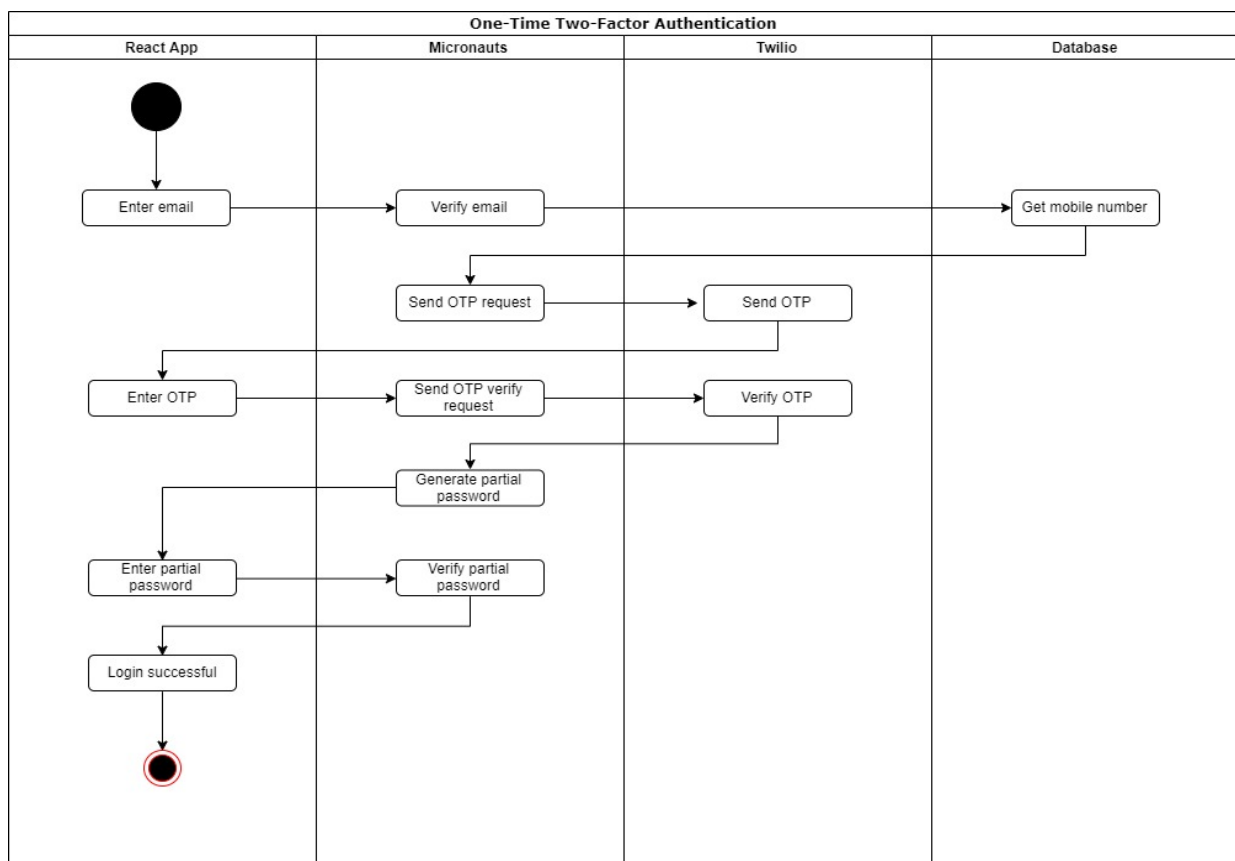
1. User opens the website and clicks to login
2. User chooses secure login
3. User enters email
4. User enters their login details
5. Micronauts verifies email
6. Twilio sends OTP

7. User enters OTP received by mobile phone
8. Twilio verifies OTP
9. Micronauts generates partial password
10. User enters partial password
11. Micronauts verifies partial password

Error scenario:

- User enters invalid email
- User enters invalid OTP
- User enters invalid partial password

ACTIVITY DIAGRAM



This activity diagram represents the happy case for our secure login. When the user has chosen the secure login method, they will be asked to enter their email. Micronauts will call the database to check if the email is valid. If valid, the database will then get their mobile number and use this to send the OTP. Micronauts sends an OTP request to Twilio. Twilio sends the OTP to the user, who will then be asked to enter it into the website. Micronauts then sends a verify OTP request to Twilio. If a valid OTP is entered, micronaut will generate a partial password. The user will

then be asked to enter their partial password. Micronauts verifies this. Once verified, the user will gain access to their account. Our final activity diagram remained relatively the same as the original but had just been updated slightly with better names and functionality. We feel the final diagram better represents the flow of our application.

FRONTEND

No one on our team really had much experience in front end development so we had to do some research as to what would be best to build our front end. We decided to use React.js because one of our team members had a small bit of experience using it.

