AmazonGeek

Ky Eltis

Digital Solutions IA2
Project – Digital Innovation
Hillbrook Anglican School
2021

Contents

EXPLORE

Page Summary

This page is the introduction to the problem and derivation & explanation of the solution. It explores the current context of how the problems came about, then itemises the problems. The target audience and user persona are identified as to understand who the solution is directed at. It looks at other solutions to similar problems, then develops a new solution.

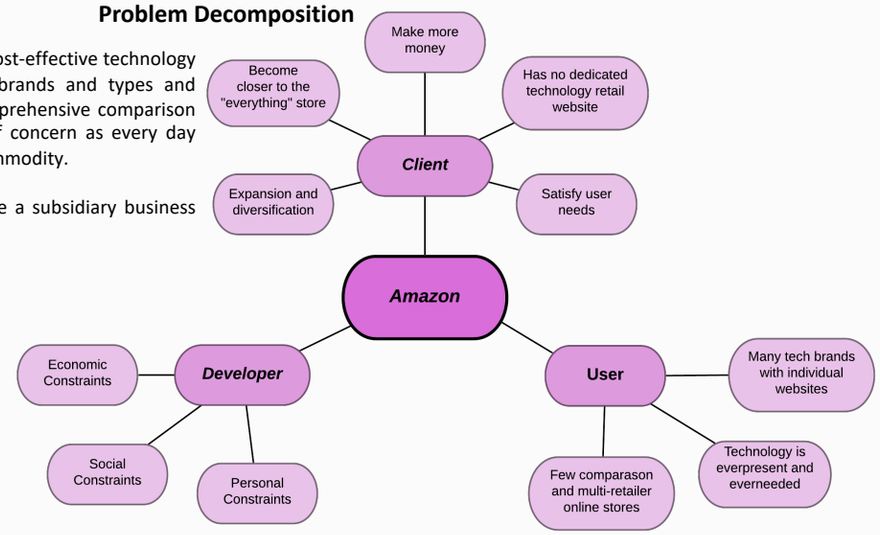
Context

In the fast-paced world of B2C (business-2-customer) e-commerce, the way consumers shop online is always changing. Just think about how different online shopping is now, than it was when Amazon launched back in 1994, or even how different things are now than they were 10 years ago. When Amazon started in 1994, it was a site that only sold books. Within a month of its inception, the company had already shipped books to over 40 different countries. Since then, Amazon has become one of the world's largest ecommerce companies, accounting for around 50% of total e-commerce sales in the United States for 2020, and 38% globally. As consumers' online shopping behaviours continue to change and new technology is introduced, the design of elements such as user experience (UX) and user interface (UI), need to be updated to match customer needs. Over the last 12 months, Amazon have received customer requests to expand their product base, range of payment methods and delivery options. To satisfy customer demands and maintain its competitive market edge, Amazon wants to grow and diversify its ecosystem.

Problem Decomposition

People nowadays have a very hard time buying cost-effective technology products. There is an enormous selection of brands and types and models out there and it is difficult to get a comprehensive comparison and analysis of all of them. This is especially of concern as every day technology is a becoming more and more of a commodity.

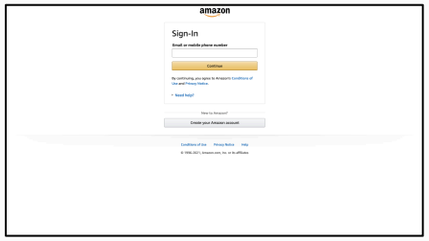
Amazon has approached the developer to create a subsidiary business technological model of a solution to this issue.



Existing Solutions

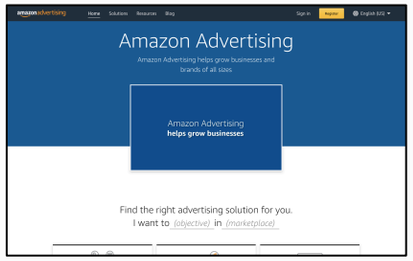
Amazon

The main amazon website. This will be used as a colour and style template for the solution.



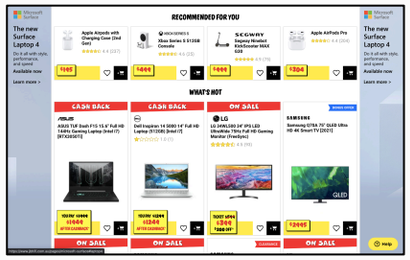
AmazonAdvertising

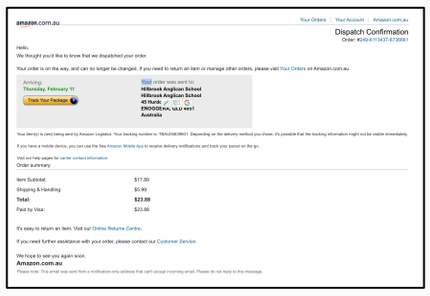
An example of how a team solved a similar problem posed by amazon.

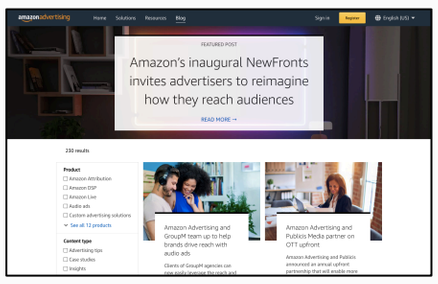


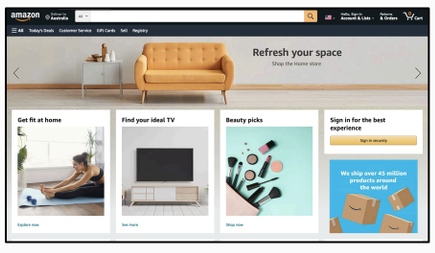
JB Hi-FI

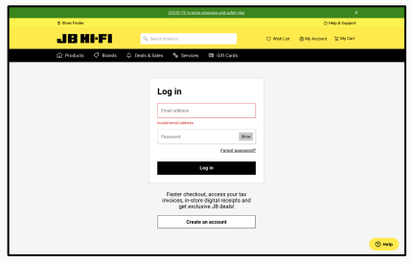
A technology competitor







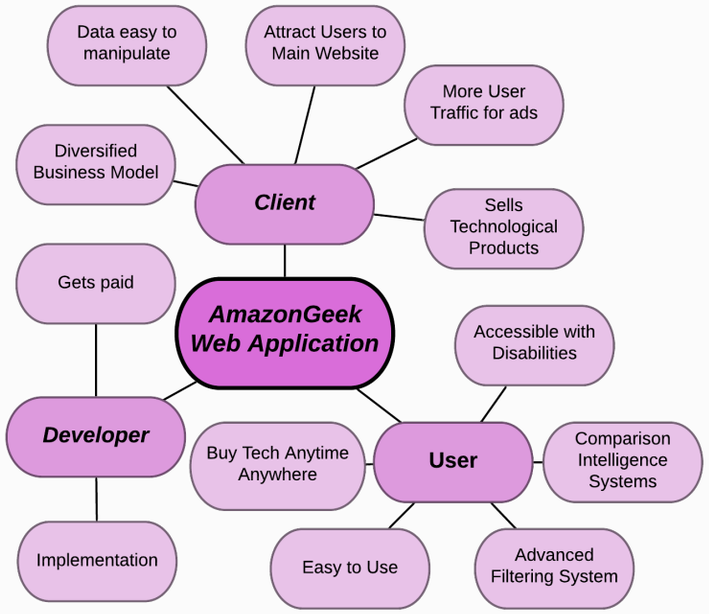




Company	Good	Bad	Learnings
Amazon (main website)	Sleek & Simple Good UX	No clear direction of where it wants the user to look or click	As amazon is the main client, the style and colour scheme will be similarly incorporated into the solution
AmazonAdvertising	Sleek and intuitive design	Menu system is complicated	This was to identify how similar the amazon website is to its subsidiary companies that have "amazon" in the title. The removal of the second nav bar detracts from the efficiency and clarity of navigation.
JB Hi-Fi (a technology retailer)	Great use of colour Streamlined forms	Harmony not well shown, the bright yellow and red sales images are eyesores and detract from the site	This was to identify another company's techniques to host a web application. The forms are very streamlined but the clutter in the bright images detract from the overall experience.

Rationale / Solution Decomposition

The solution to the posed problem was a web application called AmazonGeek. It would be an online platform where users could browse an extensive selection of technology from many different brands and categories. An advanced filtering system will be implemented to search for the exact items users are looking for. There would also be smart comparison systems implemented to assist with choosing between items. As a web application, the AmazonGeek platform can be used on virtually any internet-accessible device. The online technology store also has benefits of a clear hierarchy and classification of categories and types of devices for efficient filtering and clear page divisions. The developer was also very familiar with tech stores, both online and offline.



Target Audience

The target audience is the projected customer base of the web application. Characteristics the target audience are expected to have include:

- Aged 15-50
- Technologically-minded
- High usage of technology
- Familiar with online shopping
- Economically stable
- Uses technology for work

The user persona (right) were supplied by the client as a more specific focus to base the solution on. They are example potential customers and gives who they are and what they want from the solution.

Jayde

- Works for Scanlan Theodore as *eCommerce Manager*
- Lives a very busy life in Melbourne with her partner Axel
- Owner of JAM Design & Creative
  - Employs 2 people to -
    - create business logos and websites
    - implement tailored social media marketing strategies for business clients
- Needs alerts sent to her mobile if products in her *Amazon* wishlist go on sale

Ellesse and Haydn

- A young couple with a baby
- Own a 42-acre cattle farm in the King Valley gourmet region of north-east Victoria
- No mobile phone access, but NBN internet in their home office
- Frequent users of online shopping
- Mostly interested in email alerts for when 'out of stock' products on the Amazon website become available

Jeremy

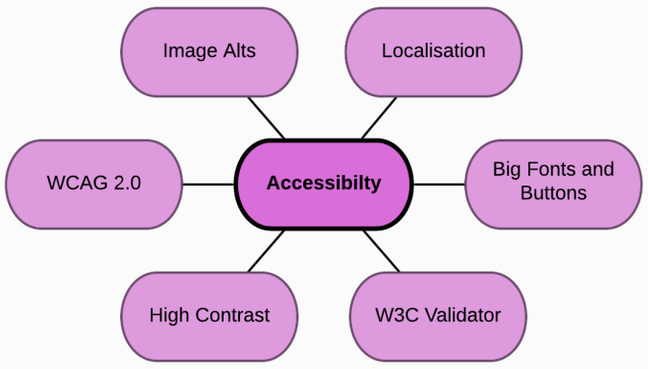
- Owner of Battman Personal Training
- Personal Trainer
- Interested in sports, fitness, functional training and health related products
- Frequent user of Facebook and Instagram for business purposes
- Alerts sent to his mobile/email when new fitness and health product are released

EXPLORE

Page Summary

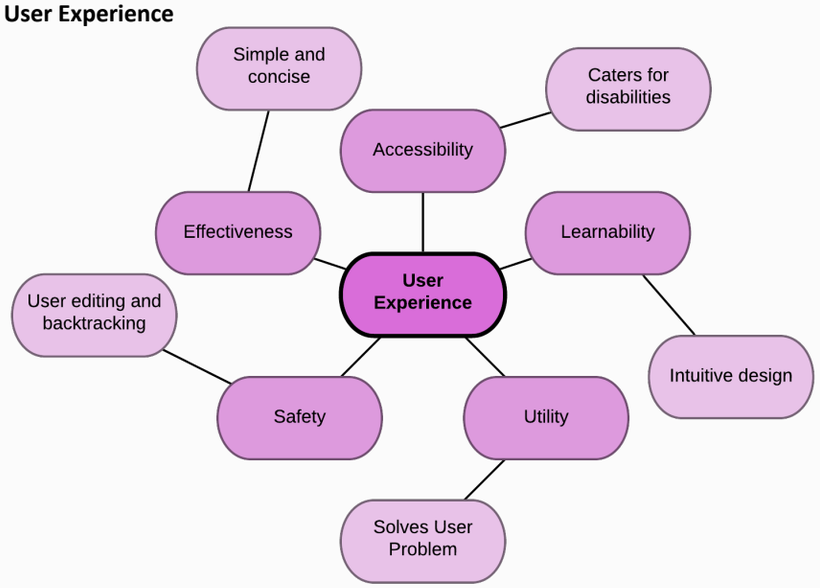
This page details a lot of the elements and principles of a good solution that will be referenced in future pages. It focuses on UX, UI, Security, and Accessibility. It also has a mind map encompassing all the Explore phase.

