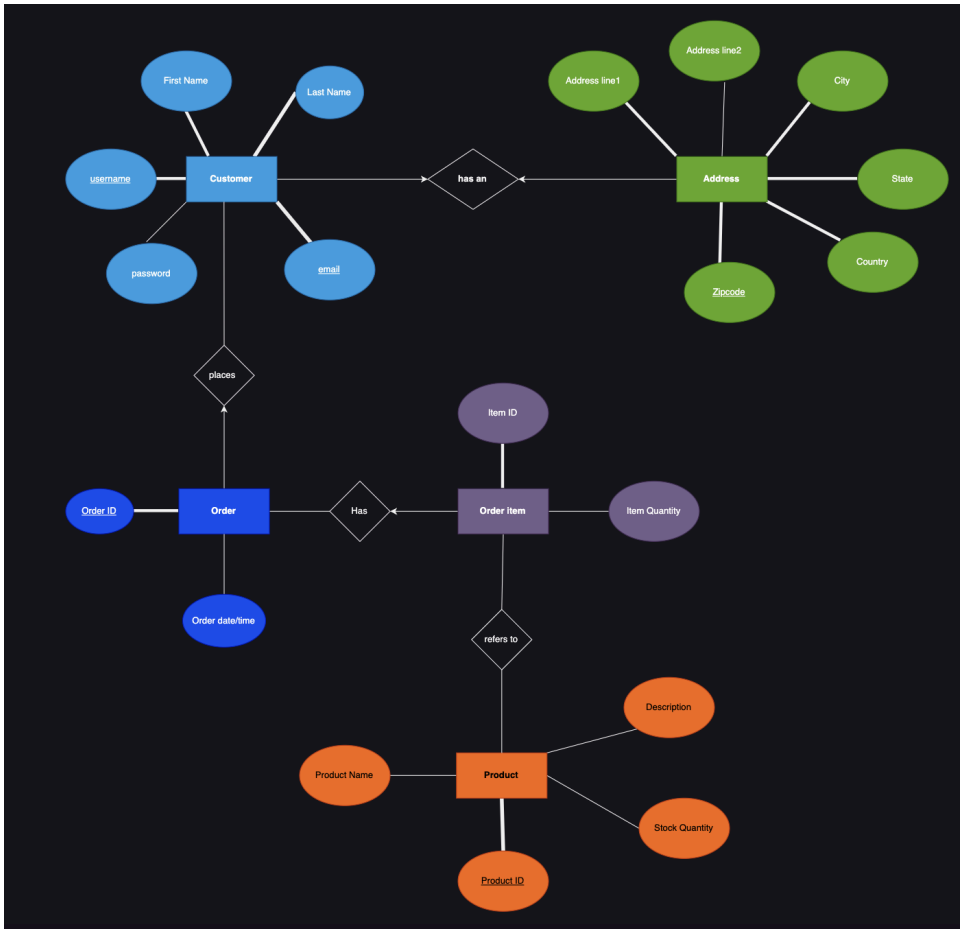
Online Retail Store Database Management System

Customers - username, password, email, firstname, last name
Products - product name, description, product id, quantity
Orders - order id, order date/time
Order Items - item qtty, item id
Address - address line1, address line2, city, state, country, zip code, phone number

Attached is an ER diagram of an **Online retail store database management system** which visualizes the relationship between entities like Customer, Address, Order, Order Item and Product. The initial design explains the online retail functions such as customer registration, product inventory management, and order tracking.

Initial state of ER diagram:



Customers -

username - TEXT, NOT NULL, UNIQUE
password - TEXT, NOT NULL
email - TEXT, NOT NULL, UNIQUE
firstname - TEXT, NOT NULL
last name - TEXT, NOT NULL

Products -

product name - TEXT, NOT NULL
description - TEXT
product id - INT NOT NULL UNIQUE
stock quantity - INT NOT NULL

Orders -

order id - SERIAL
order date/time - TIMESTAMPZ

Order Items -

item id - INT NOT NULL UNIQUE
item quantity - INT NOT NULL

Contact -

address line1 TEXT NOT NULL
address line2 TEXT
city TEXT NOT NULL
state TEXT NOT NULL
country TEXT NOT NULL
zip code TEXT NOT NULL
Phone number TEXT NOT NULL UNIQUE

