

COS 221 Assignment 5 Submission Details

Table of Contents

Task 1: Research	4
General Overview and Explanation	4
Product Categories and Their Retailers	4
Information On How Content Is Rated or Categorised	4
Additional features/ information provided:	4
References	5
Task 2: (E)ER-Diagrams	6
Iteration 1:	6
Diagram:	6
Assumptions:	6
Iteration 2	7
Diagram:	7
Changes:	7
Assumptions:	7
Iteration 3	8
Diagram:	8
Changes:	8
Assumptions:	8
Task 3: Relational Mapping	9
Step 1: Strong components	9
Step 2: Weak Components	10
Step 3: 1:1 Relations	10
Step 4: 1:N Relations	11
Step 5: M:N Relations	12
Step 6: Multivariable Attributes	12
Step 7: N-ary relations	12
Step 8: Mapping of specialization and generalization	13
Step 9: Mapping Unions	13
Task 4: Relational Schema	14
Visual Representation	14
Indexes/Secondary Keys	14
Review	14
Product_Retailer	14
Product	15
Watchlist_Item	15
Lengths and checks	15
Task 5: Web Application	15
Task 6: Data	15
Database Population Method	15
Overview of the Data Population Approach	15
Data Sources	15
Data Processing Pipeline	16
Final Database Creation	16
Data Generation Techniques	17
Database Schema	17
Task 7: Analyze and Optimise	18
View All Products	18
Query	18

Analysis.....	19
Interpretation:.....	19
Task 8: Git usage.....	20
Division of Project.....	20
Feature Development.....	20
Contributions.....	20
API.....	20
Frontend.....	20
JS Backend.....	20
Schema:.....	20
Database Population.....	21
(E)ER-D.....	21
Relational Mapping.....	21
PDF construction.....	21
Github Management:.....	21
Research.....	21
Final Structure.....	21
Task 9: Demo.....	21
Task 10: Bonus Features.....	22
Feature 1: Enhanced UI.....	22
Feature 1.1: Mobile UI.....	22
Feature 1.2: Contrast.....	24
Feature 1.3: Dark Mode.....	26
Feature 2: Security.....	26
Feature 2.1: Hashed Passwords.....	26
Feature 2.2: Error Log.....	27

Task 1: Research

General Overview and Explanation

The retail and e-commerce industry is one of the main methods people use to purchase different products in today's world. Many users prefer online shopping in comparison to in person. This places importance on the trust of the company, and by extension the method of interacting with its customers. The navigability of their website, the ratings of the products and the overall user experience is vital to the success of the industry.

Product Categories and Their Retailers

Retailers categorise products to improve customer experience by allowing users to efficiently locate and browse products of a specific purpose. Retailers stock a variety of categories and some of these categories are more susceptible to market volatility and supply than others, leading to products categories with unstandardised prices across various retailers. Customers tend to manually verify the best retailer to purchase a product from after considering an array of retailers. The goal of a price checker is to streamline this process by providing all the necessary information for a customer to pick a retailer in one website.

Information On How Content Is Rated or Categorised

Rating: The rating of products is typically calculated based on user reviews. Users who have purchased the product may leave a review ranging from 1-5 and the overall rating of the product is its average rating. Comments about the product can be added for an explanation of their score.

Categories: The category of a product assists the user in determining the use of a product and grouping products with similar uses together for evaluation. Categories typically used to describe the purpose of a product include: Brand (Brands tends to specialise in one category), Availability (users may only want to view products readily available), Title and Description (The purpose of a product is often mentioned in its title and/or description).

Additional features/ information provided:

Many of the leaders in the e-commerce industry have features such as recommendations, allow users to leave product reviews, have daily deals and so on. Some gained a competitive advantage by allowing users to price check products against other websites, showing where to get the best deal. Furthermore, frequently mentioned keywords across user reviews can be extracted as buzzwords and used as tags for better content categorization.

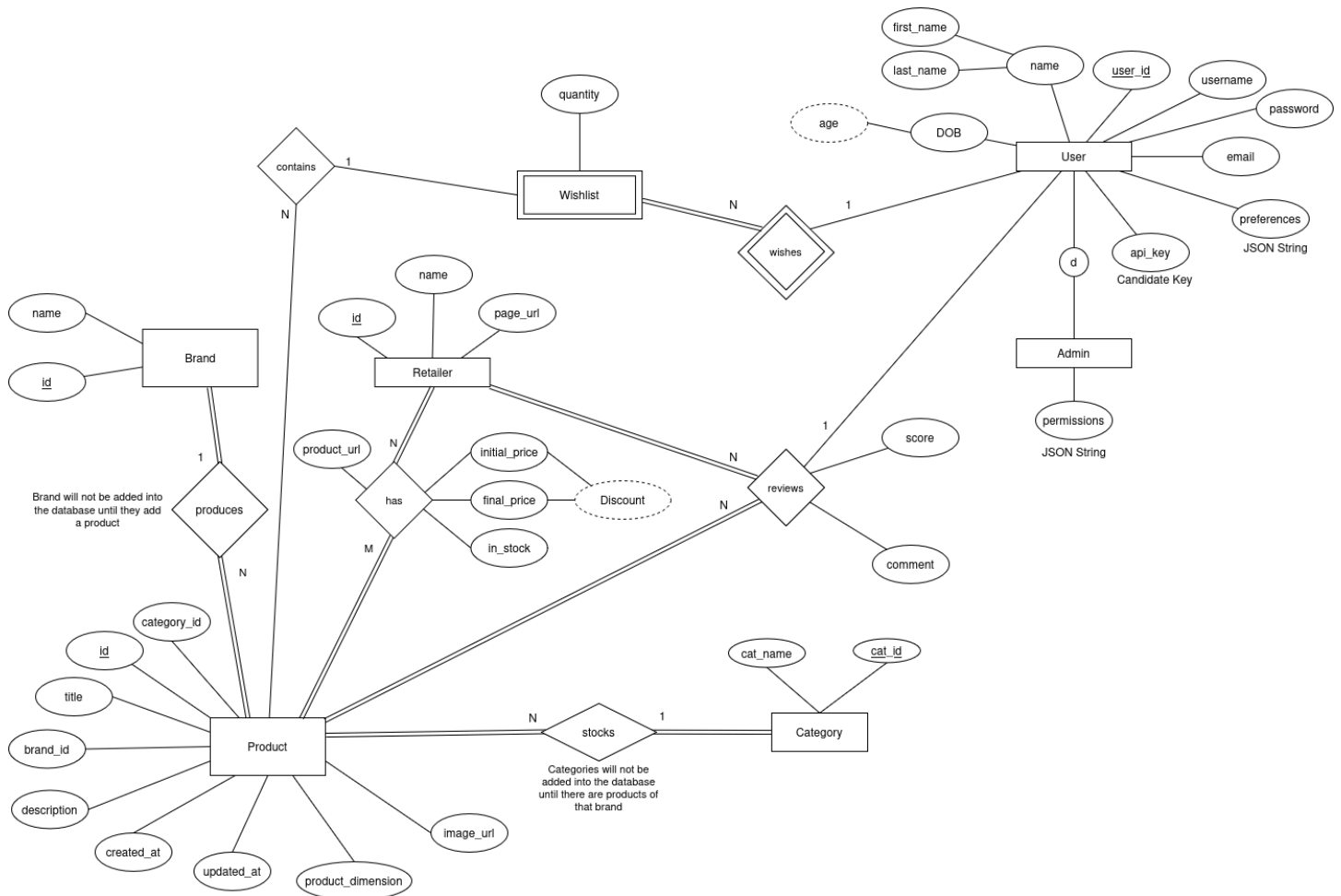
References

1. "Best Price Comparison Site: PriceCheck." PriceCheck. <https://www.pricecheck.co.za/> (Accessed 01-05-2025).
2. "idealo – Your Price Comparison." idealo. <https://www.idealco.co.uk/> (Accessed 01-05-2025).
3. BC. Lim, CMY. Chung, "Word-of-mouth: The use of source expertise in the evaluation of familiar and unfamiliar brands," Asia Pacific Journal of Marketing and Logistics, vol. 26, no. 1, pp. 39-51, Sep 2013, doi: <https://doi.org/10.1108/APJML-02-2013-0027>.
4. AA. Alhaddad, "Perceived Quality, Brand Image and Brand Trust as Determinants of Brand Loyalty," Journal of Research in Business and Management, vol. 3, no. 4, pp. 01-08, May 2015. [Online]. Available: <https://www.questjournals.org/jrbm/papers/vol3-issue4/A340108.pdf>.
5. A. koçer. "Amazon Pricing Study: The Most Expensive Products, Category Volatility, and Seasonal Price Shifts." Alienroad.
<https://alienroad.com/amazon-pricing-study-the-most-expensive-products-category-volatility-and-seasonal-price-shifts/> (Accessed 01-05-2025).
6. "7 Popular Purchasing Methods Consumers should Use in 2025." Holistique Training.
<https://holistiquetraining.com/en/news/7-popular-purchasing-methods-consumers-should-use-in-2023> (Accessed 01-05-2025).

