**Q1. *Explain the problem that this app will solve and explain how this app solves or addresses the problem.***

Brands and consumers often face challenges when trying to make informed decisions when buying quality products, as not all products are created equally. With an overwhelming amount of products to choose from both online and instore, it can be difficult to discern which products are indeed high-quality and worth the price tag and the investment. Consumers frequently encounter some of the following concerns when purchasing products.

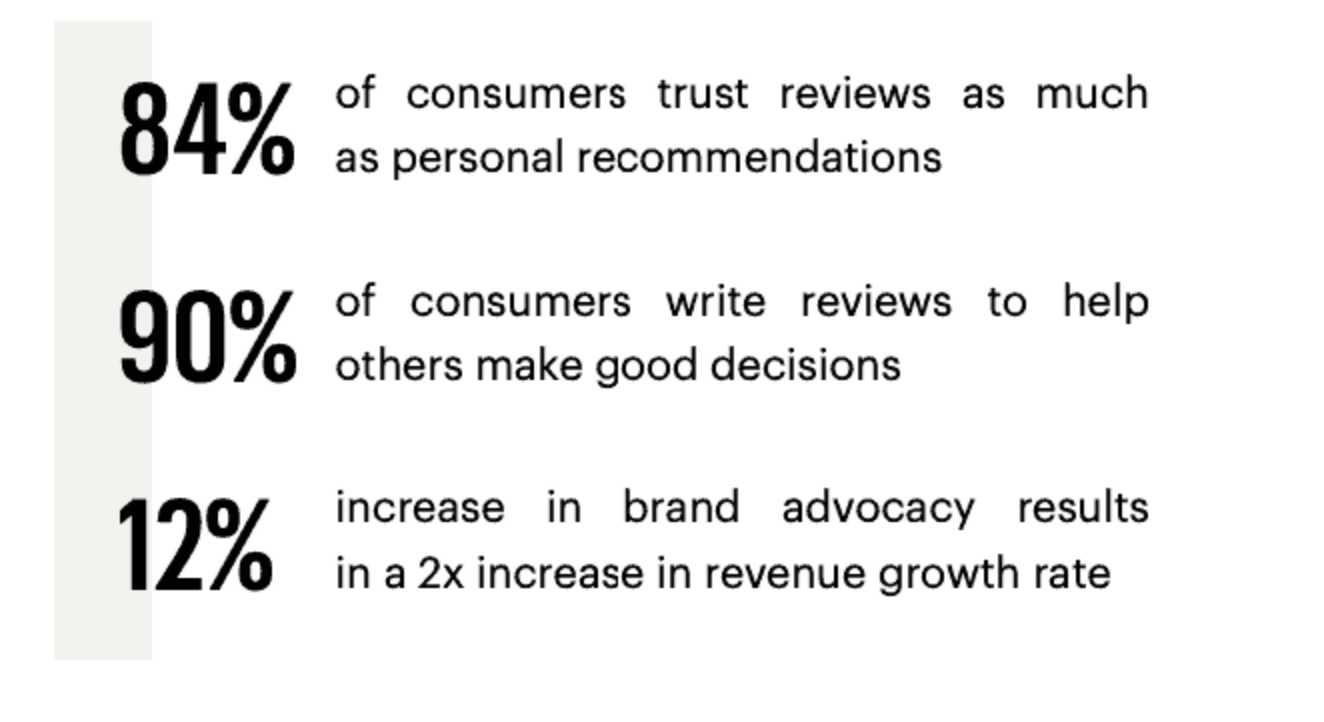
**Overwhelming choices** – The sheer volume of products available on the market can lead to decision stalemate.

Inconsistent review – Not all review are trustworthy, biased, fake or bought reviews can skew people perceptions, and people are unsure if the reviews are legitimate.

**Lack of Personalisation** – Reviews that don’t tailor to customers’ needs and individual preferences.

**Difficulty Comparing Products** – Consumers struggle to compare the features, prices and user experiences, as again consumers are unsure of trustworthy reviews.