WHY WE CHOOSE REACT

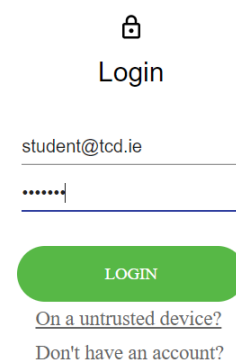
React is a JavaScript library used for building user interfaces. Using node, you can run your application on local host to see the development. Another reason we picked react is that it is easy to adapt for the likes of ios deployment. We thought would be useful as realistically the majority of people these days use their phones for everything. Given that we were working on login authentication it seemed like a smart move for thinking about moving forward beyond this project. It is also really easy to use once you have node.js which is fairly easy to install on any computer.

PARTIAL PASSWORD DESIGN

To begin with the partial password aspect of our project was part of a monolithic structure. However, once we found our feet with the idea of using microservices we decided to migrate the partial password aspect to a microservice of its own that could be hit from the front end. We decided the most secure way to ask for the partial password was to ask for three random indexes of your password. We had considered just asking for the last three characters but this felt less secure as it would be easier to guess. We also made the design choice to not display the tokenized password on the screen so as to make it more difficult for unauthorized users to access your account because they would have no idea how long that password is.

PARTIAL PASSWORD PROTECTION

As a security feature we tokenized the password on any text field the user is required to enter the private information in, so as to prevent others seeing them type in their password.



A login form mockup with a lock icon and the text 'Login' centered at the top. Below this are two input fields: the first contains the email 'student@tcd.ie' and the second contains a masked password '.....'. A green rounded rectangular button labeled 'LOGIN' is positioned below the password field. At the bottom, there are two links: 'On a untrusted device?' and 'Don't have an account?'.

DEPLOYMENT/HOSTING

We looked into a few different hosting services for the front end such as aws,s3 and firebase and we decided that Heroku was the best way to go about deploying our website. Heroku is an online deployment service that allows you to host your website or application. We had some difficulties deploying our final product due to connection errors between Heroku and https websites.

DATABASE

When we first attempted to make a database to connect to our front end, we used MySQL and Django for our project to store our user's data and their encrypted partial password. We had a lot of trouble connecting our database to our front end and learning how to make database calls from the microservice built on Micronaut.

Our first plan was to build an SQLite database with a Django REST API framework. The Django framework is very powerful from a UI and flexibility perspective. It allowed us to run both the lightweight React and Djangos at the same time. The Django comes with a superb backend visualization as well, allowing for easy addition for test values and testing of the JSONS being delivered.

One of the first issues that we encountered, was that although Django allows for the rendering of views in many ways, it must be parsed within a Django or React app. This creates an issue with using the microservice to process the data. We tried many different parse forms, inputs, key rendering, and view posting, but none were readable by the service.

The solution we ended up going with was to set up a standard MySQL. This is a very common build, and most will be able to set it up without difficulty. The trick to enabling the REST API is to use the program xmysql. This program creates all possible APIs for a mysql, allowing for the kind of dynamic returns that create a much more flexible database.

We successfully connected the database to the front-end and were able to make calls to the database using fetch commands in the front end that then query the SQL database.

DATABASE SCHEMA

```
CREATE TABLE `accounts` (  
  `email` varchar(45) NOT NULL,  
  `password` varchar(45) NOT NULL,  
  `phone_number` varchar(45) NOT NULL,  
  PRIMARY KEY (`email`),  
  UNIQUE KEY `email_UNIQUE` (`email`),  
  UNIQUE KEY `phone_number_UNIQUE` (`phone_number`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

DATABASE FUNCTIONALITY

GETTING USER FROM DATABASE