Task 2: (E)ER-Diagrams

Iteration 1:

Diagram:

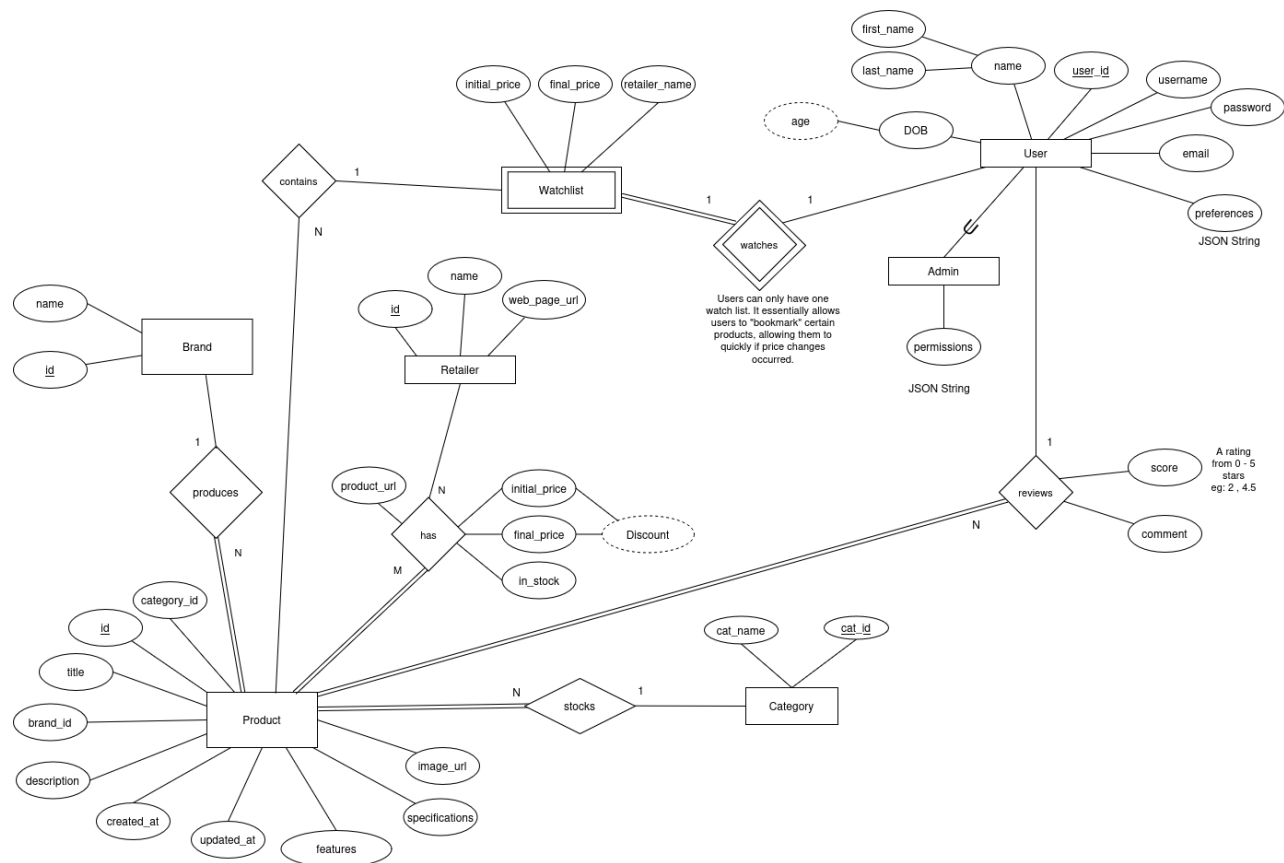


Assumptions:

- Brand and category cannot be added without having an existing product,
- The user could use an api key to login
- Wishlist contains wished products and how many are needed
- Products and retailers form the backbone of the entire system.
- You can leave reviews on both retailers and products individually.

Iteration 2

Diagram:



Assumptions

dev notes

decide how we will approach the links

Changes:

- Apikey is now removed, due to being redundant and leading to security issues.
- Wishlist is named to watchlist to be more appropriate.
- Watchlist no longer stores quantity, but now stores the price the product was at that very instant so that it can be compared to future prices.
- Product got 2 fields in features and specifications, instead of everything being in description, these are JSON objects that will be used to format the product page of that specific product.
- Removed total participation on 1 side of category, brand, and retailer, so that it is easier to insert these items with queries in the future.
- Changed admin entity relationship to be clearer as to what purpose it serves.
- Removed relationship between retailer and products, the implementation of it was not feasible in retrospect, and would lead to complications later in the line. This is a feature being removed.

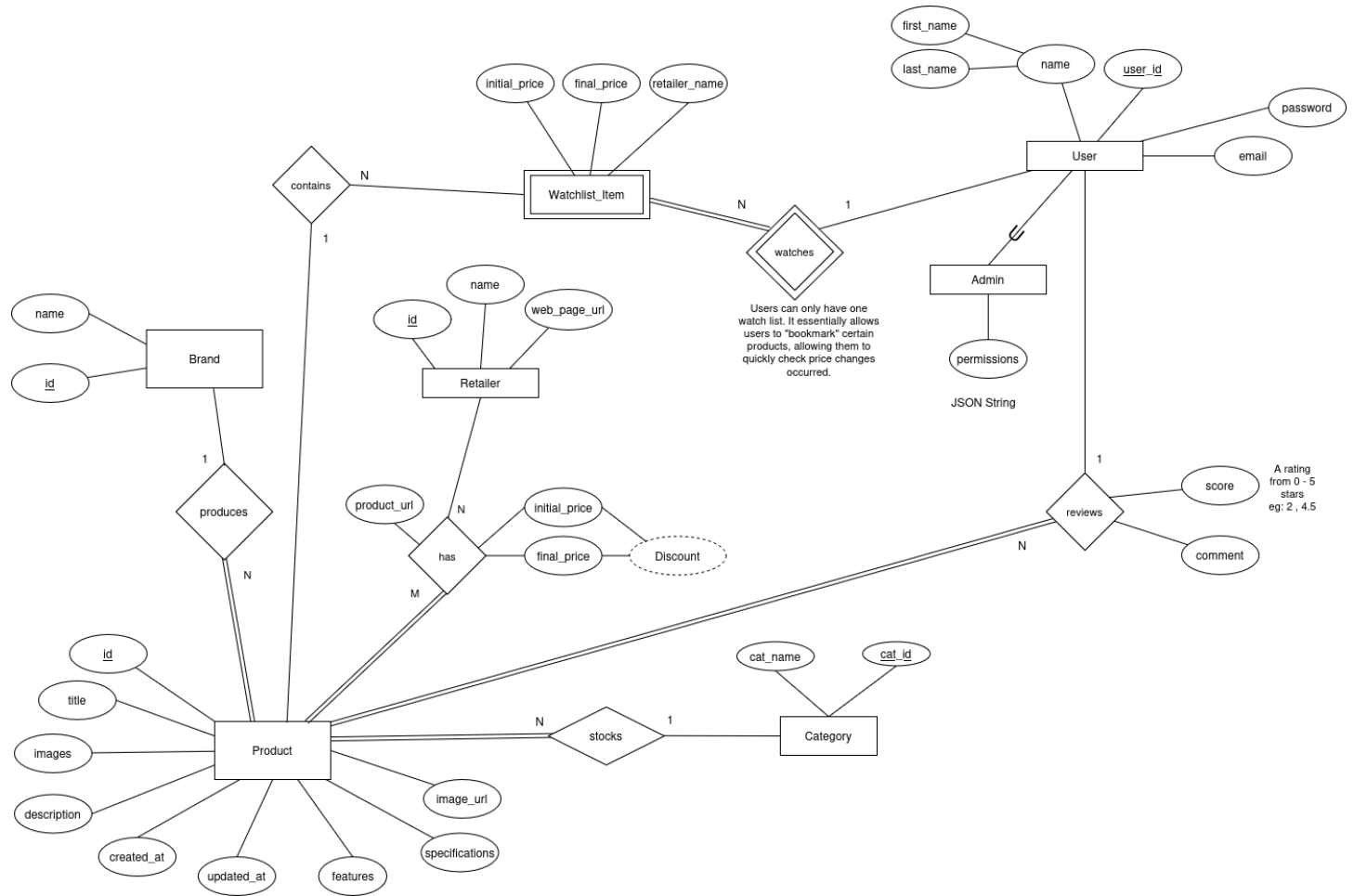
Assumptions:

- Watchlist is identified by a user id and a specific product being watched, therefore not requiring its own primary key, and classifying it as a weak entity, as it cannot exist without a user. A redundant primary key was recommended, but it would make getting all items on the watchlist clunky as it would need to be a composite between user id, product id, and that users watchlist id, which means watchlist is redundant. The other scenario is that each entry has a unique key, but that means it relies on the user

id to see which keys belong to which users, and a list of all keys would need to be given to the website, which was determined by the user id, so therefore we believe it is better to not have the key.

Iteration 3

Diagram:



Changes:

- Removed redundant attributes, such as preferences, since only dark mode was decided to be handled by preferences, which is automatically determined by the users browser settings, so no need to store it in the database.
- DOB was removed as it was used.
- In_stock was removed as it was not used.
- Removed foreign key references in Product, their existence is implicit with the relationship.
- Added images array to allow for multiple images on a product for image carousels.
- Renamed Watchlist to Watchlist_Item to better state its purpose.
- Removed username as it was redundant with first name and last name.
- Switched some cardinalities to be in the correct position

Assumptions:

- A watchlist cannot exist without a user, and will be identified by the users id and the product id it references. There is no retailer id as the data is static and does not need to change, it will be gotten from the request.

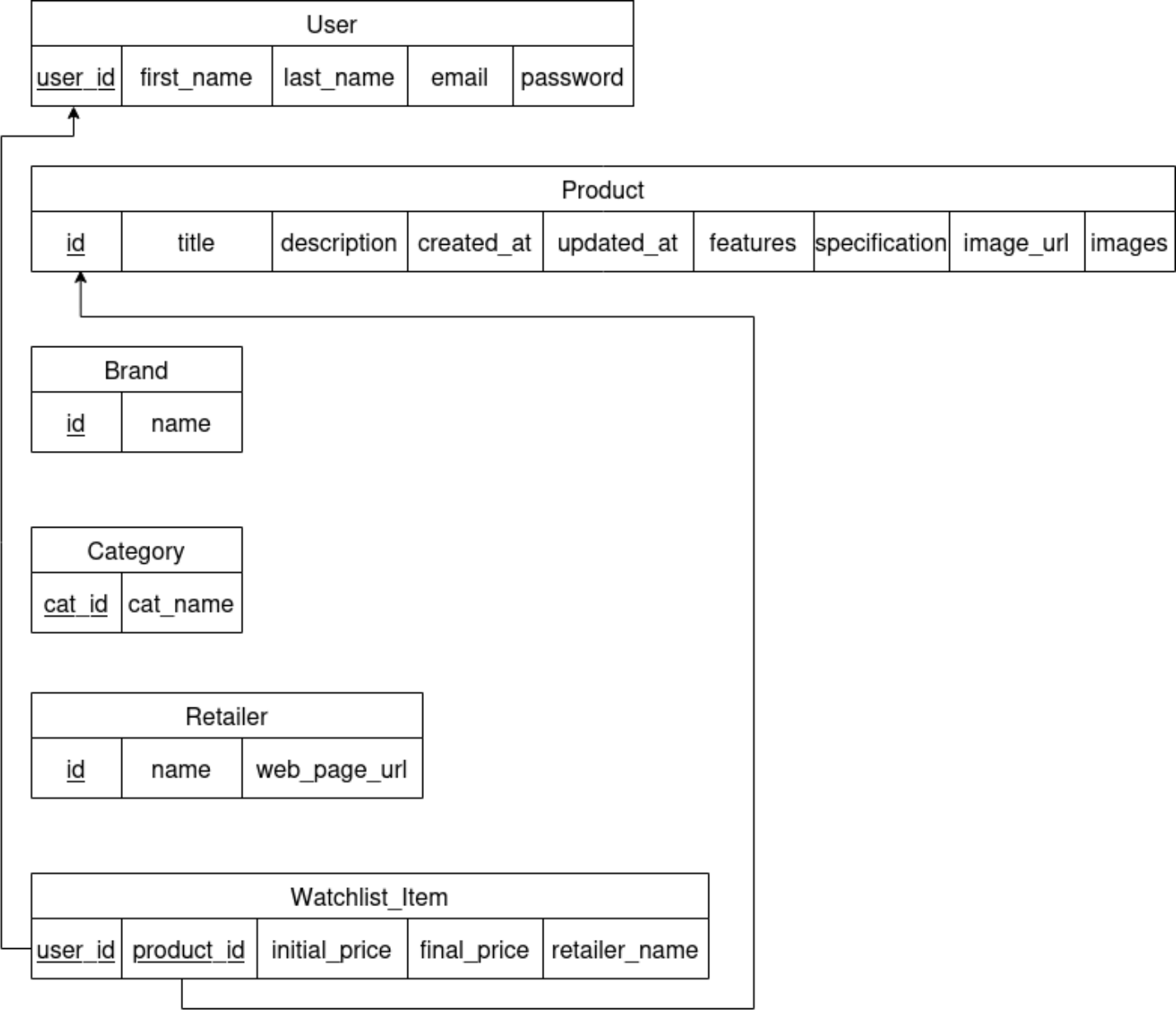
- A product must belong to a brand and category at the minimum, and nothing will be displayed if there is no retailer.
- There will be some dead urls in the population as this is only a demonstration, it would be too much work to get a unique url for everything, but it should in theory be possible. There will likely only be 3 urls that are redirected to in the demo.
- Email is a candidate key.
- It is assumed that a product is always in stock.

Task 3: Relational Mapping

Step 1: Strong components

User									
<u>user_id</u>	first_name	last_name	email	password					
Product									
<u>id</u>	title	description	created_at	updated_at	features	specification	image_url	images	
Brand									
<u>id</u>	name								
Category									
<u>cat_id</u>	cat_name								
Retailer									
<u>id</u>	name	web_page_url							