This app solves the problem of legitimacy, as it is an independent app where people can post their honest reviews, good bad or indifferent, it can be customized into categories. This app will incorporate customer reviews and/or buying guidelines to enhance the credibility of the product being reviewed, whereby helping consumers make more informed decisions. It will foster a community of users who can ask questions, share their experiences, and provide real-time feedback, creating a more interactive environment.



According to surveys, about 84% of consumers trust online review as much as personal recommendations. This highlights the significance of reviews in consumers decision making. 90% of consumers write reviews to help others make good decisions. Research indicates that 70% of consumers advise they research a product before making a purchase and read on average 7 reviews before trusting a product. Products with positive reviews can increase sales but up to 18% while a negative review can deter around 22% of potential consumers. While 12% in brand advocacy results in an increase of two times in revenue.

With the introduction of iPhones, mobile phone shopping is accounted for more than 50% of consumers purchases, with a strong lean to those looking at product reviews while in store before purchasing.

This emphasises the need for accessible review platforms. These statistics illustrate the critical role product review apps play in shaping consumer behaviour and driving sales, thus highlighting their value in the ecommerce landscape

**Q2. *Describe the way tasks are allocated and tracked in your project.***

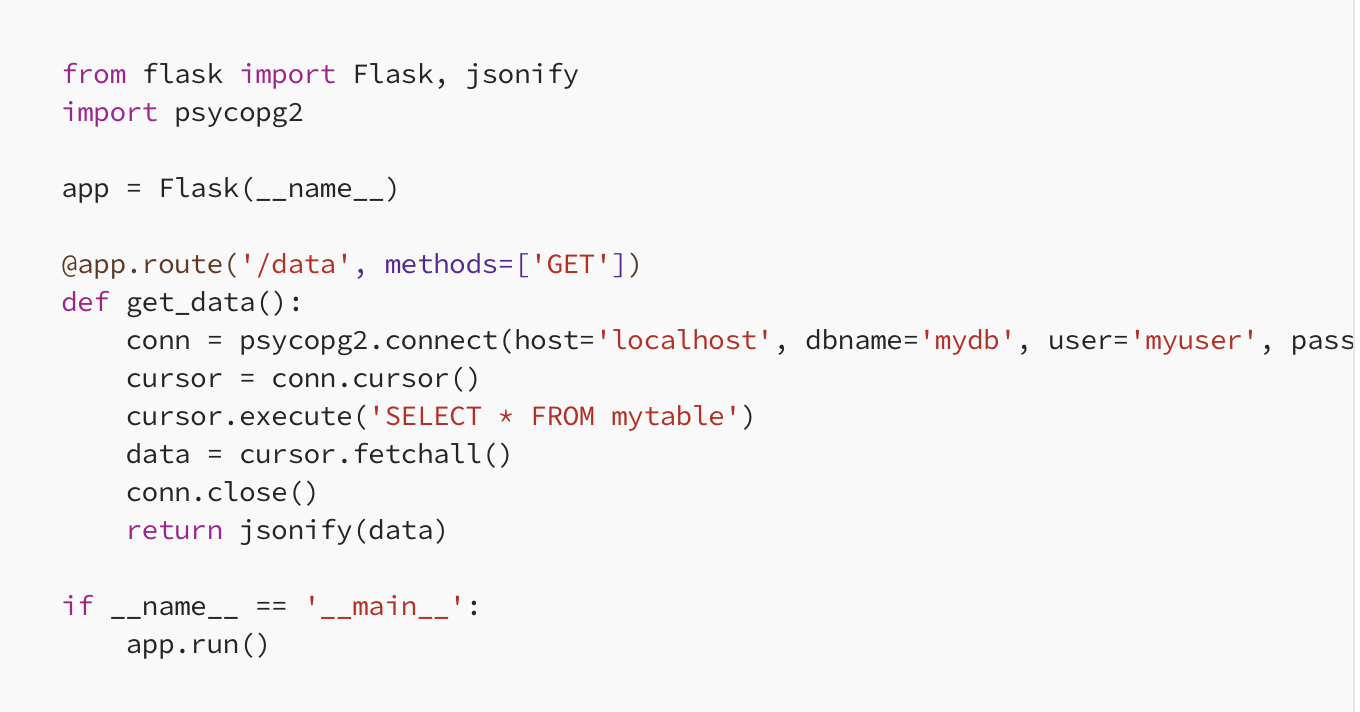
**Q3. *List and explain the third-party services, packages and dependencies used in this app.***