```
SELECT password,phone_number FROM `simple-react-sql-db`.accounts WHERE email='student@tcd.ie';
```

	password	phone_number
▶	1234	0858197109

ADDING A USER TO THE DATABASE

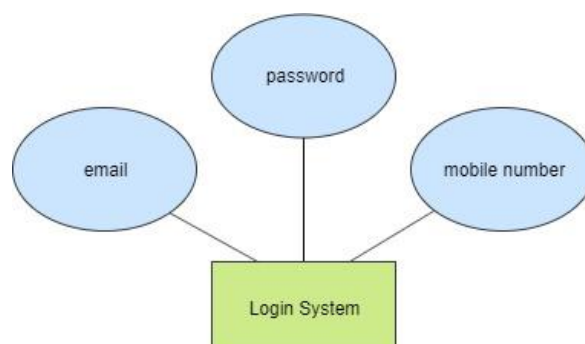
```
INSERT INTO `simple-react-sql-db`.accounts (email,password,phone_number) VALUES('student@tcd.ie','1234','0858197109');
```

	email	password	phone_number
	ciara@tcd.ie	csb	08598765432
	college@tcd.ie	123456	0851031234
	jack@tcd.ie	college	123456787654
	liam@tcd.ie	pizza	0851234567
▶*	NULL	NULL	NULL

	email	password	phone_number
▶	ciara@tcd.ie	csb	08598765432
	college@tcd.ie	123456	0851031234
	jack@tcd.ie	college	123456787654
	liam@tcd.ie	pizza	0851234567
	student@tcd.ie	1234	0858197109
*	NULL	NULL	NULL

Although we were able to get the database connected, it was run over localhost server and so our deployed website does not have access to the database. We adjusted our code to function fully even without the database and if we had more time, we would have been able to get the database hosted and accessible on any machine. However, we decided to explore more technologies such as the microservices rather than working more on the database as we felt it would be a better and more beneficial learning experience to explore new things

ENTITY RELATIONSHIP DIAGRAM



BACKEND

For our backend, we used microservices developed in Java with Micronaut that builds with Gradle. We used RESTful API calls from our microservices to communicate between the frontend and backend. We used Twilio's OTP SMS services to send and verify OTPs. while we could've used Twilio's RESTful API, their SDK made integration more seamless in practice.

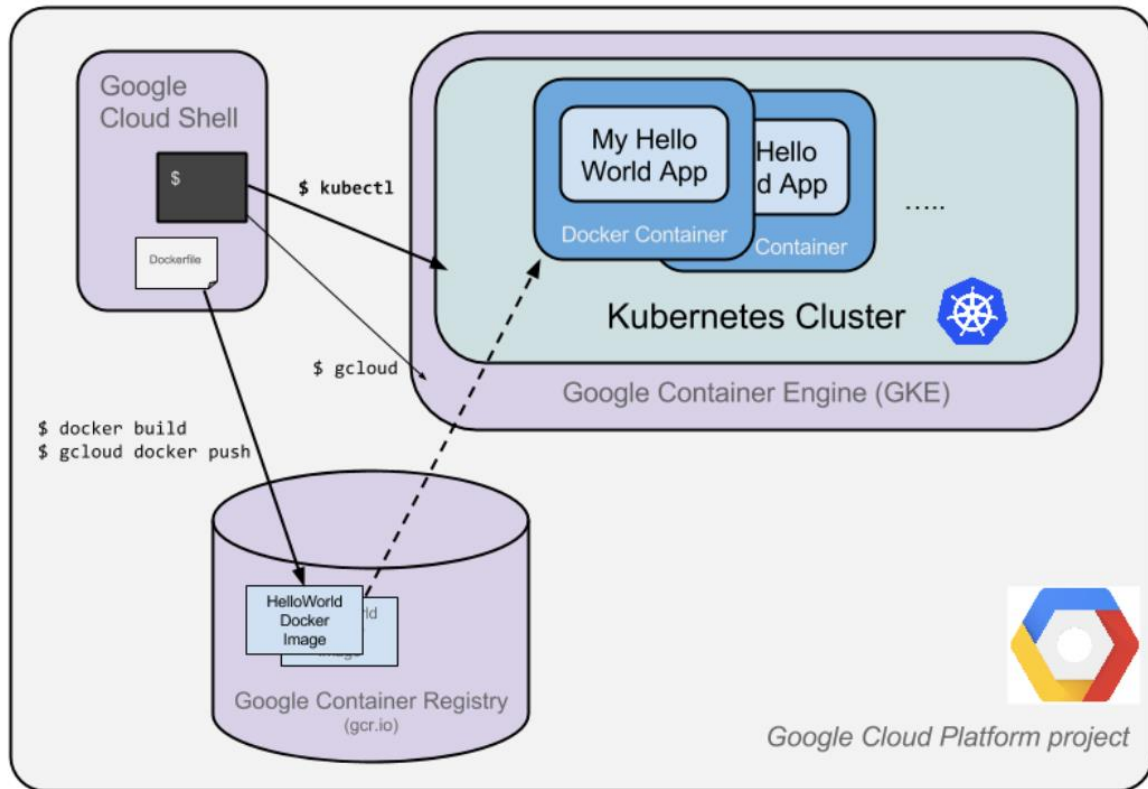


Image Source: <https://codelabs.developers.google.com/codelabs/cloud-micronaut-kubernetes/index.html?index=..%2F..index#9>

WHY MICROSERVICES

Based on information we learned in the course as well from the guest lecture Conor Gallagher's presentation, we found that microservices would be the best fit for our application. Microservices allowed us to easily deploy our backend application and access it via a RESTFUL API. Furthermore, as a platform for an ebusiness, an instance of our microservice could be easily used and expanded on. If more customers need our service, we could expand it accordingly

WHY MICRONAUT

We choose micronaut mostly because our backend developer has experience working with it to create microservices. Furthermore, Micronaut is a leading framework for developing low memory usage microservices as mentioned by Conor Gallagher. Even though micronaut is not close to the center of the tech radar currently, we feel that exploring new technology would be the best use of our time.

HOW WE GENERATE OUR PARTIAL PASSWORD

Our backend service generates by taking the AES encrypted password and randomly selecting 3 indexes from the password and then sending those indexes to the frontend to verify. These means that the only information that is being sent from the backend are the indexes of the password rather than the whole password. Our service is also able to generate new partial password every time the user logs in. The following code displays how we accomplished this:

```
public PartialPasswordResponse generatePartialPassword(String userName){
    PartialPasswordResponse response = new PartialPasswordResponse();
    if(userName != null) {
        response.setUserName(userName);
        ArrayList<Integer> tempArray = new ArrayList<>();
        for (int i = 0; i < AES.decrypt(userHashMap.get(userName).getPassword(), testKey).length(); i++) {
            tempArray.add(i);
        }
        Collections.shuffle(tempArray);

        StringBuilder indexes = new StringBuilder();
        for (int i = 0; i < 3; i++)
            indexes.append(tempArray.get(i));
        char tempStringArray[] = indexes.toString().toCharArray();
        Arrays.sort(tempStringArray);
        response.setIndex1(tempStringArray[0] - '0' + 1);
        response.setIndex2(tempStringArray[1] - '0' + 1);
        response.setIndex3(tempStringArray[2] - '0' + 1);
        return response;
    }
    response.setErrorMessage("No Username");
    return response;
}
```

API DOCUMENTATION AND DEPLOYMENT

It would be too long to explain the full functionality of all of our API endpoints. To find an in-depth exploration of our API, please visit the GitHub page for our project:

<https://github.com/engelsj/Ebusiness-Project>

To deploy our project we used a guide that can be found here:

<https://codelabs.developers.google.com/codelabs/cloud-micronaut-kubernetes/index.html?index=..%2F..index#9>

Because DevOps was not something covered in our course work, we do not have a deep technical knowledge on the inner workings of our Kubernetes's Cluster, however we used information from the course to choose our node route and location structure to optimize connection for Western Europe.

Reference: Week 2 – Entities, Servers, Connections

LIMITATIONS

DATABASE

Currently, our database is linked to the front-end. We had a number of difficulties connecting our database with the rest of our project, resulting in use storing some of our client information on our microservice. Given more time, we would have hoped to have the database interact with our microservices alone. At the moment, our database only runs on local host. A database hook-up would make this deployment ready, but we were unable to do this within the given time constraints.

CACHING VALUES

Our react app revolves greatly around using data input on previous pages to provide some functionality for other pages which involved looking into caching values. This was an issue we were dealing with for quite some time. Luckily we found a solution! For caching values to use across pages and class in our react app we relied heavily on the use of localStorage. LocalStorage is a web API that allows websites and apps to store data in the browser for later use. Unfortunately, this is not very secure and given more time this is definitely something we would have liked to find a different solution to solve our issue.

URL ROUTING

We would have liked to make it impossible for users to skip forward in the login process by changing the appended route path in the URL for our website. Due to the time constraints, we were unable to add this feature to our final product.

PUT OVER POST

When designing our Microservice, we mostly used POST requests. We understand the problems with POST requests as they not idempotent, which would allow the user to send multiple requests before their first one is processed. Ideally, we would like to have PUT requests on our microservice that have a client ID so that this is not a problem. Additionally, if we would have liked to looked into the POST requests with a client side ID that Stripe is developing that Conor Gallagher mentioned during his presentation.

DEPLOYMENT

When we went to deploy our microservice, we found that there were budget constraints that prevent us from running a lot of microservice through google cloud. To circumvent this, we somewhat broke traditional microservice architecture by having both the login and OTP endpoints within the same service. While both of these have their own unique endpoints, they are hosted under the same API. If we had more time and did not have the cost of cloud hosting to worry about, we would like to have had to different services hosted on Google cloud.

SSL AND CORS

We ran into an error that prevent our HTTPS microservice from properly communicating with our HTTPS Heroku website where our browser did not accept the SSL certificate that was generated from Micronaut. The workaround that we used was forcing the user to manually give permission to their browser to allow communication with our microservices HTTPS IP. In the future we would have liked to purchased our own SSL certificate that would be trusted by the user's browser.

CONCLUSION

Overall, we feel that our project has gone well. We were able to develop our knowledge by testing out various ideas and solutions. We now have a better understanding of the plethora of concepts that goes into developing a protected login service. If we were to do this project again, we would manage our time and tasks better and hopefully deliver a fully deployable project. We would like to say thank you to Dr Joeran Beel for teaching us about a variety of different services to do this. This project enabled us to expand our knowledge and put it into practice. We also appreciate the opportunity to publish all our hard work.