Step 2: Weak Components

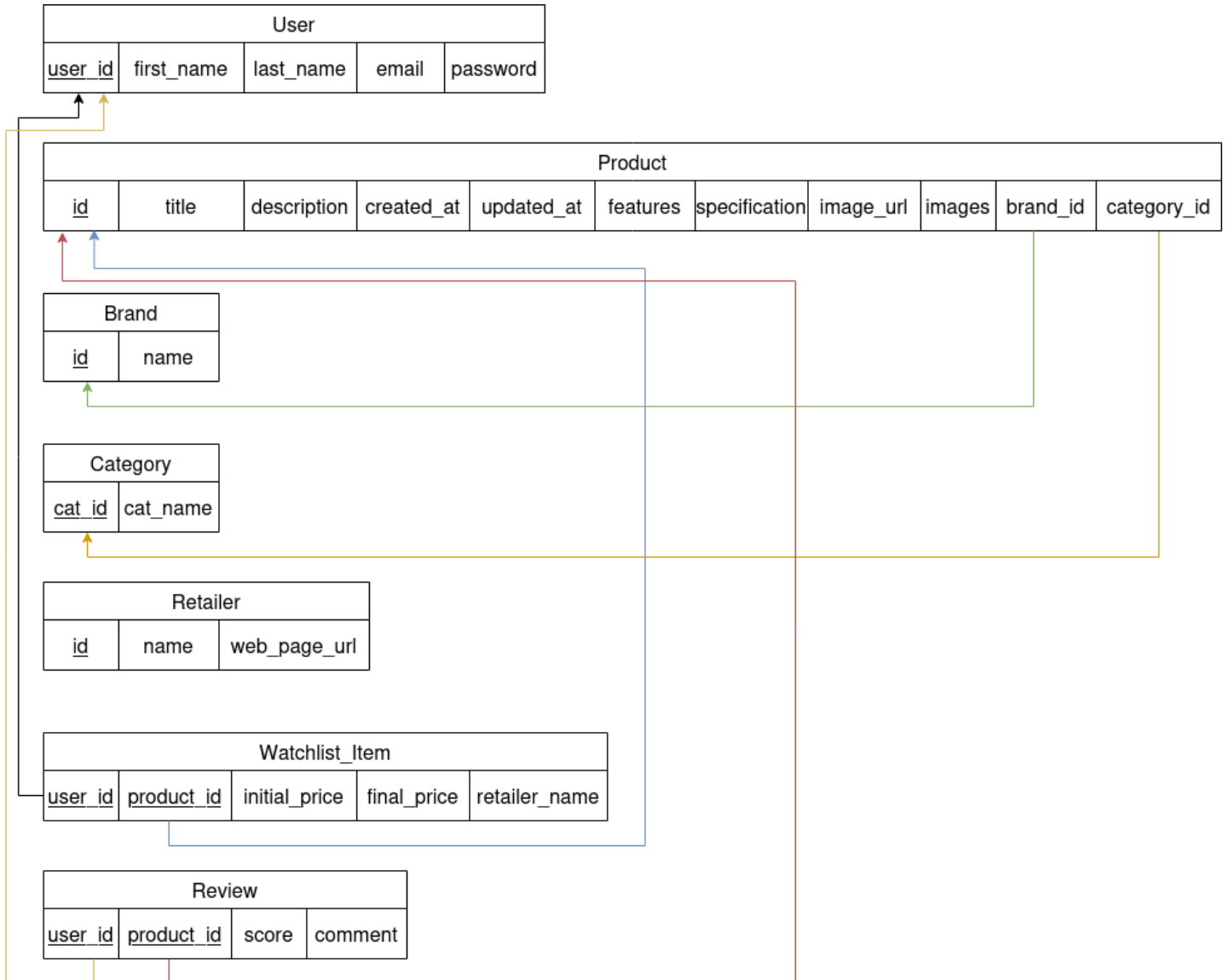


Step 3: 1:1 Relations

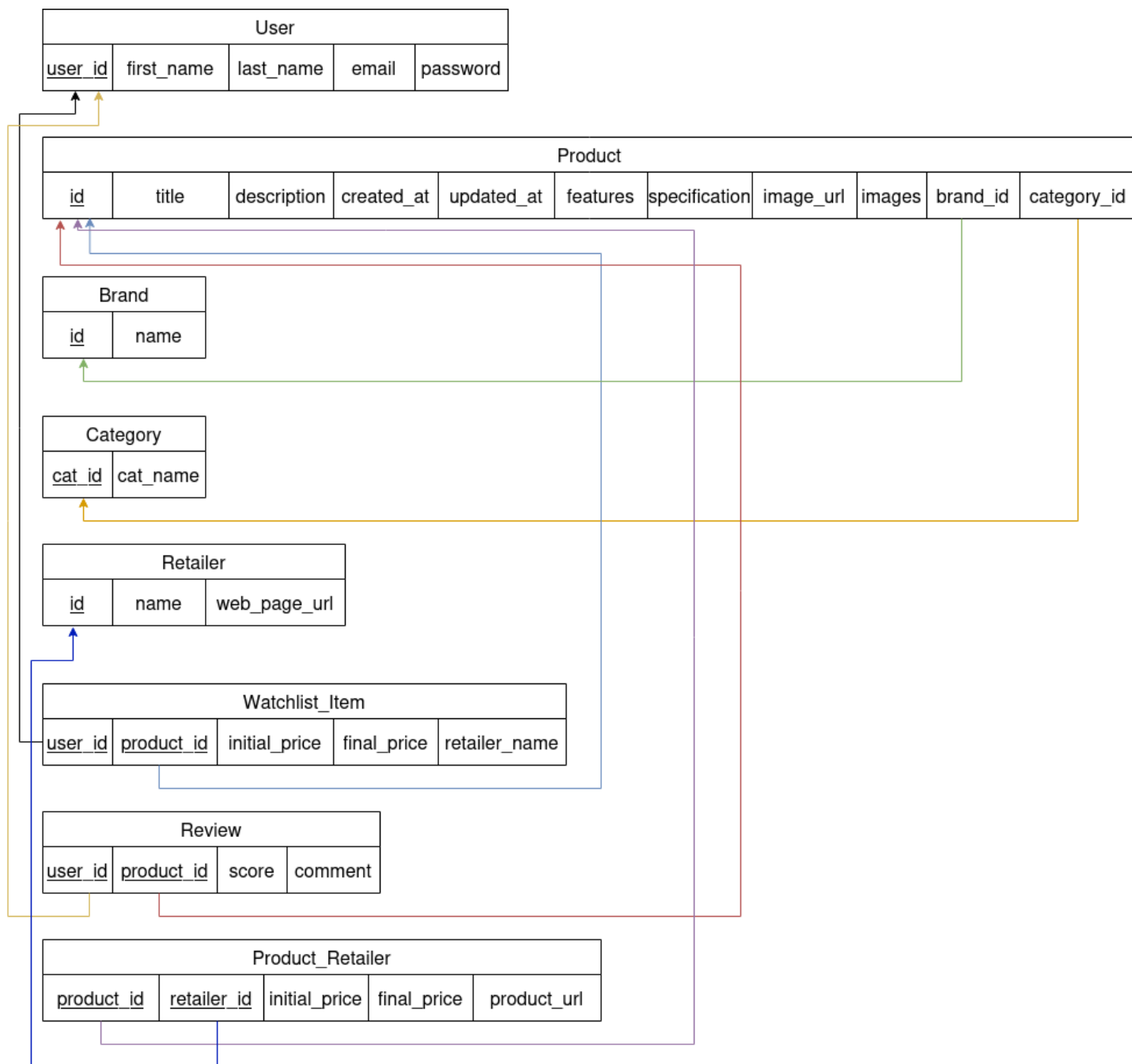
The Diagram has no 1:1 relationships included.

Step 4: 1:N Relations

Used both approaches for product and review, review uses both foreign keys as it's primary key, as to enforce referential integrity.