**Q4. *Explain the benefits and drawbacks of this app’s underlying database system.***

Flask is a standard Python web structure that is used to craft RESTful APIs for data engineering tasks. It provides a flexible and simple way to build your API’s that can handle HTTP requests and responses. Flask API projects commonly use PostgreSQL database systems. PostgreSQL is an open-source relational database management system which is known for its extensibility, reliability, data integrity, robustness, and extensive feature set. It can handle advanced data types, complex queries, foreign keys, triggers, and views as well as procedural languages for stored procedures. Its high expandability, allowing users to add new functions and data types. Its strong SQL compliance and combined support for ACID (Atomicity, Consistency, Isolation, Durability) properties, make it an efficient, scalable and secure database choice for developers.

Below are some of the examples of using Flask.

1. **Data Retrieval**: Flask can be used to construct APIs to retrieve data from databases, files or external API’s. See example below; When you access the http address in your browser it will retrieve data from the PostgreSQL database and return it as a JSON response.



**Data Transformation**: Flask can be utilised to construct API’s to perform data calculations. Here is an example that calculates the sum of two numbers.

A screenshot of a computer program

Description automatically generated

1. **Data Ingestion**: Flask can be used to construct API’s that accept data uploads or input data from external sources. Example below of accepting a file upload and saves to the server.

A screenshot of a computer program

Description automatically generated

These are just a few examples of the ways in which Flask can be used in data engineering tasks

Look at the Statistics

“StackOverflow statistics show that 26% of developers preferred it in 2017, 34% in 2019, and 40% in 2021. Most recently, in StackOverflow’s  [2022 Stack Developer Survey](https://survey.stackoverflow.co/2022/), PostgreSQL took a slight lead over MySQL (46.48% to 45.68%) as the most popular database platform among professional developers.“ Author -Pete Scott – Percona Feb 2, 2024.

**The Pros and Cons of using PostgreSQL in Flask;**

***Pros of Using PostgreSQL***

**Advanced Features**Acid Compliance – it ensures data integrity is maintained by supporting transactions and rollback capabilities.

**Extensibility**  
It allows you to define custom data types, operators and index types. It also supports extensions like PostGIS for geographical/geospatial data, which is useful for location-based applications.

**Scalability**PostgreSQL performs well on large scale datasets and is capable of handling high workloads by vertical scaling (adding more powerful hardware) With tools like Citus, PostgreSQL can also scale horizontally distribution data across many nodes.

**SQL Compliance**PostgreSQL adheres closely with standards compliance making it easier for developers to work with who are familiar with SQL. It also supports advanced SQL window functions, these are valuable when using analytical queries.

**Community and Available Documentation**PostgreSQL has a strong active community, who provide support and contribute to further developing. It’s highly detailed documentation assists developers utilise its systems effectively.

***Cons of Using PostgreSQL***

**Complexity**PostgreSQL can be complex to configure and tune for optimal performance, especially on large scale projects. New users may find it hard to learn, particularly those unfamiliar with the advanced database features and functions.

**Performance Overhead**While PostgreSQL excels in read heavy scenarios, write heavy scenarios might encounter performance overhead due to its consistency and transactional features. Also, its indexing and query optimization, PostgreSQL complex queries and indexing can sometimes create performance issues when not properly optimized.