Accessibility



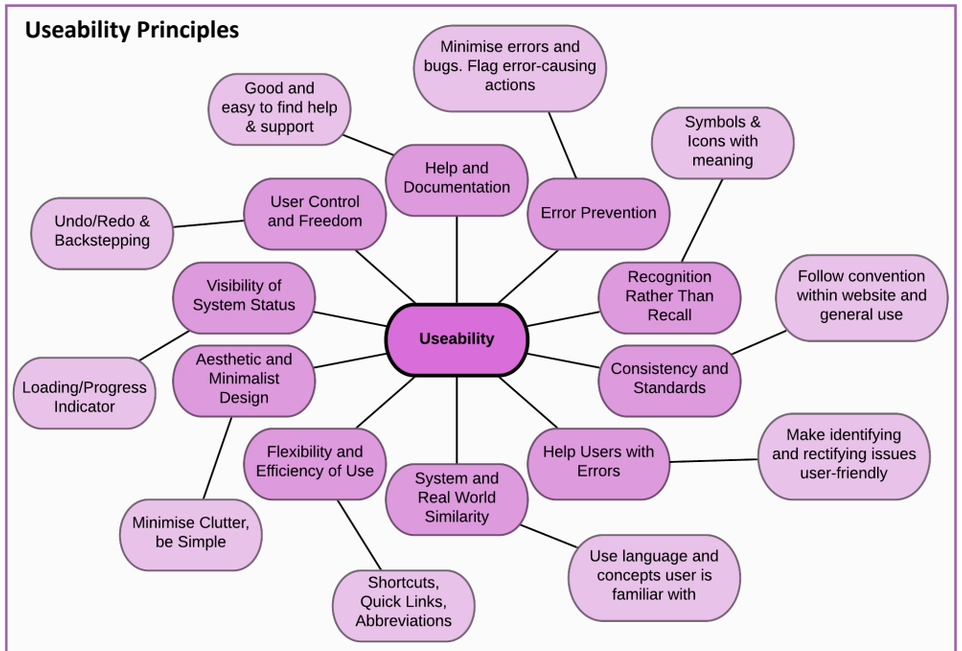
Accessibility was a very important value to consider in creating the web application. It allows a broader range of users to access the app. This was incorporated into AmazonGeek by implementing features such as localisation, image alts, and high contrast of text. The HTML of the website was also put through the W3C Validator to evaluate its ability to meet the WCAG 2.0 Accessibility Guidelines

User Experience



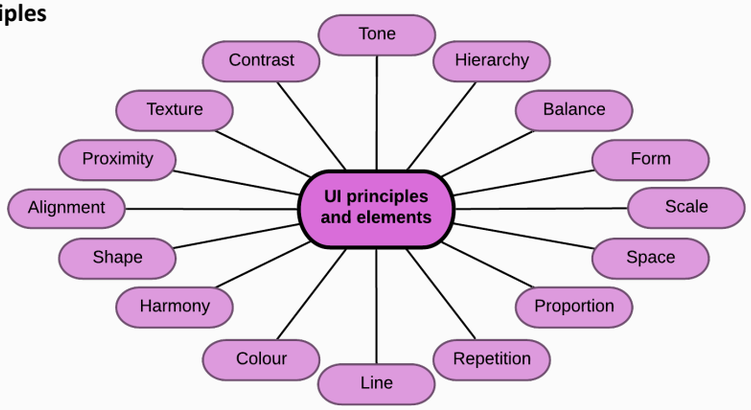
User experience is an important factor to incorporate into the design of AmazonGeek. It was used in the website by creating a purposefully simple and intuitive design scheme that can be used by anyone, regardless of technological skill. This incorporated the learnability and effectiveness UX principles. It was done by minimising the amount of buttons and text on the screen to the bare minimumz and by contrasting the important buttons a different colour and font.

Useability Principles



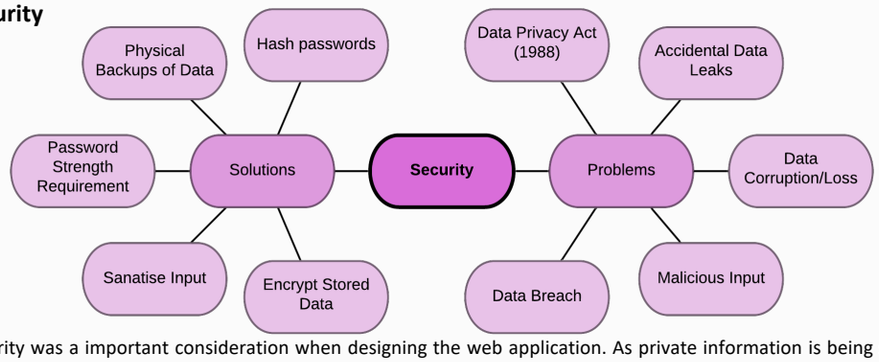
Useability principles are a part of UX and of great importance to the quality of the web application. It was used to make the web application easy and enjoyable. Examples are the errors in the application flashing a warning box in user-friendly language so they can mitigate minor issues themselves.

UI Principles

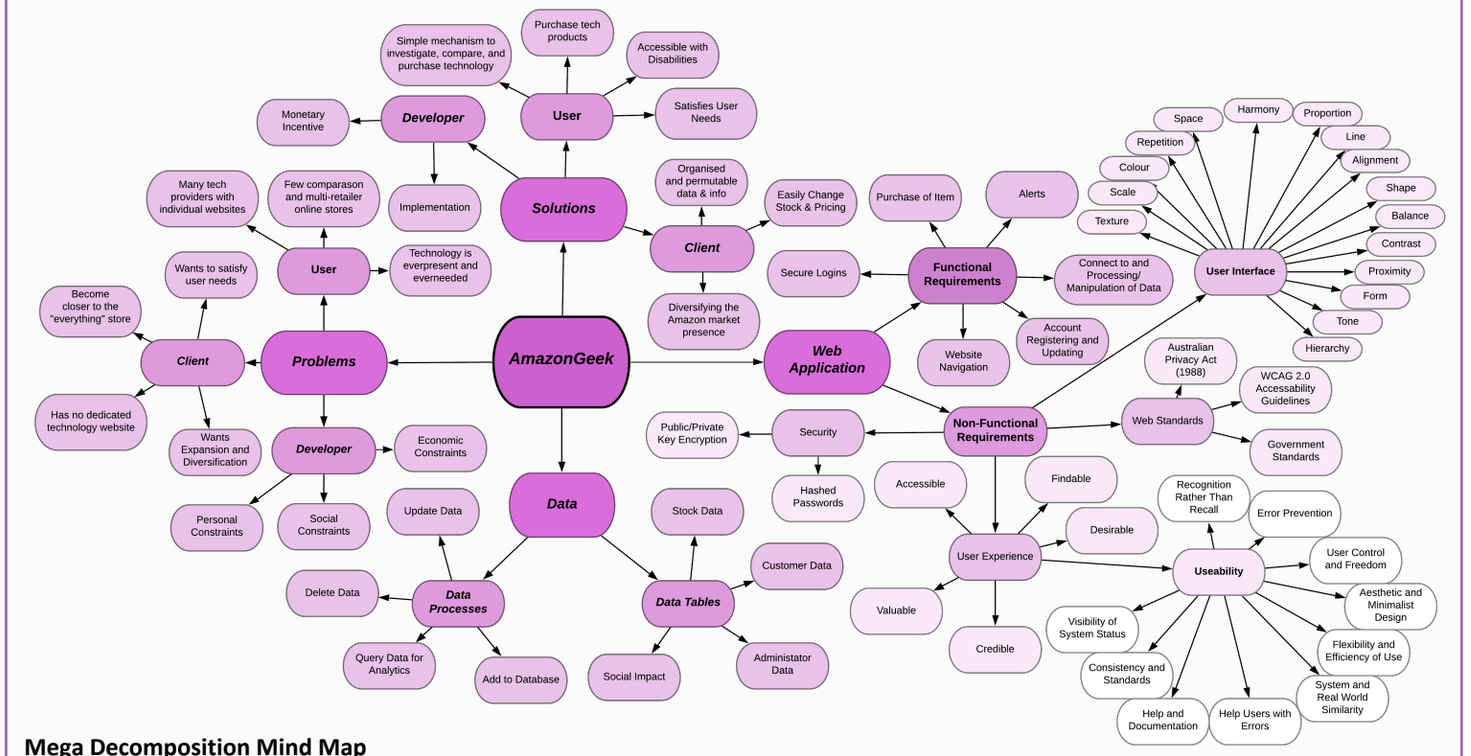


User Interface was a significant aspect of the design. The principles and elements of visual communication were used as they detailed how to create an aesthetic digital design. For example, the element of contrast was used to separate important text (like prices and special deals) and buttons from the rest of the body. This makes it easier to locate the important parts of the web application and minimises effort and time while using AmazonGeek.

Security



Security was a important consideration when designing the web application. As private information is being stored in the database, there are conventional and legal privacy requirements that must be met. Other security problems include loss of data or malicious attacks. These problems will be averted by incorporating encryption of all data, hashing and salting of passwords, sanitisation of customer inputs, and other methods.



Mega Decomposition Mind Map

This diagram is a culmination of the whole Explore phase of the project. It shows the interrelationships of the different mechanisms to produce the final product. In summary, the stakeholders (i.e. user, developer, client) have had their problems met with solutions that incorporate the use of data, and functional & non-functional elements of the web application.

EXPLORE

Page Summary

This page details the scope of the solution. It discusses the requirements, criteria, constraints, and resources for the project. This is a very important page as it says exactly what is needed to be done, and the types of things to focus on to make a successful and high quality solution.

Solution Requirements

To identify the scope of the project and to figure out what needs to be implemented, the client supplied a set of solution requirements for the project. It consists of both functional and non-functional elements. Any elements that are not implemented will be discussed in relation to the constraints and future improvements in the Evaluate phase.

Functional	Non-Functional
Data such that: - Administrators can easily create, upload and maintain the concept shop data on the website. - Administrators can upload product data to the website from the supplied CSV file. Code for: - Web-based digital solution - Database and tables - Search and display a specific product - Search and display by product category - Read records from a CSV file and process them for storage in a database - Create a secure password based on criteria - Shopping cart transaction processing - Pay for goods or services - Calculate shipping - Calculate total cost - Product inventory management - Product quantity is updated when a product is sold - Shopping cart Code such that: - An incorrect user registration will not be stored in the database - Customers are able to register for an account, setting their own username and password - Registered users are able to login to their account using a username and password - Securely store the personal details and passwords that customers register with the site - Save the types of alerts pre-selected by a customer - Customers are sent alerts either by email or SMS - Customers can add to cart, edit cart, save cart, continue shopping, select from a range of payment and delivery options	User Interface/User Experience: - The application must have a responsive web design. Visual Components for: - Customer account registration - Customer account login - Alert notifications – email alerts about sales, restock item alert, wish list offers - Wish list - Admin account login - Data upload (admin) - Data maintenance (admin) Must comply with: - Government web design standards - The Australian Privacy Act (1988) - Web Content Accessibility Guidelines (WCAG 2.0) - Copyright Law - Useability Success Criteria - Elements and Principles of Visual Communication

Developer Resources

Software	Hardware	Knowledge & Skills
- Adobe XD - MariaDB - MySQLWorkbench - Google Suite - Microsoft Office - Visual Studio Code - Lucidchart Website - Udemy - Grok Learning - Python Shell	- Apple MacBook Air (2017)	Needs to code in: - SQL - HTML - CSS - Python (Flask, sqlalchemy) Proficiencies in software and hardware given.