Step 5: M:N Relations



Step 6: Multivariable Attributes

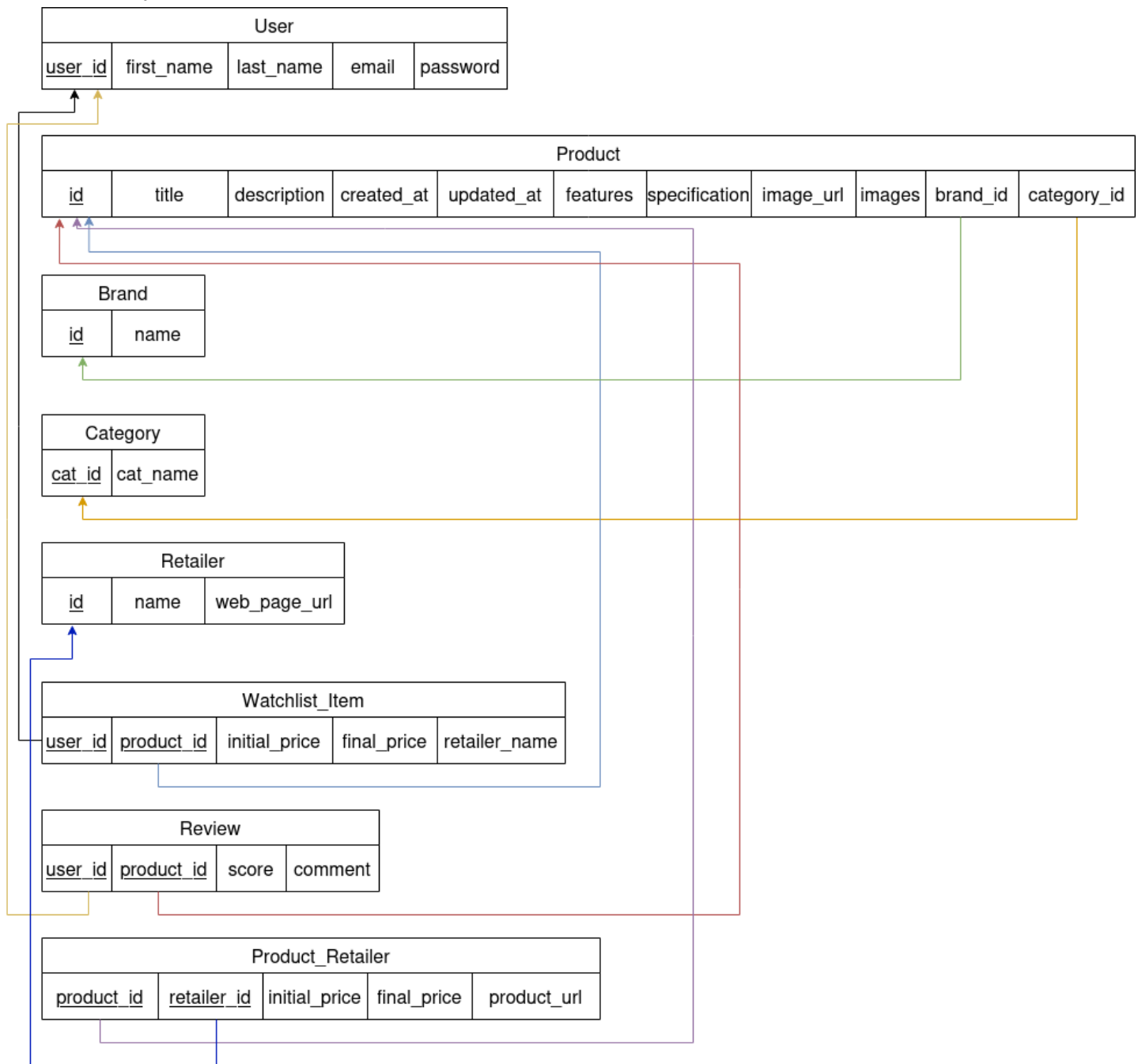
There are no multivariable attributes. It can be argued that images in products is multivariable, but we decided against it, as it will be stored as an array of string urls, which can easily be parsed by our api and website to extract data from. The alternative solution is to make a separate relation with both as primary keys, and joining both tables whenever a specific product is accessed, which we decided was less efficient than just keeping it there.

Step 7: N-ary relations

No N-ary relationships included in the final version of the diagram.

Step 8: Mapping of specialization and generalization

Option C was used from the lecture slides, as well as removing the permissions field. This is because the type implies which permissions you have. The end program will be able to decide what those permissions are based on the type.

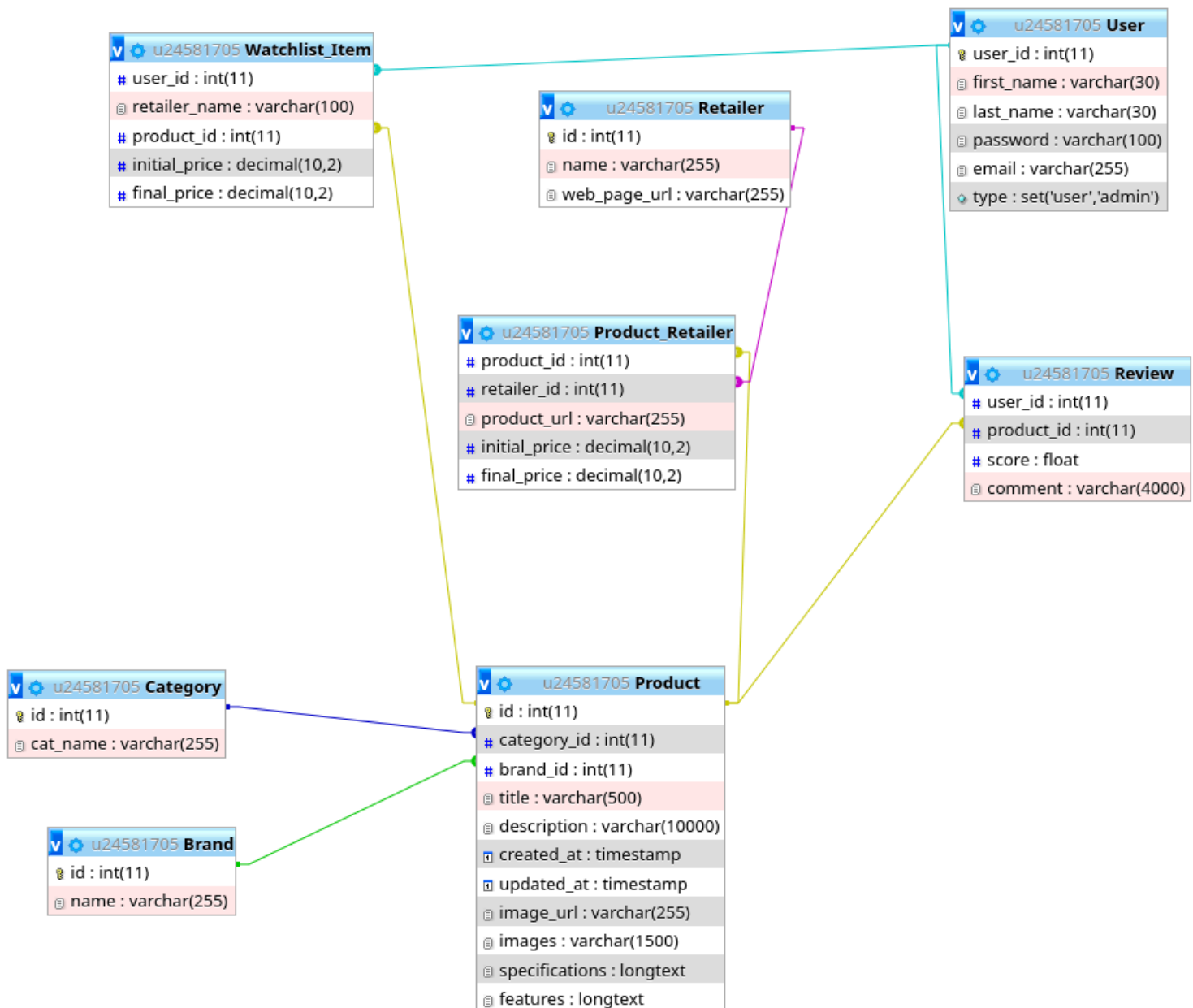


Step 9: Mapping Unions

There were no unions in our diagram

Task 4: Relational Schema

Visual Representation



Primary keys are denoted by the yellow key symbol,

Indexes/Secondary Keys

Review

- `user_id`
- `product_id`

Product_Retailer

- `product_id`
- `retailer_id`

Product

- brand_id
- category_id

Watchlist_Item

- user_id
- product_id

Lengths and checks

- Password: Bcrypt has a fixed hashing length, so the password is a bit longer than should be reasonable, to include the bcrypt hash
- Comments are 4000 characters long to store a comment.
- Description is 10000 characters long since it needs to store the entire product description.
- Score is a float, since precision isn't strictly necessary, and math won't be performed on it regularly.
- Values are decimal, to ensure precision, at the cost of some efficiency,
- Images is 1500 since it can store multiple urls.
- Longtext is for json data.
- Varchar is used even for large elements, since it is able to be indexed by the dbms unlike TEXT.
- By default relational integrity is update cascade, and reject remove.

Task 5: Web Application

Maybe include a screenshot of every web page just to showcase

Or use the headings from the memo

Task 6: Data

Database Population Method

Overview of the Data Population Approach

The project uses python scripts that process and transform raw data from various source files into a structured SQL database. This method combines pre-defined data sets with transformation to create a comprehensive database with products, retailers, brands, categories, and user reviews.

Data Sources

The data population relies on several source files, which were either hand-written or exported from the database used for COS216 Practicals, hosted on Wheatly.

- brands.sql - Contains brand names for products. Exported from Wheatly.
- categories.sql - Contains product categories. Exported from Wheatly.
- prices.sql - Contains price information. Exported from Wheatly.
- products_with_text_brand.sql and products_with_text_category.sql - Contains product information with text references to brands and categories. Exported from Wheatly.
- retailers.txt - Contains retailer information. Hand-written.
- reviews.json - Contains user review data. Hand-written.
- users.sql - Contains user account information. Hand-written.

- dimensions.sql - Contains product dimension data. Exported from Wheatly.
- bulk.sql - Contains bulk product data. Exported from Wheatly.