**Resource Usage**  
Disk and memory usage can be resource intensive, it uses significant amounts of disk space and memory, which may be a concern for certain constrained environments.

**Tooling and Ecosystem**  
Compared to other databases there may be limited tools and third-party options, integrations specifically designed for PostgreSQL are continuously improving.

**Replication**Complex replication setups and high availability may be involved and require additional configurations or tools.

For many Flask applications, PostgreSQL is a robust choice, because of its strong feature set, community support and complex data requirements, its advanced features like full-text search and custom data types, couple that with high data integrity and consistency, it’s understandable why its popular. However, it is still essential to base your applications specifics on whether PostgreSQL is suitable for your project’s needs.

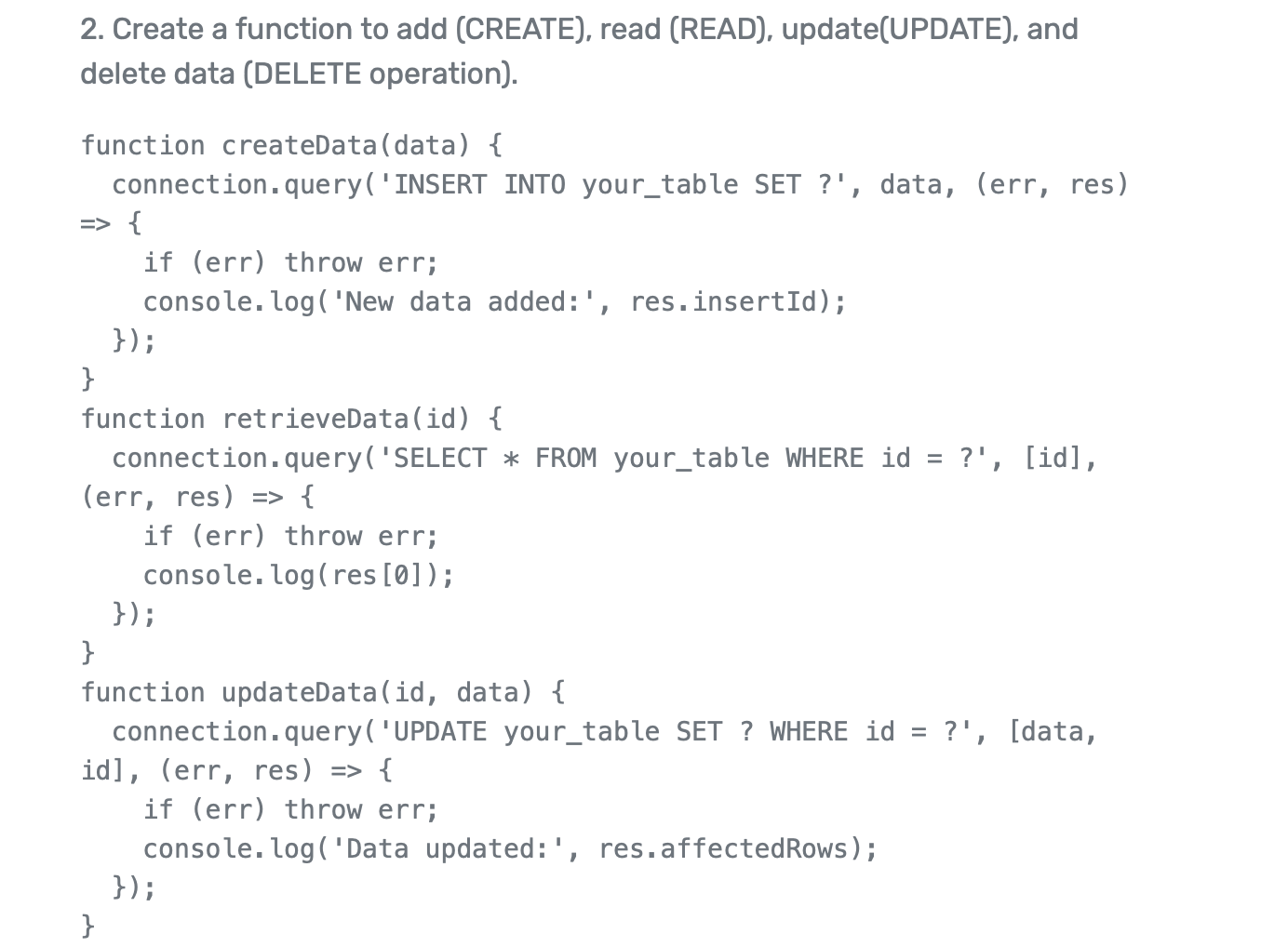
**Q5. *Explain the features, purpose and functionalities of the object-relational mapping system (ORM) used in this app.***