The developer was given a certain set of software and hardware by the client to use. The developer had to learn many new software and coding languages and packages to complete this project.

Evaluation Criteria

To evaluate the success of the project in the Evaluate phase, evaluation criteria were created. These were both given by the client (prescribed) and made by the developer (self-determined). These criteria assess the quality and functionality of the project.

Prescribed Criteria
Developed technical proposal for a web application
Low-fidelity prototype solution
Relevant user interfaces and algorithm component of the prototype digital solution
Administrators able to easily create, upload and maintain the menu data
Customers able to create accounts and purchase drinks
Upholds usability principles, including accessibility, effectiveness, safety, utility and learnability.
Explores the interrelationships between user experiences and data
Identification of errors to yield refinements
Ability to upload data via CSV file uploads
Compliance with the Australian Privacy Act 1988 and Australian accessibility guidelines

Self-Determined Criteria
Aesthetically pleasing design
Efficient navigation of website
Logical and intuitive buttons
Intuitive and clear information and instructions
Attractive and desirable User Interface
Appeals to user persona
Usable for everybody, including those with sensory or cognitive impairments
Works on a laptop
Minimalistic and sleek Design
Maintain a consistent and harmonious style guide throughout
Show status of system wherever possible

Constraints

Personal	Social	Economic
- Limited skills of developer - Limited knowledge of developer - Low-Fidelity prototype	- Follows accessibility guidelines in WCAG 2.0 - Follows Data Privacy Act - Useable by anyone, including people with limited technological competence and disabilities. - Legal usage and attribution of images and documents	- Money drives Personal Constraints - Limited tools and resources - Time limitations

The developer had many constraints upon them that inhibited their ability to complete the solution requirements. One major limitation was the initial lack of coding experience in the particular field of databases and web development. The developer had to learn SQL, advanced Python, and many specific Python packages such as flask, flask-sqlalchemy, and wtforms.

Scope

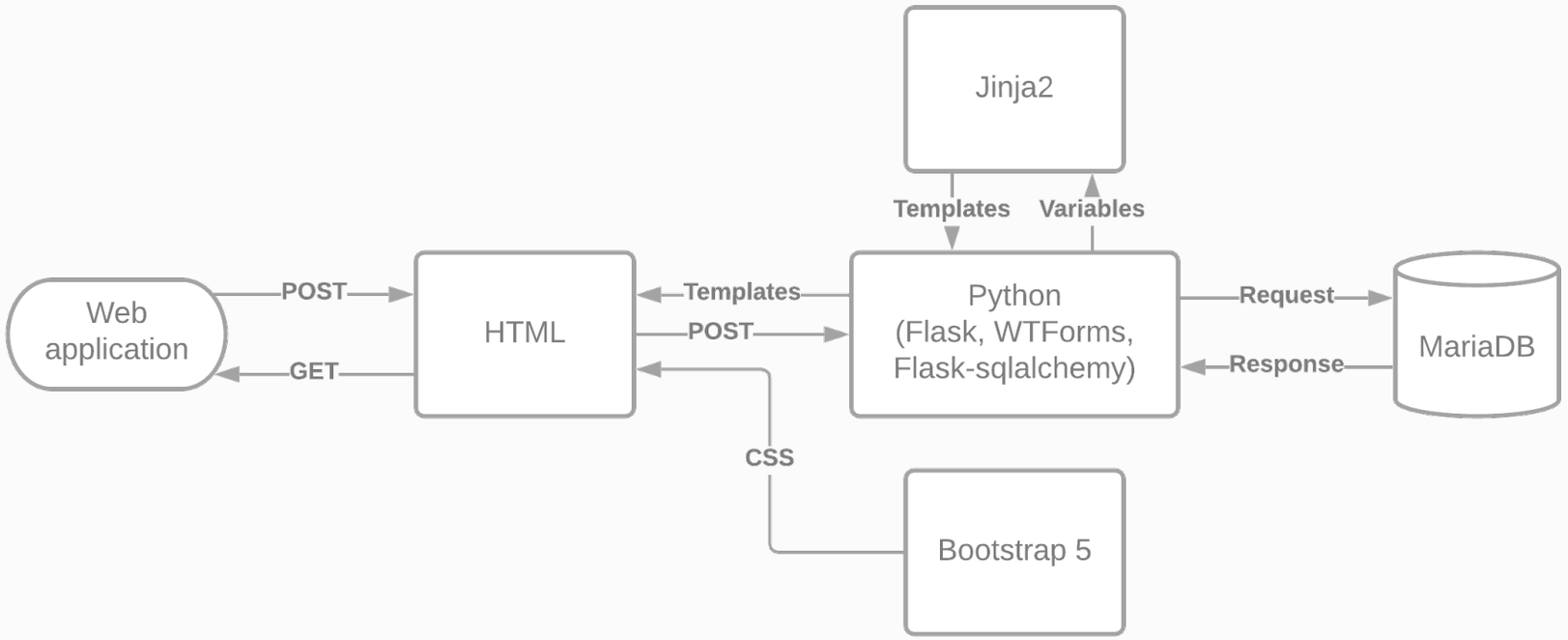
The scope of the project is the extent to which work has to be done. This is a prototype project and will not have all the full functionalities of a fully realised web application. The constraints also limit the extent to which this web application can be made. As the developer is unfamiliar with the content and the time and software restrictions are very tight, the scope of the project was very small, to the point of becoming a proof of concept.

DEVELOP

Page Summary

This page comes in two parts: The software framework, and the UI component of the solution. The software framework details how libraries and languages interact together to create the web application. The UI components focus on the overall design scheme of the website and how to use the principles and elements of visual communication to produce an effective aesthetic design. Responsive design to support different viewports is also considered.

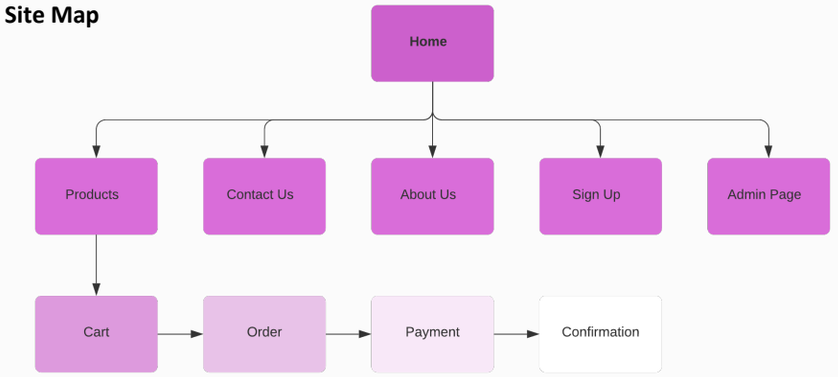
Software Framework



Overview: Python Flask library creates a web app that interacts with HTML via Jinja templating and interacts with a MariaDB database with the flask-sqlalchemy Python library.

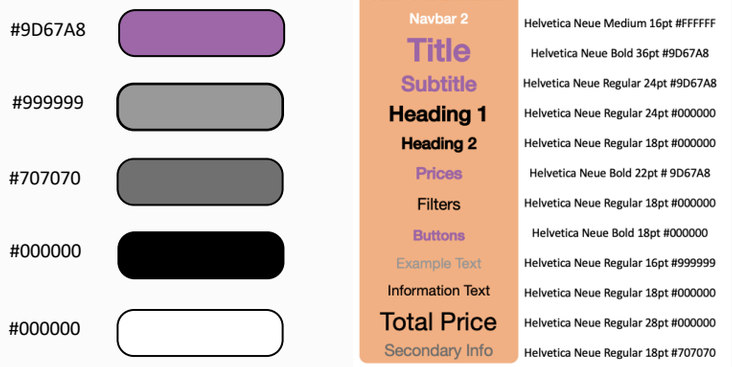
To create a web application that can interact with user input and a database, many different software solutions were used. On the front end, the compiled and styled HTML is what the user's browser reads to display the page. Bootstrap is also involved to format and insert smart CSS. The HTML and CSS comes via a GET request to the web server. The web server provides the HTML based off a Jinja2 template linked through Python. The Jinja2 template is written in HTML with modifications which remove the need to repeat html headers and incorporates the ability to pass in Python variables. The Python library running the application is Flask. It connects with the Jinja and HTML directly. In order to have sets of data in the application, MariaDB was used as a database system. To connect the Flask app to the database, the flask-sqlalchemy Python library was used. This allowed direct SQL code to be executed on the database. Forms in the application send data from the browser via HTML back to the Flask app where it can be manipulated into changing the database or the Jinja templates..

Site Map



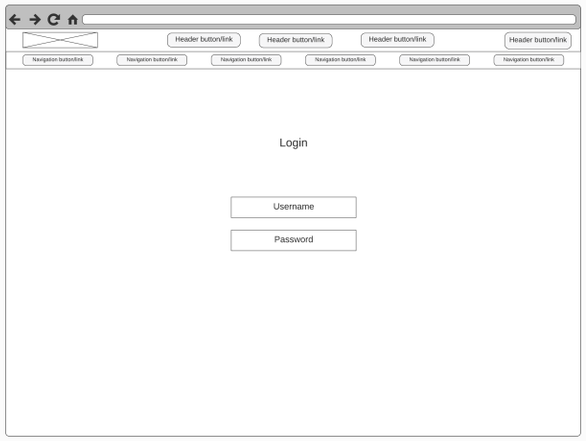
The site map is an overview of the page navigation of the web application. It was chosen to be like this as an effort to use the UX principles of efficiency as the functional process (buying a product) is very streamlined and linear path.

Style Guide



The style guide is to ensure consistency in colour and font scheme. Consistency is important as it allows users to recognise rather than recall important information. E.g. you don't need to remember what the button is called or where it is on the page because its contrasting font and colour will reveal it. Additionally, the colours and fonts chosen follow the visual communication elements and principles to be an aesthetic design. They are kept streamlined and basic as to not overwhelm or distract the user, as witnessed in the JB Hi-Fi example solution.

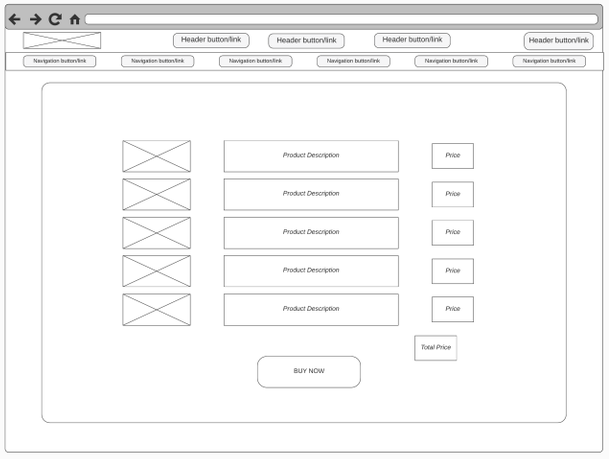
Wireframe – Login & Navbar



Wireframes were created to design the overall look and placement of elements. The navbar was designed like this to be both streamlined and very functional. The first bar will have the main link to different pages, like the Contact Us page, Login page, Home page, and Products page. The second bar was for specific product pages separated by category. This enables quick and direct access to exactly what the user wants.