Data Processing Pipeline

The data population process follows an ordered sequence of scripts that build upon each other:

- 1. Brand Processing ([1brands.py](#))
 - Parses brand names from brands.sql
 - Maps text brand names to numeric IDs
 - Generates products_with_brand_ids.sql
- 2. Category Processing ([2categories.py](#))
 - Processes category data from categories.sql
 - Maps text categories to numeric IDs
 - Generates products_with_category_ids.sql
- 3. Specifications Generation ([3specifications.py](#))
 - Combines brand, category, and dimension data
 - Creates JSON specifications for products
- 4. Data Consolidation ([4merge.py](#))
 - Merges outputs from previous scripts and bulk.sql
 - Creates a unified product dataset
- 5. Retailer Setup ([5retailers.py](#))
 - Processes retailer information from retailers.txt
 - Generates retailers.sql with structured retailer data
- 6. Price Conversion ([6currency.py](#))
 - Processes price data from prices.sql
 - Converts prices to ZAR
 - Generates prices_in_zar.sql
- 7. Product-Retailer Relationships (product_retailers.py)
 - Creates random relationships between products and retailers
 - Assigns random ranged prices to product-retailer combinations
 - Generates product_retailers.sql
- 8. Review Processing ([reviews.py](#))
 - Processes review data from reviews.json
 - Randomly links reviews to products and users
 - Generates reviews.sql

Final Database Creation

The run.py script controls the script execution and creates a final SQL file (DROP-TABLE-COMPLETE.sql) that

includes:

- 1. The database schema from DROP-TABLE.sql
- 2. All the processed data in the correct order to maintain referential integrity:
 - Brands and categories (no foreign keys)
 - Users (no foreign keys)
 - Retailers (no foreign keys)
 - Products (references brands and categories)
 - Product-retailer relationships (references products and retailers)
 - Reviews (references users and products)

Data Generation Techniques

The implementation uses several techniques to generate data:

- **Deterministic Randomisation:** The `product_retailers.py` script uses seeded random number generation (`random.seed("DROP TABLE")`) to ensure reproducible results while still creating varied data.
- **Price Variation:** Product prices are generated with realistic variations:
 - Base prices are taken from the price data
 - Retailer-specific scaling factors (0.85-1.15) simulate market competition
 - Weighted discounts using triangular distribution create realistic final prices
- **Relationship Modeling:** Each product is associated with a random number of retailers, simulating availability patterns.

Database Schema

The database schema (`DROP-TABLE.sql`) defines the structure for:

- Brands and categories
- Products
- Retailers
- Product-retailer relationships
- User accounts
- Reviews
- Watchlists

Advantages of This Approach

1. **Reproducibility:** The scripted approach ensures the database can be regenerated consistently.
2. **Scalability:** The scripts can process large datasets efficiently.
3. **Realistic Data:** The transformation process maintains realistic relationships between entities.
4. **Referential Integrity:** The final SQL generation respects database constraints and foreign key relationships.
5. **Maintainability:** The modular script approach makes it easy to modify individual aspects of the data generation process.

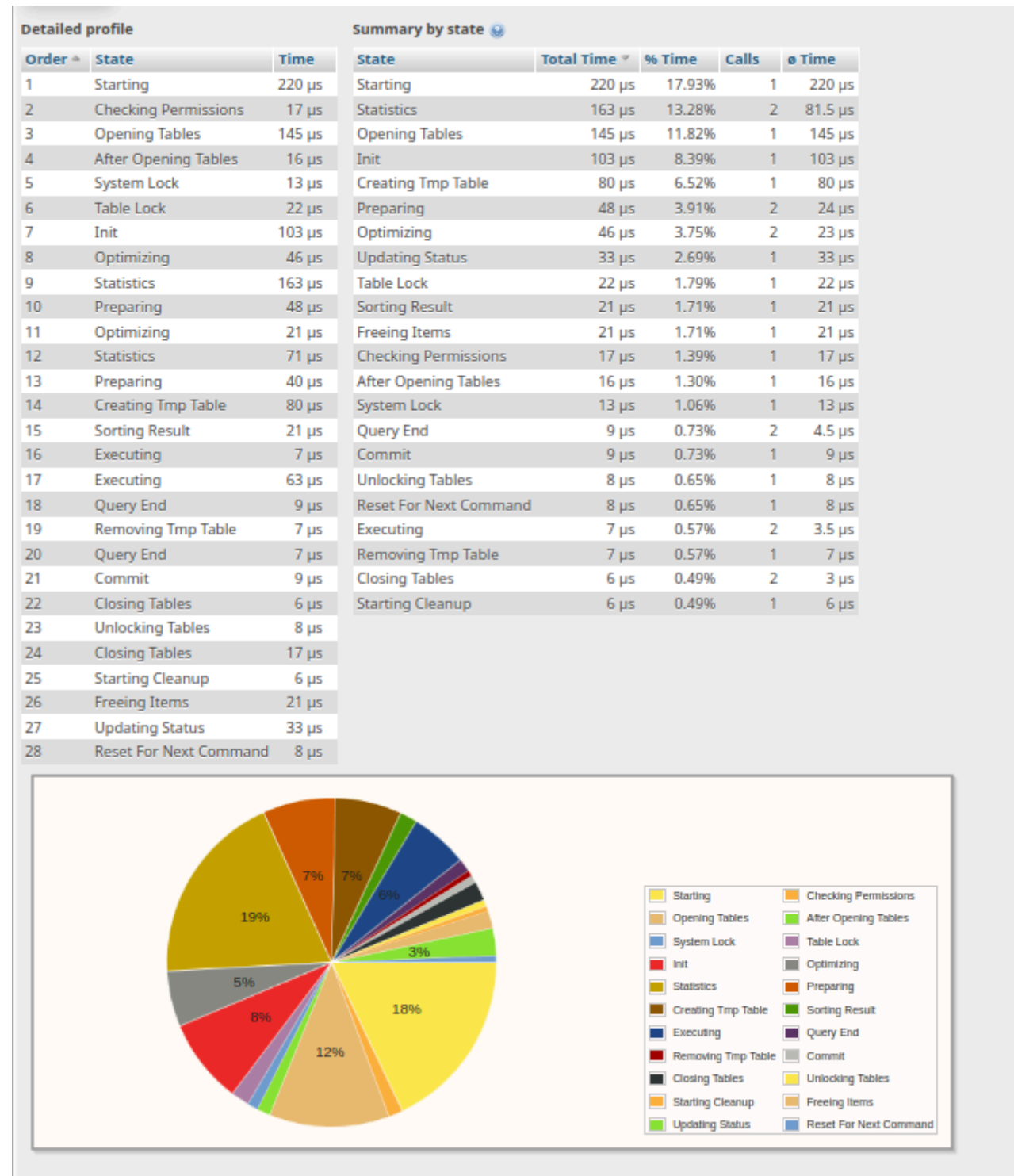
Task 7: Analyze and Optimise

View All Products

Query

```
SELECT P.id, P.image_url, P.title, PR.final_price, PR.initial_price, R.name, R.id
AS rID, ((PR.initial_price - PR.final_price)/PR.initial_price) AS Discount,
AVG(RT.score) AS Rating, B.name AS brand, C.cat_name, CASE WHEN EXISTS (SELECT 1
FROM Watchlist_Item WHERE user_id = 0 AND product_id = P.id) THEN TRUE ELSE FALSE
END AS watchlist FROM Product AS P INNER JOIN Product_Retailer AS PR ON P.id =
PR.product_id INNER JOIN Retailer AS R ON R.id = PR.retailer_id INNER JOIN Brand
AS B ON B.id = P.brand_id INNER JOIN Category AS C ON C.id = P.category_id LEFT
JOIN Review AS RT ON RT.product_id = P.id GROUP BY P.id, C.id, B.id, R.id ORDER
BY RAND()
```

Analysis



Interpretation:

The query is able to execute quickly taking 0.26 seconds to execute with 0 filters applied, and much quicker if even a single filter is applied, taking 0,03 when filtering by electronics. The reason for this is that the database contains indexed fields for product_retailers and brands and reviews and such, which causes a B-Tree structure to form when that data is stored. This does increase the size of the database to store index information, but causes the query to execute much quicker, as the dbms is able to use indexed joins.