The primary purpose of the ORM system in the product review app is to streamline the database interactions, improve developer productivity and enhance maintainability and quicker feature development for easier updates. This is a brief list of features below.

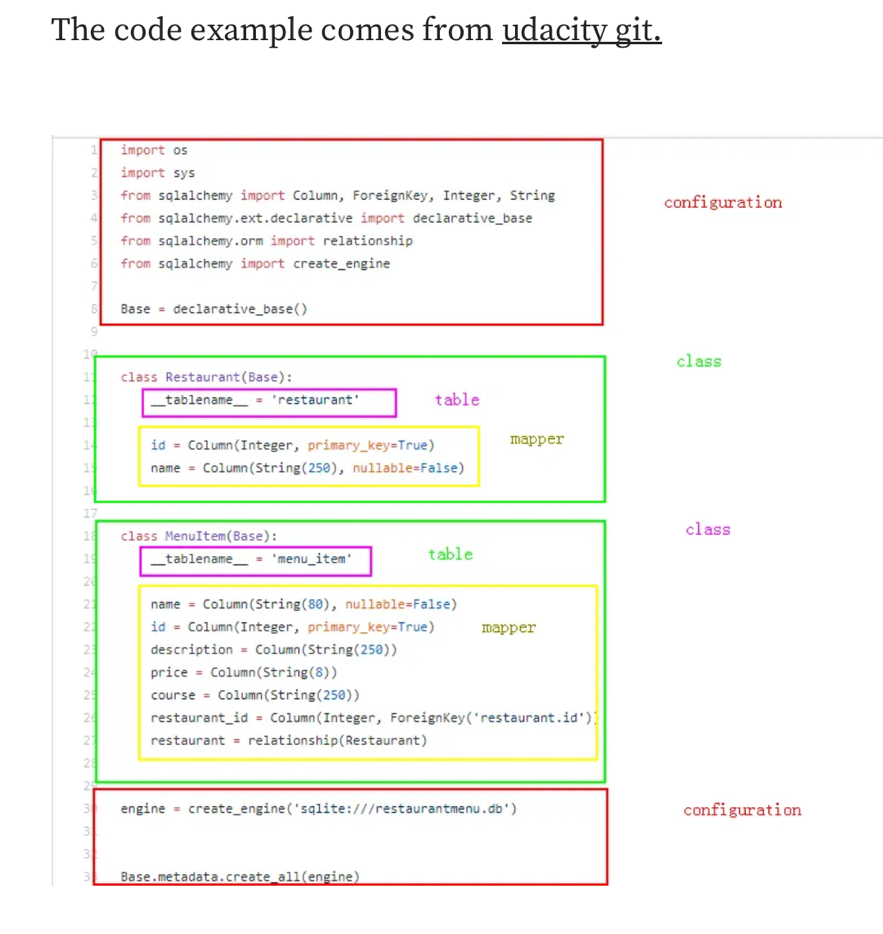
* ORM provides a high-level construct over the database interactions and allows developers to work with the objects rather that the raw SQL queries by simplifying the database operations.
* It forces you to write MVC code, which makes your code a little cleaner.
* By mapping classes to database tables, this makes it easier to manage and manipulate the data.
* Sanitizing prepared statements or transactions are as easy as calling a method.
* Each class represents a table, and each instance of the class represents the row of each table.
* ORM automatically generates SQL queries based on the object operations, this then reduces the need for manual SQL coding which minimises errors.
* It keeps track of changes made to objects, ensuring only modified data is sent to the database.
* It handles relationships between the objects (1 to many, many to many) using associations, to simplify and manage the complex data relationships.
* ORM has built in support for transactions management to ensure data consistency and integrity.