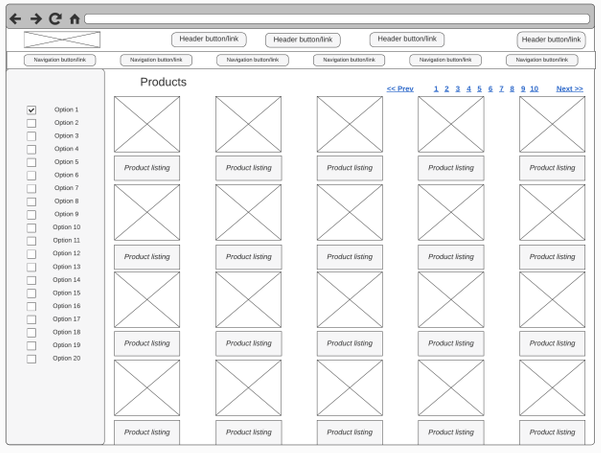
As for the login, it was kept to a very minimal UI. This is to remove extraneous information that is not needed and to focus the user's attention on the important things.

Wireframe – Shopping Cart



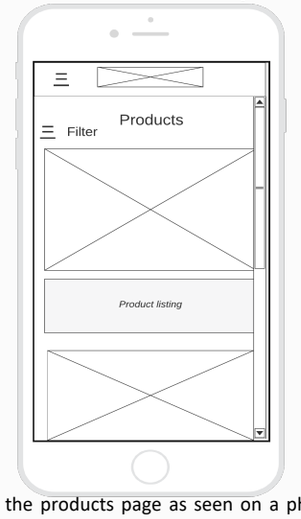
The shopping cart was also kept as skinny and streamlined as possible. It clearly lists your cart and shows a big contrasting button to submit. There is also respective images for all the items so the users can recognise the items rather than recall them.

Wireframe – Products



The products page might seem cluttered but the images are largely whitespace and decently spaced. There is a filtering system on the left to make browsing and searching for products easy and intuitive. The products and their descriptions are aligned in a grid formation so it looks uniform and aligned. There are pagination controls at the top of the page to see more products if the user desires.

Wireframe – Alternate viewport



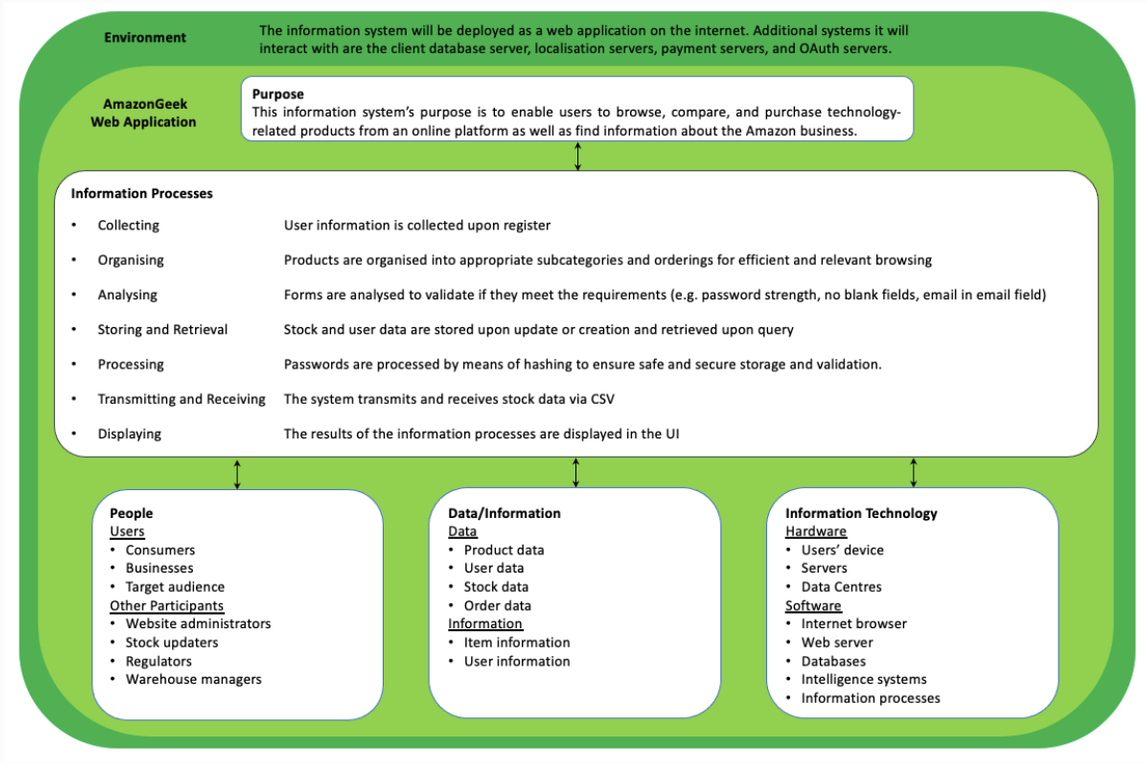
This is the products page as seen on a phone. As phones are a major viewing platforms for users, it is important that AmazonGeek caters for it. This was done by using responsive design, incorporated by Bootstrap 5. This turns large menus into dropdowns with the hamburger icon. The products are also resized to fit a phone screen and be easily readable.

DEVELOP

Page Summary

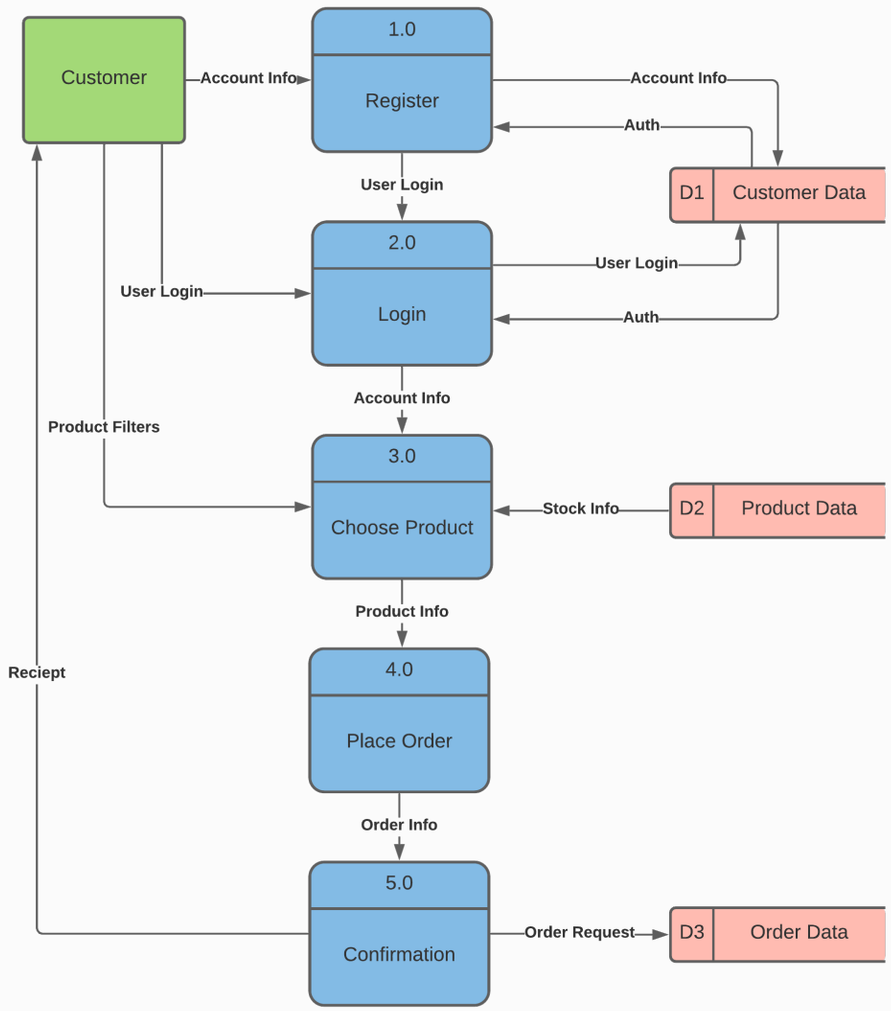
This page looks at the data flow around the website. This is throughout the overarching ISIC diagram as well as the three DFDs. The ISIC diagram expressed a high level overview of how the web application interacts with itself and with external entities. The DFD Level 0 provides a closer look at what data goes in and out of the website. The other DFDs follow the data flow of all interactions and processes a type of user can have.

Information Systems In Context Diagram



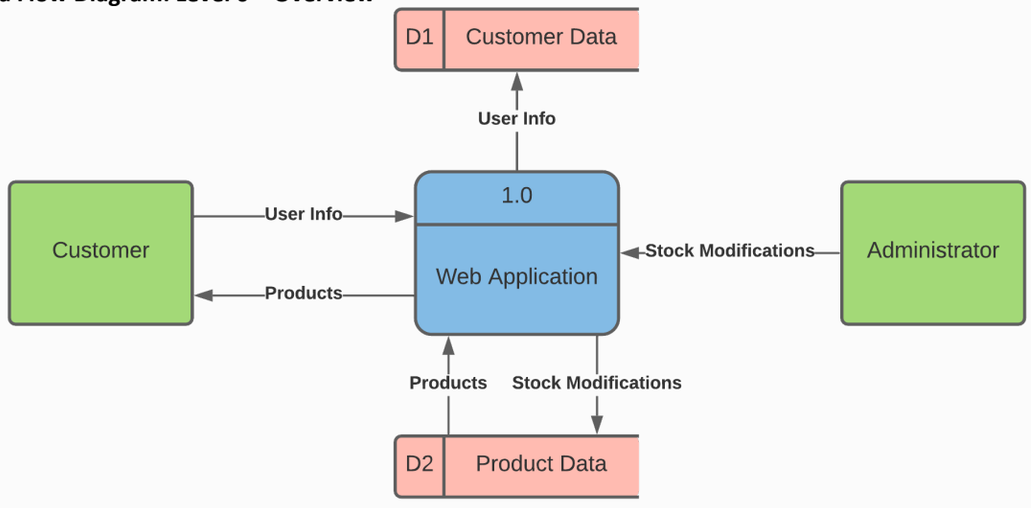
The ISIC diagram above details how data is manipulated in the web application. It shows the environment and outside interactions of the application and how it responds using internal information processes. A summary is that the users and admins use this site to access and manipulate the data and information. For the users that is via a layman UI. For the administrators, it is purely tables and forms. This is managed by the information technology and information processes

Data Flow Diagram: Level 1 – Customer



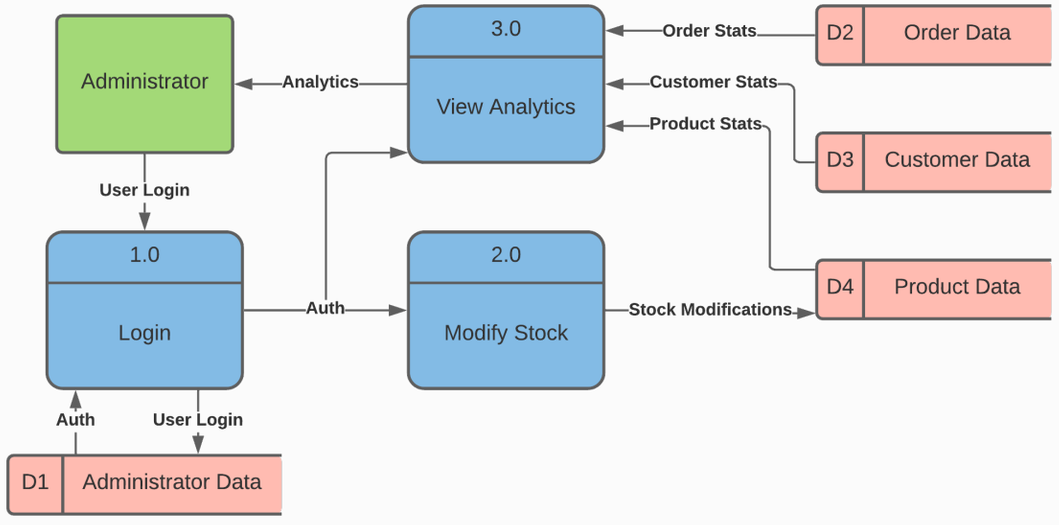
This is the data flow diagram for a customer user. It details how their need is solved. i.e. all the processes from registration to confirming an order.