Task 8: Git usage

Division of Project

We divided the project into 3 different sections, The frontend, the javascript, and backend API. The frontend section involved the visual design of the frontend (HTML and CSS), the javascript section included the frontend javascript for client-side functionality and coupling with the API, and finally the API section is the REST API which handles requests from the frontend javascript and returns data from the database.

The group was split into 3 teams, 1 for each section. Once a section was completed, the members of a team could move to a different section to increase productivity. Each section includes its own [README.md](#) file which documents the features of the section and how they function to ensure efficient coupling of the sections into a complete product later on. The collection of README files is the documentation of the product, detailing all functionality.

Feature Development

Each section branched into their features. Examples of these features include: feature/database-connection for the API, FE/html-and-css for frontend and feature/frontend-js for javascript. Any development on a feature was performed on a new branch based on a feature (E.g. API/GetProduct is a branch of feature/database-connection to develop the /Get/Product endpoint), and later merged into the feature files. The development branch is then deleted following the merge to maintain an orderly structure within the DROP-TABLE Github.

Contributions

API

- Stephan: SQL Queries, Rest of the endpoints, quality control, Documentation
- Byron: 40% of Endpoints, Documentation
- Caitanya: 40% of Endpoints
- Graeme: SQL Queries
- Michael: Documentation draft, requirements analysis
- Dandre: Documentation draft, requirements analysis

Frontend

- David: Designing and implementation.

JS Backend

- Michael: Login, Register, Settings, Admin
- Dandre: Products, Deals, Settings, Reviews, Watchlist

Schema:

- Caitanya: Original Schema Draft
- Stephan: Improved Schema, creation of mariadb user

Database Population

- David: Python scripts to generate dump

(E)ER-D

- Everyone in a meeting, revisions later made by David

Relational Mapping

- Stephan: Conversion of (E)ER-D to Relational Mapping

PDF construction

- Byron: Github section, Powerpoint construction
- David: Data Population
- Stephan: Analysis and Optimization, Bonus Features

Github Management:

- Stephan: Reviewing API changes, and merging branches
- David: Merging JS into the frontend.

Research

- Everyone in an initial meeting.

Final Structure

The final structure of the DROP-TABLE GitHub consists of the following branches:

- feature/coupling : Details the final coupling between the frontend javascript and API sections.
- dev and data/data-manipulation: Details the database development.
- fe/html-and-css: Details the frontend section, specifically the HTML and CSS.
- feature/API/database-connection: This is the final REST API section.
- feature/API/testing: This branch was used to coordinate testing the REST API.
- feature/frontend-js: This branch details the final frontend javascript.
- feature/API/queries: This is a branch for various queries included in the REST API.
- frontend

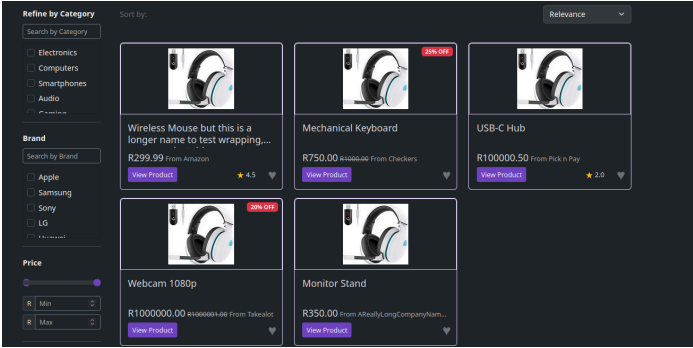
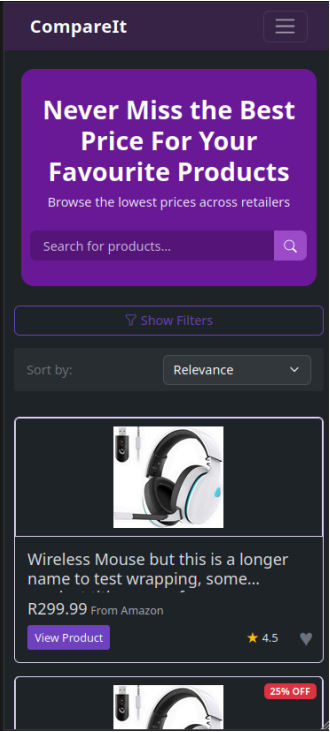
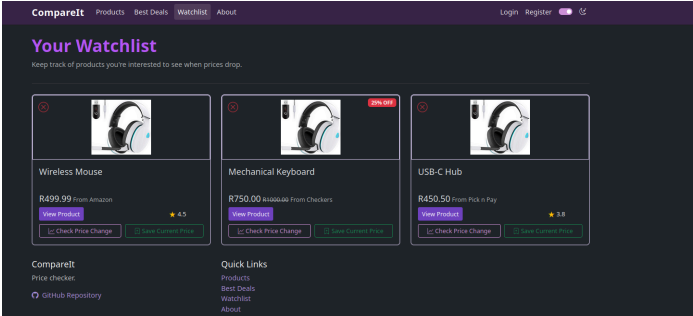
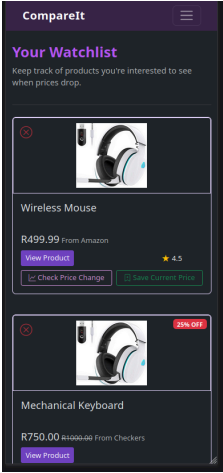
Task 9: Demo

Check the github for the demo slides.

Task 10: Bonus Features

Feature 1: Enhanced UI

Feature 1.1: Mobile UI


Desktop	Mobile
	
	


CompareItProductsBest DealsWatchlistAboutLoginRegister


About Price Checker

Our Mission
Price Checker helps you find the best deals across multiple retailers in South Africa. We compare prices for you so you never miss out on savings.

How It Works

**Search**
Find products across multiple retailers

**Compare**
See prices from different stores side by side

**Track**
Add items to your watchlist for price alerts

Our Features

- Price Comparison**
Compare prices across major South African retailers
- Price History**
View price trends over time to make informed decisions
- Watchlist**
Save products to track price changes

Supported Retailers


GameCheckersPick n PayWoolworthsInt


CompareIt


About Price Checker


Our Mission
Price Checker helps you find the best deals across multiple retailers in South Africa. We compare prices for you so you never miss out on savings.

How It Works

**Search**
Find products across multiple retailers

**Compare**
See prices from different stores side by side





Sony WH-1000XM4 Wireless Noise Cancelling Headphones

R4999.99 ~~R5999.99~~ **15% OFF**

From Takealot **4.8** (1245)

[Add to Watchlist](#)[View all retailer prices](#)

Description

Industry-leading noise cancellation with Dual Noise Sensor technology. Next-level music with Edge-AI, co-developed with Sony Music Studios Tokyo. Up to 30-hour battery life with quick charging (10 min charge for 5 hours of playback). Touch Sensor controls to pause/playback tracks, control volume, activate your voice assistant, and answer phone calls. Speak-to-chat technology automatically reduces volume during conversations.

Specifications


Brand:	Sony	Model:	WH1000XM4
Color:	Black	Connectivity:	Bluetooth 5.0
Battery Life:	30 hours	Charging Time:	3 hours

Key Features

- Industry-leading noise cancellation
- 30-hour battery life
- Speak-to-chat technology
- Touch sensor controls
- Wearing detection with smart playback
- Seamless multiple-device pairing

CompareIt

[← Back to Products](#)