By incorporating an ORM system the product review app can efficiently manage the data, streamline development processes and ensure that the application is scalable and maintainable, some of ORM’s functionalities below.

* CRUD operations like CREATE, READ, UPDATE, DELETE through object manipulation.

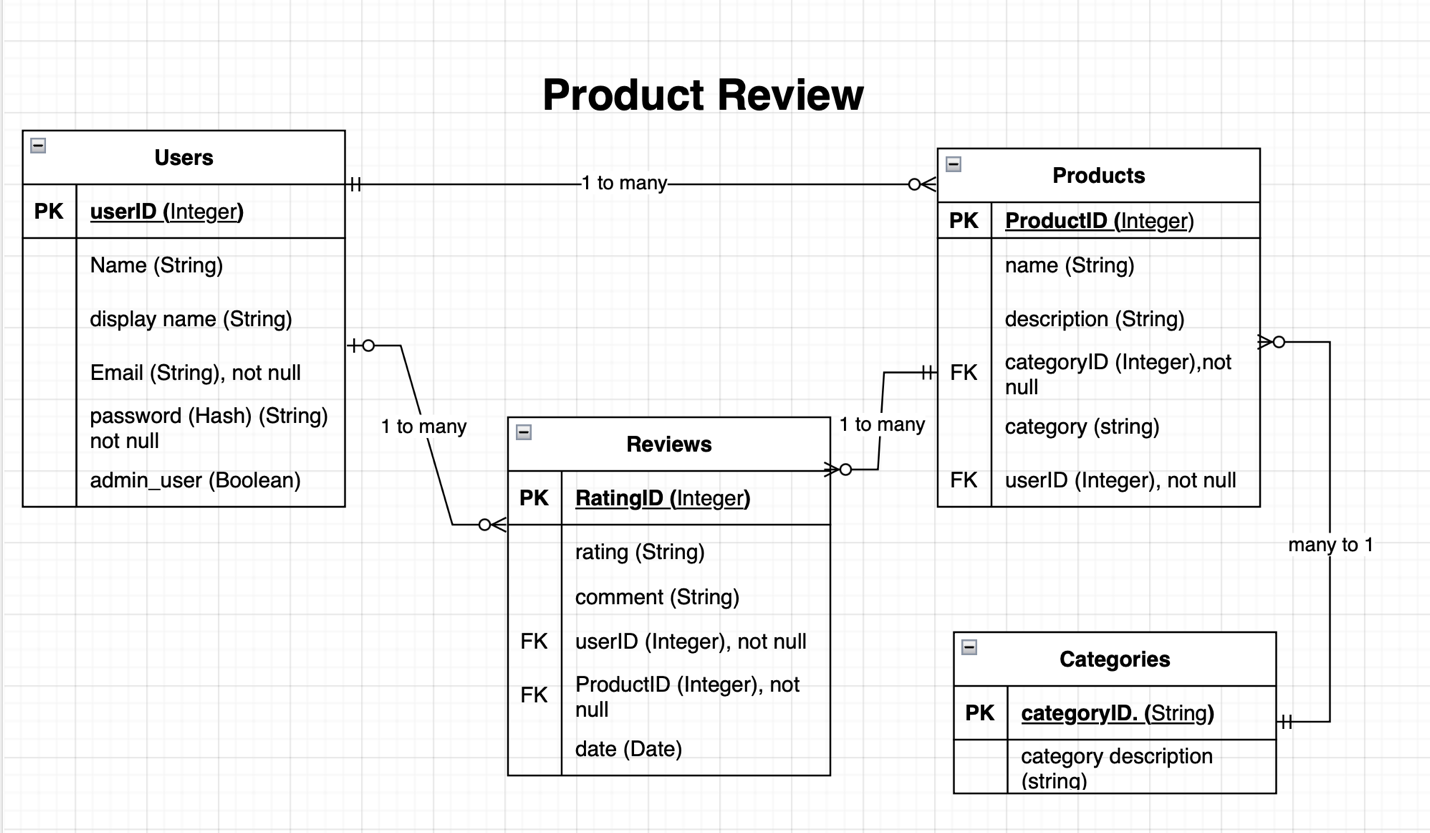


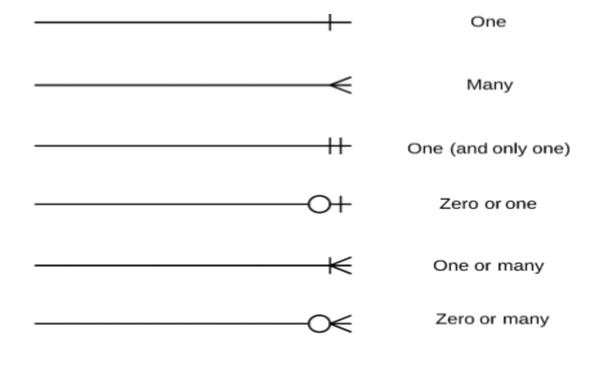
* Data Validation which provides built in validation features to ensure data integrity.
* Supports complex queries for filtering through object-oriented interfaces to allow developers to retrieve data efficiently.
* Implements caching to reduce database load, it stores frequently accessed data efficiently for improved application performance.
* Easy database schema migrations, make managing changes in the database structure easier over time.
* Offers a multi database system support, allowing the app to switch databases with minimal code changes.



**Q6. *Design an entity relationship diagram (ERD) for this app’s database and explain how the relations between the diagramed models will aid the database design.***

***This should focus on the database design BEFORE coding has begun, eg. during the project planning or design phase.***





An Entity Relationship diagram is a visual representation of the entities in a database system and their relationships to one another. For the product review app, the ER diagram typically includes entities such as Users, Products, Reviews and Categories. Below is the breakdown of each entity table.

**Users**

This entity represents the users of the app, storing their details, and passwords safely (hashed). Each user can submit multiple reviews.

* UserID (Primary Key) and not null
* Name