Data Flow Diagram: Level 0 – Overview



This is an overview of the data flow around AmazonGeek. The administrators manipulate the product data which is viewed by customers as products to be ordered. This solves the user problem as they can order products and solves the client problem as it allows them to modify the data.

Data Flow Diagram: Level 1 – Administrator



This is the data flow diagram for an admin user. It details how their need is solved. i.e. all the processes from logging in to viewing and manipulating the data.

DEVELOP

Page Summary

This page details the sample data used in AmazonGeek. The ERD explains the relationship between the tables on the rest of the page. The tables include the supplied tables, amended tables, created tables, and all the respective data dictionaries. The created tables were the order and the product_order tables that were derived from the supplied receipts, seen below. All tables were created at normalisation level 3 thus no further normalisation was required. All sample data passwords were converted into hashes for security and matching with the database.

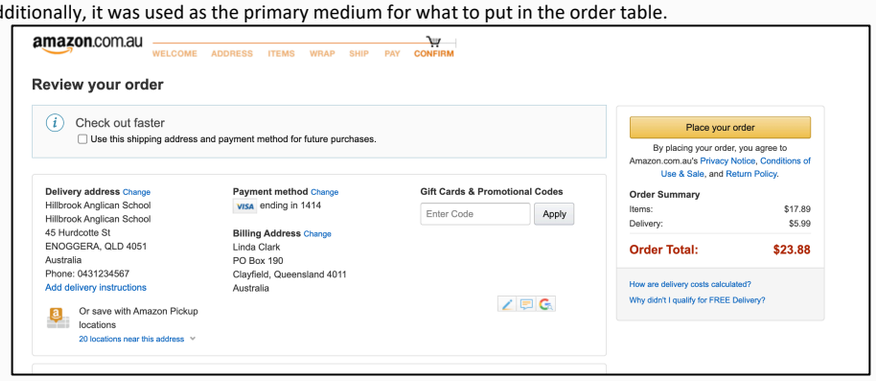
Data Types

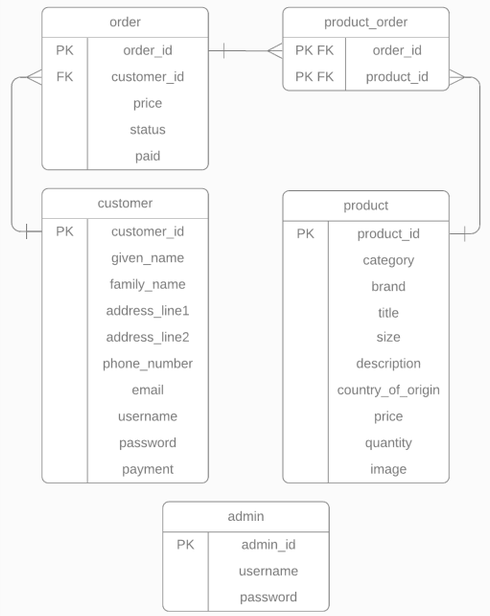
Data Format	What is it?	What is it used for?
Integer	A numeric object with no fractional part	IDs, amounts of things
VarChar	A generic alphanumeric object	Text, variants, item names
Float	Numeric object with a fractional part	Prices, exact values, operable numbers
Boolean	A binary true/false object	To test conditions

This lists the type of data values that this solution will incorporate and their usage.

Supplied Receipts

These are receipts directly from Amazon, the client company. They were used to get an insight into Amazon's database system so AmazonGeek's could match similarly. Additionally, it was used as the primary medium for what to put in the order table.





Entity Relationship Diagram

To the left is the ERD of the database used in AmazonGeek. It comprises of the (amended) supplied tables as well as two created ones. The order and product_order tables were creates as a means of connecting the customer and product tables via orders. The customer can place many orders, but each order can only come from one customer. Each order can have many products and each product can be in many orders. This results in a many to many relationship which is difficult to manage in the database. To mitigate this, an intermediate table, product_order, was created.

The supplied and created tables were already in 3rd normal form as there were only atomic values (one value per field), there were no double-up columns, and all columns purely depended on the PK of the table. This meant that no normalisation took place.

Admin Table

The supplied data (left) was amended by adding an id primary key column. It stores the admin login details.

username	password
Admin1	123456
ADMIN-ALL	password
Admine2	mycatsname

admin_id	username	password
0	Admin1	tbisdufhu…
1	ADMIN-ALL	873ub2ib…
2	Admine2	f28dfguw…

Field Name	Data Type	Size	Example Data
admin_id (PK)	Integer	8	101
username	VarChar	32	Admin1
password	VarChar	256	6y4hhfgk37…

Order Table

This table was created to allow customers to purchase products in an order. It was created based off the supplied receipts (see above) and intuition for what an order might entail.

order_id	customer_id	quantity	price	status	paid
192387	101	1	821.52	processing	TRUE
9736	104	7	1,052.62	payment	FALSE
298356	102	16	689.15	in delivery	TRUE

Field Name	Data Type	Size	Example Data
order_id (PK)	Integer	8	101
customer_id	Integer	8	104
quantity	Integer	8	7
price	Float	16	821.52
status	VarChar	32	In delivery
paid	Boolean	1	TRUE

Product_Order Table

This table was created as an intermediary table for the many to many relationship of the order and product tables.

order_id	product_id
192387	101
9736	104
298356	102

Field Name	Data Type	Size	Example Data
order_id (PK FK)	Integer	8	101
product_id (PK FK)	Integer	8	412

Customer Table

The top table is the exact supplied data for the customer table. This was amended slightly to produce the bottom table. The reasoning behind this was to simplify and clarify field names and fields. Examples are: "customer_ID" to "customer_id", keep lowercase consistent; "email_address" to "email", simplify name; and "mobile_phone_number" to "phone_number", simplification.

Field Name	Data Type	Size	Example Data
customer_id (PK)	Integer	8	101
given_name	VarChar	32	Jayde
family_name	VarChar	32	Grant
address_line1	VarChar	64	1569 Boggy Creek Road, Myrrhee, VIC
address_line2	VarChar	64	45 William Street, Melbourne, VIC, 3000
phone_number	String	16	0491575254
email	VarChar	64	jgrant@gmail.com
username	VarChar	32	Jammy
password	VarChar	128	fd3g45y5eytt434tyr456756y56y456756y456hhfgk37…
payment	VarChar	32	Visa

customer_ID	given_name	family_name	address_line1	address_line2	mobile_phone_number	email_address	user_name	password	payment
101	Jayde	Grant	JAM design and creatives	45 William Street, Melbourne, VIC, 3000	0491 574 118	jgrant@gmail.com	jammy	123456	Paypal
102	Eliesse	Grant	1569 Boggy Creek Road, Myrrhee, VIC		0491 574 632	egrant43@gmail.com	egg-hay	haydn	Visa
103	Haydn	Eagle	1569 Boggy Creek Road, Myrrhee, VIC		0491 575 254	soaringeagle@hotmail.com	soaring eagle	123456	Paypal
104	Jeremy	Smith	29 Stanley Terrace, Brighton, QLD, 4017		0491 575 789	jezsmith125@telstra.com.au	jeremy_smith	$ecureAspo$$ible password	Paypal

customer_id	given_name	family_name	address_line1	address_line2	phone_number	email	username	password	payment
101	Jayde	Grant	JAM design and creatives	45 William Street, Melbourne, VIC, 3000	0491 574 118	jgrant@gmail.com	jammy	jbsduhfosubp9w8rp9vqui…	Paypal
102	Eliesse	Grant	1569 Boggy Creek Road, Myrrhee, VIC		0491 574 632	egrant43@gmail.com	egg-hay	45htrgfbvfj56haydnhfghh…	Visa
103	Haydn	Eagle	1569 Boggy Creek Road, Myrrhee, VIC		0491 575 254	soaringeagle@hotmail.com	soaringeagle	3jgi9bg2ejbogb12345i6g3…	Paypal
104	Jeremy	Smith	29 Stanley Terrace, Brighton, QLD, 4017		0491 575 789	jezsmith125@telstra.com.au	jeremy_smith	2g80475g97fsyeu2fuhd2b…	Paypal

Product Table

This table was supplied and minorly amended. The original supplied csv had "product_ID" instead of "product_id" and the prices were preceded by a "$". It was changed to be more consistent and for the price to be more easily manipulated.

Field Name	Data Type	Size	Example Data
product_id (PK)	Integer	8	101
category	VarChar	32	audio
brand	VarChar	32	Apple
title	VarChar	64	Playstation 5
size	VarChar	32	13 inch
description	VarChar	256	Advanced keyboard for gaming
country_of_origin	VarChar	32	USA
price	Float	16	569.00
quantity	Integer	8	6
image	VarChar	64	garmin_watch.jpg

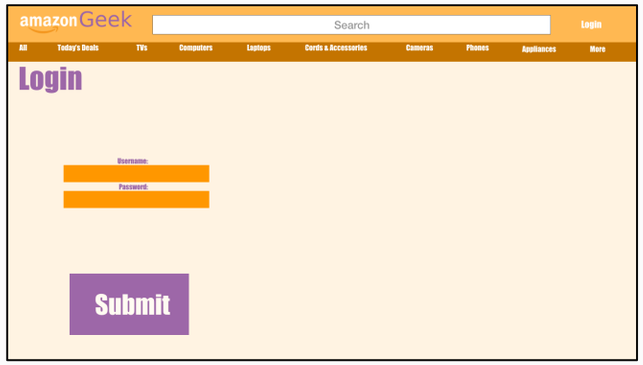
product_id	category	brand	title	size	description	country_of_origin	price	quantity	image
66890456	computers and tablets	Apple	MacBook Pro	13 inch	Laptop computer. 1.4GHz i5 512GB (Space Grey) [2020]	USA	2069.10	6	13inch_apple_mac.jpg
77895643	computers and tablets	Apple	MacBook Pro	16 inch	Laptop computer. 2.6GHz i7 512GB (Space Grey) [2019]	USA	3419.10	1	16inch_apple_mac.jpg
78904534	audio	Sony	Overear headphones		Noise Cancelling Processor, light-weight design with improved fit and comfort	Malaysia	395.00	40	sony_headphones.jpg
78945672	mobile phones	Samsung	Note20 Ultra		Android smartphone with 108MP Triple rear camera with Laser auto focus, 50x space zoom & 8K video recording capture, responsive S Pen for scribing.	Vietnam	1999.00	0	samsung_phone.jpg
10023445	health and fitness	Garmin	VivoActive 4 GPS Smart Watch		All-day health monitoring features, Stay connected with smart notifications for incoming calls, text messages, calendar reminders and more (Slate/Black)	USA	569.00	10	garmin_watch.jpg
10026789	gaming	Razer	Mamba Wireless Gaming Mouse		Mouse, designed to deliver long-lasting gameplay with zero restrictions.	Singapore	184.00	2	razer_mamba_mouse.jpg
10026790	gaming	Razer	BlackWidow Elite Keyboard		Advanced keyboard for gaming	Singapore	299.00	7	razer_keyboard.jpg
7378786	gaming	Sony	Playstation 5		PS5 games console, delivering awesome gaming experience.	Japan	699.00	0	playstation5.jpg
7356034	television	Samsung	TU8000	65 inch	Crystal UHD 4K Smart TV	South Korea	1495.00	3	samsung_television.jpg

GENERATE

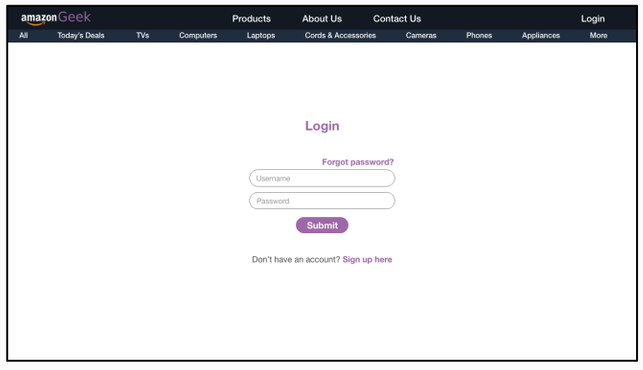
Page Summary

This page details the generated prototype web application as well as the finalized design that is yet to be implemented. The UI designs were make in Adobe XD and used many UX and UI principles in its creation in order to be the best it could. There was also a stage of refinement to get to the final UI, as seen below. These UI mockups are how the web application will look in the future, but for now, the actual working prototype is displayed on the right. It was not made for aesthetic design, only for functional elements.