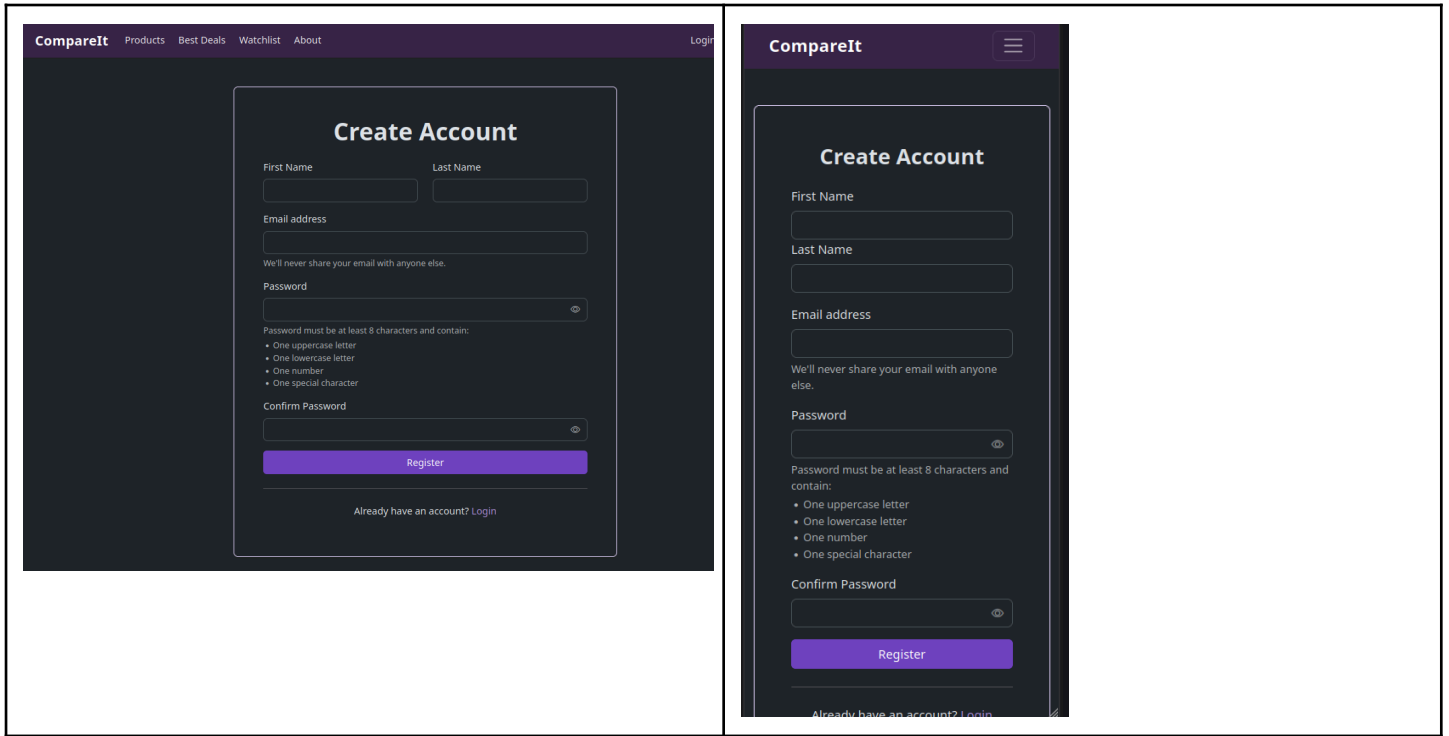


Sony WH-1000XM4 Wireless Noise Cancelling Headphones

R4999.99 ~~R5999.99~~ **15% OFF**

From Takealot **4.8** (1245)

[Add to Watchlist](#)[View all retailer prices](#)



Feature 1.2: Contrast

Contrast Checker

[Home](#) > [Resources](#) > Contrast Checker

Foreground

Hex Value
#D7DBDF

Color Picker Alpha 1

Lightness

Background

Hex Value
#212529

Color Picker

Lightness

Contrast Ratio
11.08:1

[permalink](#)

Normal Text

WCAG AA: **Pass**
WCAG AAA: **Pass**

The five boxing wizards jump quickly.

Contrast Checker

[Home](#) > [Resources](#) > Contrast Checker

Foreground

Hex Value
#F1EBF6

Color Picker Alpha 1

Lightness

Background

Hex Value
#6A1B9A

Color Picker

Lightness

Contrast Ratio
8.02:1

[permalink](#)

Normal Text

WCAG AA: **Pass**
WCAG AAA: **Pass**

The five boxing wizards jump quickly.

Foreground

Hex Value
#F5F3FB

Color Picker

Alpha
1

Lightness

Background

Hex Value
#6F42C1

Color Picker

Lightness

Contrast Ratio
5.92:1

[permalink](#)

Normal Text

WCAG AA: **Pass**
WCAG AAA: **Fail**

The five boxing wizards jump quickly.

Large Text

WCAG AA: **Pass**
WCAG AAA: **Pass**

The five boxing wizards jump quickly.

Graphical Objects and User Interface Components

WCAG AA: **Pass**

★
Text Input

Used for graphics of deals so its fine

Foreground

Hex Value
#FDF5F6

Color Picker

Alpha
1

Lightness

Background

Hex Value
#DC3545

Color Picker

Lightness

Contrast Ratio
4.21:1

[permalink](#)

Normal Text

WCAG AA: **Fail**
WCAG AAA: **Fail**

The five boxing wizards jump quickly.

Large Text

WCAG AA: **Pass**
WCAG AAA: **Fail**

The five boxing wizards jump quickly.

Graphical Objects and User Interface Components

WCAG AA: **Pass**

★
Text Input

Used for hyperlinks

Foreground

Hex Value
#A48AD4

Color Picker

Alpha
1

Lightness

Background

Hex Value
#212529

Color Picker

Lightness

Contrast Ratio
5.26:1

[permalink](#)

Normal Text

WCAG AA: **Pass**
WCAG AAA: **Fail**

The five boxing wizards jump quickly.

Large Text

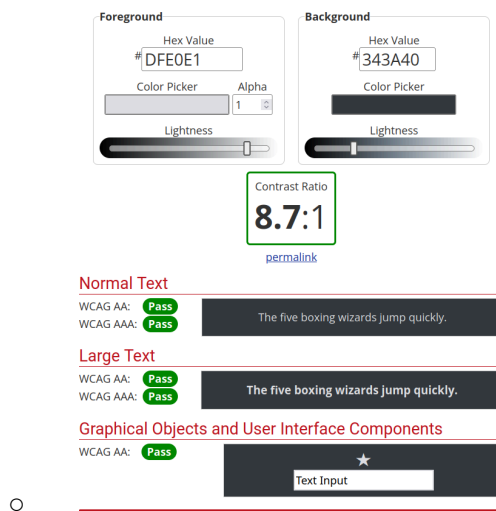
WCAG AA: **Pass**
WCAG AAA: **Pass**

The five boxing wizards jump quickly.

Graphical Objects and User Interface Components

WCAG AA: **Pass**

★
Text Input



Feature 1.3: Dark Mode

The ability to switch the website from light to dark mode and vice versa. The default is dependent on browser settings.

Feature 2: Security

Feature 2.1: Hashed Passwords

- Uses Bcrypt to hash passwords
- We chose BCrypt as it rates among the highest of secure hashing algorithms, due to being based on blowfish. BCrypt is a memory hard hashing algorithm, which means that it is computationally expensive to brute force the hash. It also uses salts, and stores them in the hash itself, which makes it convenient to store and verify with.
- BCrypt is computationally expensive, but since it will only be used on registration, where some optimisation can be given up, for the additional security provided, it is worth it.
- We take the plain text password and hash it without a salt using SHA-256, which is then sent to the API. This hashed password is then hashed again using the same algorithm, and this time with salt for added security. This new doubly hashed password is then stored on the database.
- SHA-256 is proven to be insecure, but the first hash is required to be consistent for the 2nd hash to be able to work. SHA-256 was chosen due to its speed and support for most browsers, and just to slow down man-in-the-middle attacks. If the hash is retrieved by an attack it cannot be used to login into the website, and therefore a rainbow table would need to be used to get the password, which takes time, as opposed to sending the password in plain text and being able to use the password directly.
- As long as the password meets the strength requirements, the hash shouldn't be on a rainbow table.
 - Passwords tested against crackstation.net with the SHA hash alone, not the BCrypt hash
 - Password - Found (But doesnt meet strength requirements)
 - Password1! - Found
 - December_12_1987 - Not Found
 - 7[4tYTx4r9EZr£=T - Not Found
 - My_Password_Num2 - Not Found
 - Password_123 - Not Found
- Reference: L. Ertaul, M. Kaur, VAKR. Gudise, "Implementation and Performance Analysis of PBKDF2, Bcrypt, Scrypt Algorithms," In Proceedings of the international conference on wireless networks (ICWN), 2016, p. 66, [Online]. Available: <http://mcs.csueastbay.edu/~lertaul/PBKDFBCRYPTCAMREADYICWN16.pdf>

Feature 2.2: Error Log

- We use an error log that is stored on the api server whenever an error occurs. This is used as to not expose the program stack to users in the case of an error, and an ambiguous message stating that something went wrong is instead sent to the user in the form of a 500 Response code.