* Display Name
* Email, not null
* Password (hash)
* Admin\_user

**Products**

This entity represents the products being reviewed in the app. Each product can have multiple reviews from different users.

* ProductID (Primary Key) and not null
* Name
* Description
* CategoryID (Foreign Key), not null
* Category
* UserID (Foreign Key), not null

**Categories**

This entity represents the categories that products get listed under to help with searchability. A category can have many products, but a product can only be listed under one category.

* CategoryID (Primary Key), not null
* Category description

**Reviews**

This entity is the Junction table, it connects users and products together. Users can leave many reviews, and a products can have many reviews. It stores the rating and the comment for each product review.

* RatingID
* Rating – 1-5 rating
* Comment
* UserID (Foreign Key)
* Product ID (Foreign Key)
* Date – datetime delta

As you can see there are a few entities that have relationships to each other in the form of Foreign Key connections. These connects are important in the way you manage the data in the database.

**Users** table has a 1 mandatory relationship to many optional relationships with the Products table. Being that one user can list many products to review, and many products can be listed by one user.

**Categories** table has a relationship with the Product table, being a one mandatory relationship to a many products relationship. So, a product can have a mandatory category and only one category however a category can have many products under the one category. For example, a product category of toys, can have many toy products like, wooden blocks, Lego, barbie dolls, stuffed toys etc.