Prototype Design – Refinements & Login

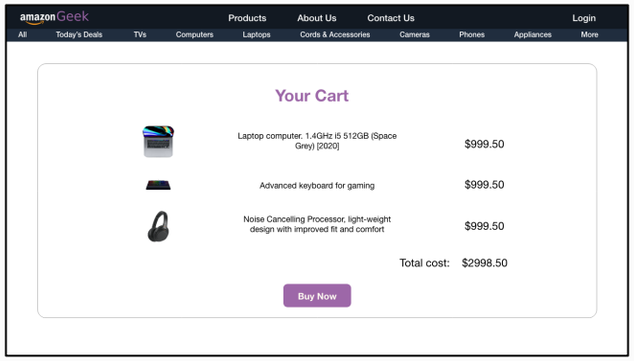


This was the first design for the web application. It attempted to use the logo colours and have interesting and eye-catching fonts. However this style is outdated and the modern styles are a lot more streamlined and sleek. The fonts should be easily readable as their primary goal. The search bar is a good idea but it takes up too much space and with the levels of filtering; it is not needed. The whitespace in the navbar also detracts from the contrast to the body content. The login form is off-centre and distasteful. These were replaced with a much more contrasted, modern, sleek, and intuitive design:



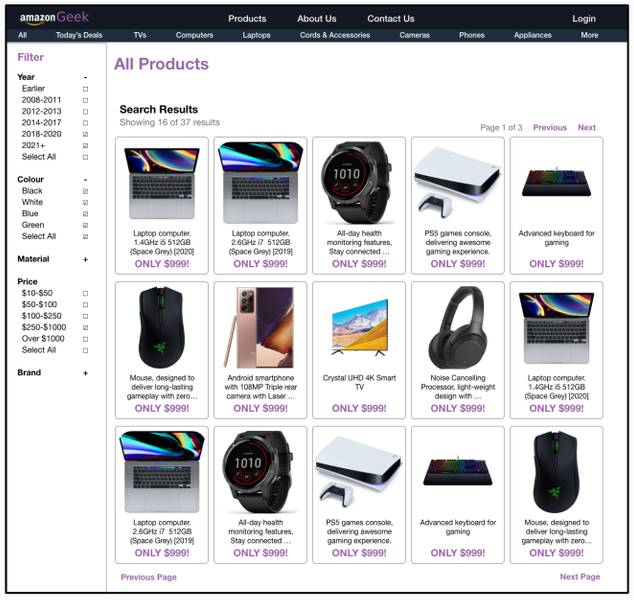
Here the UI is very streamlined and purposeful. Everything has its place and meaning. Things are unobtrusive and contrast is used to highlight important information, such as headings or buttons. The contrast is achieved without overuse of bright colours as to not overwhelm the user or detract from the overall experience. The login form is centred and the proximity makes them behave like one cohesive idea. The sleek and modern design makes great use of whitespace. It leaves out unnecessary information to focus the users attention solely on the important and functional things.

Prototype Design – Shopping Cart



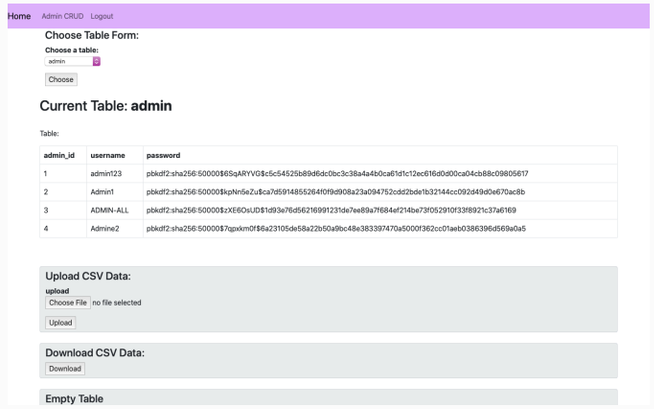
The shopping cart UI above was chosen with consideration from the elements and principles of visual communication. The container for the whole form assists with making the form feel like it have a close proximity and establishes a clearly defined area for one 'idea' of the website. The heading was big, bold, and in a different colour as to show contrast and direct the users attention to it so they are aware of where in the web application they currently are. It also adheres to one of the WCAG 2.0 accessibility guidelines about always having page headings on every major page. The image of the items was used to invoke the 'recognition rather than recall' useability principle, allowing users to identify the product without having to remember what it is just from the title description.

Prototype Design – Products Page



The design for the products page was made to be as streamlined and succinct as possible. This was to make the design learnable and intuitive. Things are clearly separated and grouped by containers and proximity. The eye-catching bold contrasted prices are purposely made to intrigue the user and make them click on it. The filter system on the left was developed to be a very simple, yet powerful tool. The products are aligned in a grid formation for alignment and there are pagination tools to access further pages.

Implemented Solution



This is an excerpt from the implemented functioning solution. This is not the final UI but rather a prototype designed to meet the functional requirements. The pages shown are the admin-login and admin pages. These allow administrator uses easy access to the database and implements tools to manipulate the data. These tools include forms for uploading a CSV into the database, adding new rows, deleting rows, emptying the table, reading the whole database as a table, querying the data with specific requirements, and downloading the selected table as a CSV. These forms were **fully implemented in code** (as seen in the algorithms document) to function and work in all circumstances. The forms actually manipulate the MariaDB database.

Meeting User Needs

One of the key criteria for a good solution is the ability to solve the user problem. After all, it is its the primary purpose. The identified user problems were:

- Many tech brands with individual websites
- Technology is ever-present and ever-needed
- Few comparison and multi-retailer stores

These were addressed by the solution in a few ways. The first problem is solved by the core design of the web application. It was designed to have a wide range of products from a wide range of brands and companies. It tries to act as a hub for all different technologies in many ranges. The second user problem was similarly answered. The extensive collection and the super-cheap prices offer an easy solution to the increase in technology demand. As for the comparison part, yes many retail companies were included in the products, a comparison system was not yet implemented.

Additional to the general user problem. There were also user problems raised by the user persona. They all had to do with alert systems which were not implemented into the solution.

EVALUATE

Page Summary

This page details the evaluation of the solution. The primary source of evaluation were considering the criteria and requirements laid out in the Explore phase. The solution requirements evaluates the content of what the solution should be. The prescribed and self-determined criteria discusses the UI and UX of the solution. Overall the solution was evaluated to be effective but lacking in completion. Additionally, the solution was tested by test users and feedback was recorded. The use of the W3C Validator and a reflection of the impact of constraints were also detailed. This was then summarised into the future recommendations and overall evaluation of the solution as is.

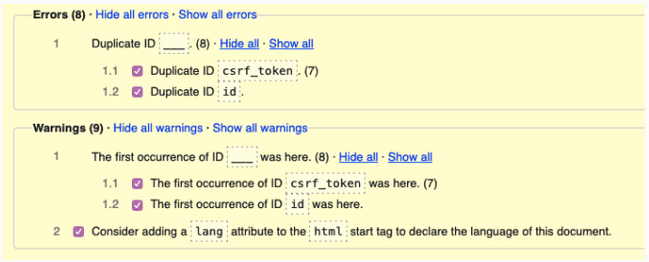
User Testing

		Best				Worst
Topic	Question	5	4	3	2	1
Suitability for target audience	Does the website represent the client and their needs?	XY	Z			
	Is it suitable for retail purposes?	X	Y	Z		
Suitability for purpose; informative and point of contact	Does the website showcase a range of technical skills?	YZ	X			
	Is the website engaging for the user?	Z	XY			
	How would you rate this website compared to others?	XY	Z			
Appearance; layout, colours, effects, creative ideas	Are background and image colours used effectively?	XY			Z	
	Are the website and logo appealing?		XY	Z		
Technical features - images and effects	Does the website look complete and /or professionally created?		XY	Z		
	Is the website and logo well designed and professional looking?	X	YZ			

Tester	Comments
X	This is fabulous! Polishing here & there and giving it an exciting look would improve it.
Y	The websites needs a search bar and some change in colour styling to make different sections stand out more.
Z	This is a decent site. Poor use of colour. It's boring.

The tables are the result of user testing of both the functional design as well as the prototype designs. This was created to evaluate the actual impact on users the application has. The testing yielded a positive result, with much feedback given. The key takeaways were the poor use of colour made the website bland and boring.

W3C Validator



The W3C Validator is an online tool that evaluates the accessibility and quality of HTML code. The HTML was extracted from the compiled main admin page and was tested using this software. The response above shows that the csrf_token from the security feature from wtforms library was not properly used. Further investigation will be partaken in the future to rectify this issue. This was a minor issue and shows the HTML is largely good.

Evaluation Against Criteria

Prescribed Criteria	Self-Determined Criteria
Developed technical proposal for a web application	Aesthetically pleasing design
Low-fidelity prototype solution	Efficient navigation of website
Relevant user interfaces and algorithm component of the prototype digital solution	Logical and intuitive buttons
Administrators able to easily create, upload and maintain the menu data	Intuitive and clear information and instructions
Customers able to create accounts and purchase products	Attractive and desirable User Interface
Upholds usability principles, including accessibility, effectiveness, safety, utility and learnability.	Appeals to user persona
Explores the interrelationships between user experiences and data	Usable for everybody, including those with sensory or cognitive impairments
Identification of errors to yield refinements	Works on a laptop
Ability to upload data via CSV file uploads	Minimalistic and sleek Design
Compliance with the Australian Privacy Act 1988 and Australian accessibility guidelines	Maintain a consistent and harmonious style guide throughout
	Show status of system wherever possible

The above tables show the criteria identified in the Explore phase. These relate to the quality of the solution. In this case, the criteria relating to UI and UX were directed towards the prototype designs and the functionality, security, and accessibility criteria were directed towards the coded application. The green fields represent that the criteria has been met, and blank fields represents items that were not completed at all. Overall, the vast majority of the criteria were successfully completed to a high standard, even with the difficult constraints. The two that were not completed were due to the developer running out of time. They are a small set of functionality that was not fully implemented and did not affect the main project. They had a minimal impact on the success of the project. However it is important that these criteria are addressed in the future.