**Products** table has a relationship with the User table, Categories table and the Review table. The relationship with both the Users table and the Categories table has been discussed above.

**Reviews** table is where it all comes together as a Junction table, it is linked to the Users table via the Foreign Key and is linked with a one optional to many optional relationship. Reviews is also linked to Product table via its Foreign Key with its one mandatory product to many optional reviews relationship, it also pulls from the products relationship with categories to pull that data into its junction table.

This ER diagram has a foundational structure which allows for efficient data management and retrieval.

**Q7. *Explain the implemented models and their relationships, including how the relationships aid the database implementation.***

***This should focus on the database implementation AFTER coding has begun, eg. during the project development phase.***

The entity relationships help to shape and design the database.

Data Integrity with the use of Foreign Keys (FK) ensures that each product review is associated with a valid user and valid product. This helps maintain referential integrity in the database helping to prevent orphaned records. By implementing Foreign Keys, the database is ensured that every review has to correspond to a valid user and product.

Query efficiency is maintained by the relationships, for example you can retrieve all reviews for a specific product or find all reviews written by a certain user. The relationship helps to support the apps functionality by making data retrieval streamlined.

Scalability, with the one-to-many relationships enable the system to scale more easily. As more users and products get added to the database, the structure can accommodate the growth without requiring major redesigns. Each user can add multiple reviews, and it is designed so that each product can receive multiple reviews.

Flexibility for filtering reviews by the user or product enables better searching and sorting functions in the app. Future features may include updated user preferences which could alter review suggestions.

Normalisation helps minimise data redundancy and improves the datas’ integrity. User information is stored once, and reviews do not need to be duplicated. By separating User, Product, and Review models/entities, the implementation adheres to normalization principles, reducing the redundancy

During the project development phase, implementing User, Product and Review entities with their respective relationships establishes a robust database structure. This design aids in efficient data handling, and prepares for future enhancements, ensuring long term scalability and maintainability. The clearly defined interconnections in the entities, ensures development teams can focus on building features that will leverage the structured data efficiently.

Add in code examples

**Q8. *Explain how to use this application’s API endpoints. Each endpoint should be explained, including the following data for each endpoint:***

* **HTTP verb**
* **Path or route**
* **Any required body or header data**
* **Response**

**CREATE a Review**

HTTP Verb – POST

Path or Route: /api/reviews

Required body/header – Authorisation: bearer token for user authentication

Response –

Success: “Review was successfully created”

Error: “Invalid input data”, (201 Created)

Insert code snippets

**GET Reviews for a Product**

HTTP Verb – GET

Path/Route: /api/products/{productid}/reviews

Required body/header – The productID for which reviews are being retrieved

Response –

Success: “Your product review for {productID}”

Error: “Invalid, product not found”, 404

**UPDATE a Review**

HTTP Verb – PUT

Path/Route: /api/reviews/{reviewID}

Required body/header – the reviewID to update.

Response –

Success: “Your review has been updated”

Error: “Review not found”, (404 Not Found)

Error: “You are not authorised to update this review”, (403 Forbidden)

**DELETE a Review**

HTTP Verb – DELETE

Path/Route: /api/reviews/{reviewsID}

Required body/header – ID of the review to delete & Authorisation: Bearer token for user authentication.

Response -

Success: “Your review has been deleted”, (204 No Content)

Error: “Your review could not be found”, (404 Not Found)

Error: “You are not authorised to delete this review.”, (403 forbidden)

GET all Reviews by User

HTTP Verb – GET

Path/Route: /api/users/{userID}/reviews

Required body/header – userID of the owner of the review being retrieved

Response –

Success: “Your reviews”, (200 OK)

Error: “User not found”, (404 Not Found)

These API endpoints provide a comprehensive way to interact with the product review’s backend. It creates, retrieves, updates, and deletes user review while ensuring the data integrity and user authentication is maintained throughout the use of the app, the use of appropriate headers and error handling makes this a robust and scalable product review system.

Test examples of returned code testing outcomes