Solution Requirements

Functional	Non-Functional
Data such that: Administrators can easily create, upload and maintain the concept shop data on the website. Administrators can upload product data to the website from the supplied CSV file. **Code for:** Web-based digital solution Database and tables Search and display a specific product Search and display by product category Read records from a CSV file and process them for storage in a database Create a secure password based on criteria Shopping cart transaction processing Pay for goods or services Calculate shipping Calculate total cost Product inventory management Product quantity is updated when a product is sold Shopping cart **Code such that:** An incorrect user registration will not be stored in the database Customers are able to register for an account, setting their own username and password Registered users are able to login to their account using a username and password Securely store the personal details and passwords that customers register with the site Save the types of alerts pre-selected by a customer Customers are sent alerts either by email or SMS Customers can add to cart, edit cart, save cart, continue shopping, select from a range of payment and delivery options	**User Interface/User Experience:** The application must have a responsive web design. **Visual Components for:** Customer account registration Customer account login Alert notifications – email alerts about sales, restock item alert, wish list offers Wish list Admin account login Data upload (admin) Data maintenance (admin) **Must comply with:** Government web design standards The Australian Privacy Act (1988) Web Content Accessibility Guidelines (WCAG 2.0) Copyright Law Useability Success Criteria Elements and Principles of Visual Communication

The solution requirements set by the client in the Explore phase were evaluated to identify how completed the project was. The green highlighting means the item is completed and the uncoloured means it is not. There were a considerable amount of items that did not get completed in the project. This was due to the restricted nature of the available resources, specifically time. This caused the scope to shrink and only the administrator pages and interactions and manipulations of data. While half of the criteria were not completed, the items that were completed were the core functionality that interacts with the data. The uncompleted items are surface-level things. However, the full interaction with the database and the manipulation of data were completed.

Impact from Constraints

The constraints identified in the Explore phase had a major impact on the completion of this project. The software was slow to get working and the developer had little knowledge of the frameworks of the project. This resulted in much time being lost due to building skills and running programs on the hardware successfully. This time was taken from an already limited pool and thus the scope of the project needed to be shrunk. This produced the effect seen in the Solution Requirements evaluation above, where almost half of the required items were not completed.

Recommendations and Suggestions

The project had many areas where improvement is possible. First and foremost, the constraints needs to be reworked. Allowing more time for the developer to implement the solution would result in a much more thorough and completed web application. Improvements to the solution that was created, however, include a much more colourful and exciting UI design to entice users. This problem was identified by user. Another suggestion is to incorporate loading and progress bars to show the system status and benefit the UX. The main recommendation is to complete the rest of the solution requirements and merge the prototype designs with the implemented solutions. This will produce a much more completed and refined solution that can be hosted and used.

Overall Summary and Conclusion

Overall, the solution was achieved to the best effort when the hefty constraints were taken into consideration. A solution was created and documented but it was not a fully thorough and completed web application. In the future, the project will come to that completed status after expanding the resources and limiting the scope.

Acknowledgements

Bibliography

Docherty K, Graham, J, & Russell, A. (2018). Nelson Digital Solutions for QCE Units 1-4. Cengage Learning Australia.

- Mind Map p. 12, 13, 15, 93, 224
- Elements of a web appl p. 99, 104
- IPO chart p.2, 39, 130
- Site map p. 13
- Wire-frame diagrams p.29-31
- Style guide Online @ Hillbrook
- Page mockups Adobe
- Desk checking p.43
- Elements and principles of visual communication p. 97
- Useability success criteria p.25, 279
- Accessibility guidelines p.279-280
- Collecting user feedback p.278
- Evaluation checklist p.75
- Evaluating with criteria p.190 - 191

All phone numbers used in this proposal use legally specified fake exemplar entries

Thompson, C. (2019). Algorithms And Pseudocode A compilation. [online] online@hillbrook. Available at: https://online.hillbrook.qld.edu.au/course/view.php?id=342 [Accessed 25 May 2020].

- Pseudocode conventions

Validator.w3.org. 2013. The W3C Markup Validation Service. [online] Available at: <https://validator.w3.org/> [Accessed 25 May 2021].
Groklearning.com. 2021. Grok Learning. [online] Available at: <https://groklearning.com/> [Accessed 25 May 2021].

Clark, L. and Batt, K., 2021. 12 Digital Solutions. [online] online.hillbrook.qld.edu.au. Available at: <https://online.hillbrook.qld.edu.au/course/view.php?id=343> [Accessed 25 May 2021].

HILLBROOK
IN BALANCE WE GROW

12 Digital Solutions

Statement of Authenticity and Academic Integrity

IA2

Project – Digital Innovation

Ky Eltis *certify that:*

- the planning, development, content and presentation of this assessment task is my own work in every respect
- this assessment task has not been copied from another person's work or from books or the Internet or any other source
- I have used appropriate research methods and have not used the words, ideas, designs, music, images, skills or workmanship of others without appropriate acknowledgement in the assessment task or its development

Student Signature: _____ **Student Number: 125881**

Date: 25/5/21

Parent Signature: _____

Date: 25/5/21

ALGORITHM

DOCUMENT

Connection with Database

```python
#IMPORTS
import os
from flask import Flask, session
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager
from flask_bootstrap import Bootstrap

#INIT FLASK
app = Flask(__name__)
Bootstrap(app)

# CONFIGS FROM PRESET ENVIRONMENT VARIABLES
mariadb_username = os.environ["MY_DATABASE_USERNAME"]
mariadb_password = os.environ["MY_DATABASE_PASSWORD"]
mariadb_host = os.environ["MY_DATABASE_HOST"]
mariadb_database_name = os.environ["MY_DATABASE_NAME"]
basedir = os.path.abspath(os.path.dirname(__file__))
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://{}{}@{}/{}?local_infile=1'.format(mariadb_username,mariadb_password,   \
                                                                                           mariadb_host,mariadb_database_name)
# SET CONFIGS
app.config['SECRET_KEY'] = 'mysecret' # USE ENV VARIABLE FOR PRODUCTION
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['UPLOAD_FOLDER'] = os.path.join(basedir, 'tmpfiles')

#INIT DATABASE
db = SQLAlchemy(app)
```

This is an excerpt from the primary __init__.py file for the application. It can be seen that the “db” variable that represents the database connection is initialised through SQLAlchemy and Flask by using the MySQL engine with the PyMySQL DBAPI.

```python
from amazongeek import db,login_manager
from werkzeug.security import generate_password_hash,check_password_hash
from flask_login import UserMixin

@login_manager.user_loader              # Decorator from flask-login
def load_user(admin_id):                # flask-login function for grabbing admin user from ID
    return Admin.query.get(admin_id)    # Return admin user

class Admin(db.Model, UserMixin):       # Admin class
    __tablename__ = 'admin'             # Name table
    admin_id = db.Column(db.Integer, primary_key=True)  # Column
    username = db.Column(db.String(64), unique=True)    # Column
    password = db.Column(db.String(128))                # Column
    def __init__(self, username, password):             # When an instance is made, have these arguments
        self.username = username                        # Set username to inputted username
        self.password = generate_password_hash(password)# Set password to the hash of the inputted password
    def check_password(self,password):                  # Function to check passwords match
        return check_password_hash(self.password,password)  # Return result
    def get_id(self):                                   # Get ID function
        return self.admin_id                            # Return ID
```

Above is a snippet of models.py file which creates the tables and field names and types as well as any function that might be used.

```python
############## ALIAS FOR EASY USE ###############
def sql(sqltext):                           # Make a function with shortens the language to do sql
    return db.session.execute(text(sqltext))    # Return code to execute sql
```

This function is used heavily in the application and it is an alias of writing the whole command. It uses the db object to directly execute SQL queries to the database. If the queries change the database, then the function is often followed by ”db.session.commit()” to save the changes.

Libraries and Imports

```python
################ IMPORT STATEMENTS ######################
from flask import render_template,request,Blueprint, flash, send_file, send_from_directory, safe_join, abort, session, redirect, url_for
from amazongeek import db, app
from amazongeek.models import Admin, Customer, Order, Product, ProductOrder
from amazongeek.admin.forms import LoginForm, SelectTable, UploadCSV, DownloadCSV, ResetTable, DynamicDeleteRow, DynamicUpdateField, \
                                   DynamicSearchTable, CauseError, AddRowProduct, AddRowOrder, AddRowAdmin, AddRowProductOrder, AddRowCustomer
from flask_login import login_user, current_user, logout_user, login_required
from sqlalchemy import text
from werkzeug.security import generate_password_hash
import csv
import tempfile
import os
import io

################ INITIALISING ######################
database = os.environ["MY_DATABASE_NAME"]       # Pull variable from system environment
admin = Blueprint('admin',__name__)             # Make blueprint for structure of files and urls
tempfile.tempdir = app.config['UPLOAD_FOLDER']  # Get upload folder for temporary files

########## CREATE NITIAL ADMIN USER FORM ###########
def add_row():                                  # This function creates the initial admin you login as
    try:
        a = Admin(username="admin123",password="password123")   # Admin data
        db.session.add(a)                       # Add to database
        db.session.commit()                     # Save database
    except:
        pass                                    # Ignore errors (i.e. repeat addition to db)
add_row()                                       # Call function

############## ALIAS FOR EASY USE ###############
def sql(sqltext):                               # Make a function with shortens the language to do sql
    return db.session.execute(text(sqltext))    # Return code to execute sql
```

This is a snippet of the beginning of the main views.py file in the solution. It imports a whole lot of things relating to flask and sqlalchemy. It also imports many classes and functions from local files. One of the main ones is the db object imported from the base directory ‘amazongeek’.

How forms work

Forms in this application work by the use of multiple libraries and software. Namely Wtforms and FlaskForm were the main contributors. They have very simple and easy templates and shortcuts to simplify the creation of the HTML forms. These modules combined with the Jinja templating offers a very streamlined and simple way of creating and managing forms.

Delete Row

Input	Process	Output
Database Table Id	Check if the selected row includes the user data for the current logged in user. Execute SQL to remove row from database.	Display success or error message.

```
BEGIN DeleteRow
INIT db = "My_Database_Name"
INIT currentId = 1
INIT adminTable = 'admin'
INIT table = CALL formTable
INIT idColumn = CALL sqlIdColumn(db, table)
INIT ids = CALL sqlIds(idColumn, table)
INIT responseId = CALL formDeleteRow(ids)
IF table = adminTable AND responseId = currentId THEN
        DISPLAY "Cannot remove logged in user!"
ELSE
        CALL sqlRemoveRow(responseId, idColumn, table)
END DeleteRow
```

```
sqlIdColumn(db, table)
SELECT COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = table
AND TABLE_SCHEMA = db
AND COLUMN_KEY = "PRI"

sqlIds(idColumn, table)
SELECT idColumn
FROM table
ORDER BY idColumn

sqlRemoveRow(id, idColumn, table)
DELETE FROM table
WHERE idColumn = id
```

```python
########### DELETE ROW FORM ####################
sqlIdColumn = sql("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = '{}' AND            \
                  TABLE_SCHEMA = '{}' AND COLUMN_KEY = 'PRI'".format(table,database))   # Search for primary key column in table in database
idColumn = [i for i in sqlIdColumn][0][0]                                               # Iterate the sql to make it workable
sqlIds = sql("SELECT {} FROM {} ORDER BY {} ASC".format(idColumn,table,idColumn))       # Find all IDs in table and order them
ids = [(str(i[0]),i[0]) for i in sqlIds]                                                # Iterate the sql to make it workable
formDeleteRow = DynamicDeleteRow(ids)                                                   # Create form to select row ID. The IDs are passed in
if "submitDeleteRow" in request.form and formDeleteRow.validate_on_submit():            # If the form is submitted:
    id = formDeleteRow.id.data                                                          # Take the form data
    if table == 'admin' and int(id) == int(current_user.admin_id):                      # Check if admin is about to delete their own account
        flash("You are trying to delete the current logged in user!")                   # Display Warning
    else:
        sql("DELETE FROM {} WHERE {} = {}".format(table,idColumn,id))                   # Deletes row from database
        db.session.commit()                                                             # Save changes to database
```

admin/views.py

```python
# Function used to pass the ids argument into the form class
def DynamicDeleteRow(*args, **kwargs):                  # Define function with any amount of arguments and keyword arguments
    ids = args[0]                                       # Set IDs to the first argument passed in (i.e. the IDs)
    class DeleteRow(FlaskForm):                         # Inherit Flaskform class
        id = SelectField("Primary ID:", choices=ids, validators=[DataRequired()])# Create dropdown select field
        submitDeleteRow = SubmitField('Delete Row') # Create submit button
    return DeleteRow()                                  # Return the class instance
```

admin/forms.py

Update Field

Input	Process	Output
Database Table Id Column Data	Check that the selected row does not include user data for the current logged in user. Check that the column is not a primary field. If the column is a password, hash the input for the new data. Execute SQL to update field in database.	Display success or error message.

```
BEGIN UpdateField
INIT db = "My_Database_Name"
INIT currentId = 1
INIT adminTable = 'admin'
INIT table = CALL formTable
INIT idColumn = CALL sqlIdColumn(db, table)
INIT ids = CALL sqlIds(idColumn, table)
INIT columns = CALL sqlColumns(db, table)
INIT responseId, responseColumn, responseReplacement = CALL formUpdateField(ids, columns)
IF column = idColumn THEN
        DISPLAY "Cannot change primary key column!"
ELSE IF table = adminTable AND responseId = currentId THEN
        DISPLAY "Cannot remove logged in user!"
ELSE
        IF column = 'password' THEN
                responseReplacement = CALL hash(responseReplacement)
        CALL sqlUpdateField(table, idColumn, responseColumn, responseReplacement, responseId)
END UpdateField
```

```
sqlIdColumn(db, table)
SELECT COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = table
AND TABLE_SCHEMA = db
AND COLUMN_KEY = "PRI"

sqlUpdateField(table, idCol, col, text, id)
UPDATE table
SET col = text
WHERE idCol = id

sqlIds(idColumn, table)
SELECT idColumn
FROM table
ORDER BY idColumn
```

```python
########### UPDATE FIELD FORM ####################
sqlIdColumn = sql("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = '{}' AND            \
                  TABLE_SCHEMA = '{}' AND COLUMN_KEY = 'PRI'".format(table,database))   # Search for primary key column in table in database
idColumn = [i for i in sqlIdColumn][0][0]                                               # Iterate the sql to make it workable
sqlIds = sql("SELECT {} FROM {} ORDER BY {} ASC".format(idColumn,table,idColumn))       # Find all IDs in table and order them
ids = [(str(i[0]),i[0]) for i in sqlIds]                                                # Iterate the sql to make it workable
sqlColumns = sql("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = '{}' AND            \
                  TABLE_SCHEMA = '{}'".format(table,database))                          # Find all column names in table
columns = [(str(i[0]), i[0]) for i in sqlColumns]                                       # Iterate the sql to make it workable
formUpdateField = DynamicUpdateField(ids, columns)                                      # Create form to select row ID. The IDs are passed in
if "submitUpdateField" in request.form and formUpdateField.validate_on_submit():        # Check if the form is complete and submitted
    id = formUpdateField.id.data                                                        # Take the id form data
    column = formUpdateField.column.data                                                # Take the column form data
    replacement = formUpdateField.replacement.data                                      # Take the replacement form data
    if column == idColumn:                                                              # Check if column is a primary key
        flash("You cannot change a primary key column!!")                               # Display Warning
    elif table == 'admin' and int(id) == int(current_user.admin_id):                    # Check if admin is about to delete their own account
        flash("You cannot change the logged in user!!")                                 # Display Warning
    else:
        if column == "password":                                                        # If column is a password column
            replacement = generate_password_hash(replacement)                           # Hash the password before storing
        try:
            sql("UPDATE {} SET {} = '{}' WHERE {} = '{}'".format(table,column,replacement,idColumn,id)) # Update field in database
            db.session.commit()                                                         # Save changes to database
        except:
            flash("Field Type does not match")                                          # If error raised, display warning
```

admin/views.py

```python
# Function used to pass the ids and columns arguments into the form class
def DynamicUpdateField(*args, **kwargs):                # Define function with any amount of arguments and keyword arguments
    ids = args[0]                                       # Set IDs to the first argument passed in (i.e. the IDs)
    columns = args[1]                                   # Set columns to the second argument
    class UpdateField(FlaskForm):                       # Inherit Flaskform class
        id = SelectField("Primary ID:", validators=[DataRequired()], choices=ids)       # Create dropdown select field
        column = SelectField("Column:", validators=[DataRequired()], choices=columns)   # Create dropdown select field
        replacement = StringField("Replace data with: ", validators=[DataRequired()])   # Create string field
        submitUpdateField = SubmitField('Update Field') # Create submit button
    return UpdateField()                                # Return the class instance
```

admin/forms.py

Search Table

Input	Process	Output
Database Table Column Filter FilterColumn FilterValue	Check if the filter was used. Execute SQL accordingly	Display success or error message

sqlSearchTableFilter(table, getColumn, filterColumn, filterValue)

```
SELECT getColumn
FROM table
WHERE filterColumn = filterValue
```

sqlSearchTable(table, getColumn)

```
SELECT getColumn
FROM table
```

sqlColumns(db, table)

```
SELECT COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = table
AND TABLE_SCHEMA = db
```

```
BEGIN SearchTable
INIT db = "My_Database_Name"
INIT table = CALL formTable
INIT columns = CALL sqlColumns(db, table)
INIT getColumn, filterColumn, filterValue, useFilter = CALL formSearchTable(columns)
IF useFilter = True THEN
        CALL sqlSearchTableFilter(table, getColumn, filterColumn, filterValue)
ELSE
        CALL sqlSearchTable(table, getColumn)
END SearchTable
```

```python
############### SEARCH TABLE FORM ######################
    sqlColumns = sql("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = '{}' AND             \
                      TABLE_SCHEMA = '{}'".format(table,database))                    # Find all column names in table
    columns = [(str(i[0]), i[0]) for i in sqlColumns]                                 # Iterate the sql to make it workable
    formSearchTable = DynamicSearchTable(columns)                                     # Create form
    if "submitSearchTable" in request.form and formSearchTable.validate_on_submit():    # Check if the form is submitted correctly
        value = formSearchTable.value.data                                            # Take the value form data
        getColumn = formSearchTable.getColumn.data                                    # Take the getColumn form data
        searchColumn = formSearchTable.searchColumn.data                              # Take the searchColumn form data
        useFilter = formSearchTable.useFilter.data                                    # Take the useFilter form data
        if useFilter:                                                                 # If user has selected filter option
            sqlSearchTableData = sql("SELECT {} FROM {} WHERE {} = '{}'".format(getColumn,table,searchColumn,value)) # Query table using SQL
        else:
            sqlSearchTableData = sql("SELECT {} FROM {}".format(getColumn,table))     # Query table using SQL
        searchTableData = [i[0] for i in sqlSearchTableData] # Iterate sql to make it workable (then passed into jinja for displaying)
    else:
        searchTableData = ''                                                          # Make empty to avoid unset variable
```

admin/views.py

```python
# Function used to pass the columns argument into the form class
def DynamicSearchTable(*args, **kwargs):                # Define function with any amount of arguments and keyword arguments
    columns = args[0]                                   # Set columns to the first argument passed in
    class SearchTable(FlaskForm):                       # Inherit Flaskform class
        getColumn = SelectField("Show values of: ", validators=[DataRequired()], choices=(columns)) # Create dropdown select field
        useFilter = BooleanField("Use Filter?")         # Create boolean (tickbox) field
        searchColumn = SelectField("Filter where ", validators=[DataRequired()], choices=columns)   # Create dropdown select field
        value = StringField(" = ")                      # Create string field
        submitSearchTable = SubmitField('Search')       # Create submit button
    return SearchTable()                                # Return the class instance
```

admin/forms.py

Add Row

Input	Process	Output
Table Data	Execute SQL to add row into the table.	Display success or error message

```
BEGIN AddRow
INIT table = CALL formTable
INIT formData = CALL formAddRow(table)
CALL sqlAddRow(table, formData)
END AddRow
```

sqlAddRow(table, formData)

```
INSERT INTO table
VALUES (value FOR value IN formData)
```

```python
############ ADD ROW FORM ######################
formAddRow = DynamicAddRow(table)
formAddRowFields = [i for i in formAddRow][:-2] # Get all fields from form (except submit and CSRF security tag)
if "submitAddRow" in request.form and formAddRow.validate_on_submit(): # If form is submitted correctly
    formData = [i.data for i in formAddRowFields] # Get all data from those fields
    if table == 'admin':
        row = Admin(*formData)                  # Set row to instance of Admin with the form data (asterisk expansion)
    elif table == 'customer':
        row = Customer(*formData)               # Set row to instance of Customer with the form data (asterisk expansion)
    elif table == 'orders':
        row = Order(*formData)                  # Set row to instance of Order with the form data (asterisk expansion)
    elif table == 'products':
        row = Product(*formData)                # Set row to instance of Product with the form data (asterisk expansion)
    elif table == 'product_orders':
        row = ProductOrder(*formData)           # Set row to instance of ProductOrder with the form data (asterisk expansion)
    try:
        db.session.add(row)                     # Add row to the database
        db.session.commit()                     # Save changes
    except:
        flash("Duplicate Row!")                 # If error, flash error message
```

admin/views.py

```python
def DynamicAddRow(*args, **kwargs):             # Define function with any amount of arguments and keyword arguments
    table = args[0]
    class AddRow(FlaskForm):                                                # Form class to add rows into Admin table
        if table == 'admin':
            username = StringField('Username', validators=[DataRequired()])         # Username field
            password = PasswordField('Password', validators=[DataRequired()])       # Password field
            submitAddRow = SubmitField('Add Row')                                   # Submit field
        elif table == 'customer':
            given_name = StringField('Given Name', validators=[DataRequired()])     # Given name field
            family_name = StringField('Family Name', validators=[DataRequired()])   # Last name field
            address_line1 = StringField('Address 1', validators=[DataRequired()])   # Address line 1 field
            address_line2 = StringField('Address 2')                                # Address line 2 field (optional)
            phone_number = StringField('Phone Number', validators=[DataRequired()]) # Phone number field
            email = StringField('Email', validators=[DataRequired(),Email()])       # Email field (special validator)
            username = StringField('Username', validators=[DataRequired()])         # Username field
            password = PasswordField('Password', validators=[DataRequired()])       # Password field
            payment = StringField('Payment Method', validators=[DataRequired()])    # Payment field
            submitAddRow = SubmitField('Add Row')
        elif table == 'orders':
            customer_id = IntegerField('Customer ID', validators=[DataRequired()])  # Customer id field
            price = FloatField('Price', validators=[DataRequired()])                # Price field
            status = StringField('Status', validators=[DataRequired()])             # Status field
            paid = BooleanField('Paid', validators=[DataRequired()])                # Paid field
            submitAddRow = SubmitField('Add Row')                                   # Submit field
        elif table == 'products':
            category = StringField('Category', validators=[DataRequired()])         # Category field
            brand = StringField('Brand', validators=[DataRequired()])               # Brand field
            title = StringField('Title', validators=[DataRequired()])               # Title field
            size = StringField('Size (Optional)')                                   # Size field
            description = StringField('Description', validators=[DataRequired()])   # Description field
            country_of_origin = StringField('Country Of Origin', validators=[DataRequired()]) # Country of origin field
            price = FloatField('Price', validators=[DataRequired()])                # Price field
            quantity = IntegerField('Quantity', validators=[DataRequired()])        # Quantity field
            image = StringField('Image Filename (Optional)')                        # Image field
            submitAddRow = SubmitField('Add Row')                                   # Submit field
        elif table == 'product_orders':
            order_id = StringField('Given Name', validators=[DataRequired()])       # Order id field
            product_id = StringField('Given Name', validators=[DataRequired()])     # Product id field
    return AddRow()
```

admin/forms.py