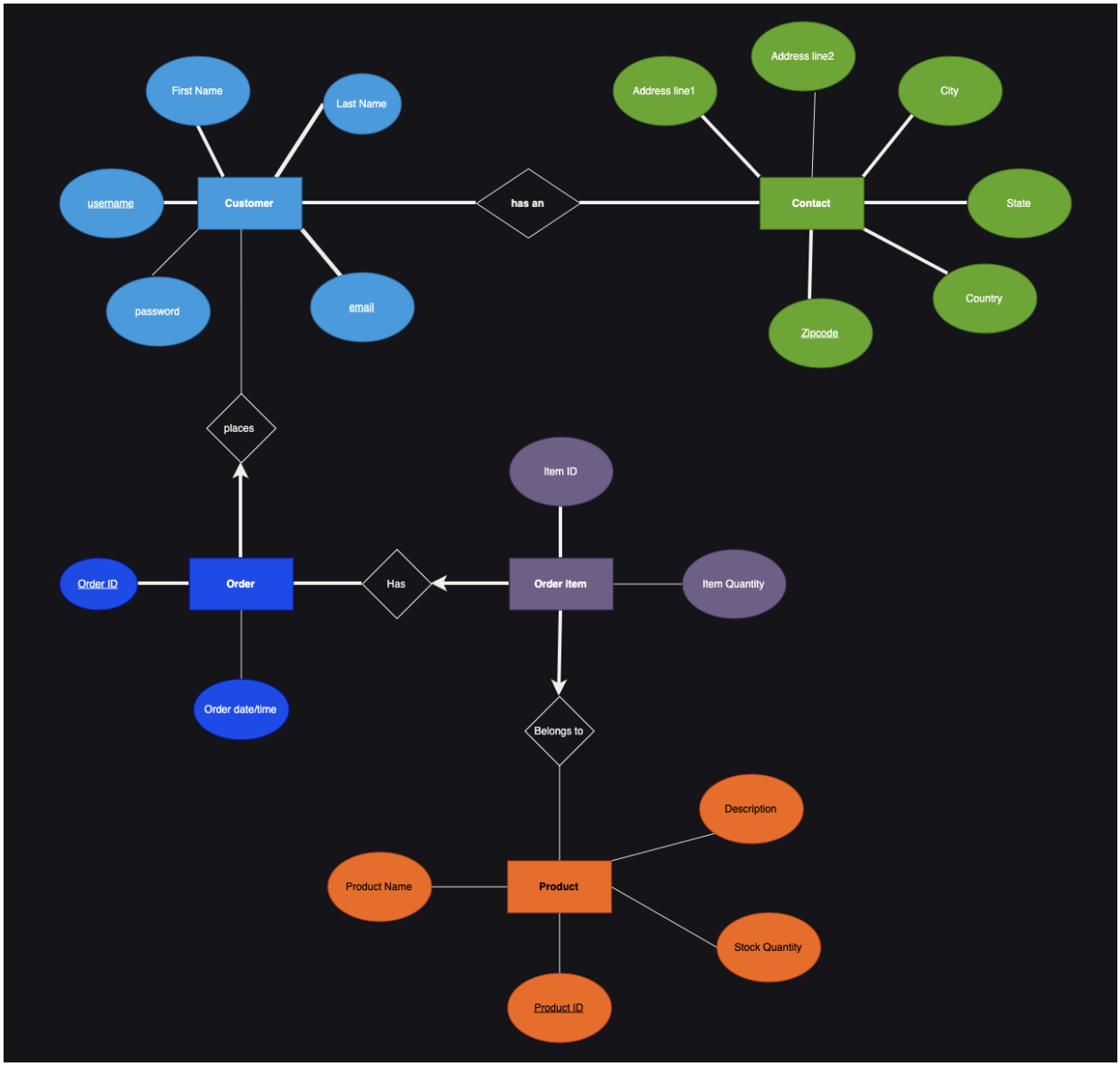
- Customer must have at least one Address.- **Bold line**
- Address must have atleast one customer - **Bold line**

- Customer has 0 or more orders - Line
- Each order belongs to one and only one customer

- Each order has one or more order items

- Each order items belongs to the order one and only once

- Each order item must belong to the product, one and only once
- Each product can be part of one or more order items



Relationship types

Relationship	**Type**	**Description**	
Customer - Address	Many to Many	• Customer must have at least one Address. • Address must have at least one customer	Customers_addresses as **bridge table.** Customer_id, address_id - **Foreign keys**
Customer-Order	One to Many	• Customer has 0 or more orders • Each order belongs to one and only one customer	Order has the **foreign key** - customer_id from customers
Order - Order Items	One to Many	• Each order has one or more order items • Each order items belongs to the order one and only once	OrderItems has the **foreign key** - order_id from Orders
Order Item - Product	One to Many	• Each order item must belong to the product, one and only once • Each product can be part of one or more order items	Order Items has the **foreign key** - product_id from Products

Question 1:

Did you choose to go with the ORM or raw SQL approach? What was the reason for your choice?

Question 2:

What endpoints (URLs and HTTP verb/methods) did you choose to implement in your Flask application, including any special details?

Question 3:

Brainstorm and describe some potential endpoints that you *could* implement in the future, that make sense for your application.

Question 4:

What challenges did you face with the Flask implementation for your portfolio project this week, and did you learn anything new from these challenges?

1.
I chose the ORM approach. In this approach, the object created maps to the entire table and using appropriate methods can bring in the required rows.ORM reduces the need for direct SQL queries, providing a more secure and maintainable way to interact with the database.
2.
GET/contacts/:id
POST/contacts
DELETE/contacts/:id
GET/orders/:id
POST/customers
3. Potential endpoints for the future would be endpoints for payment systems.
4. Challenges I faced are with the implementation of collections of endpoints for the web server using Flask Blueprints, and start the local Flask web server to serve those endpoints.

Week 1

Determined the entities and relationships that would be required.

Week 2

Translated the requirements into database tables with foreign key references using SQL in pgAdmin. Also inserted some data rows or formulated some SQL queries.

Week 3

Implemented a Python backend to interface with the database.

REST APIs were built on a Python backend using the Flask framework that exposes data to a client (e.g. Insomnia) via JSON responses.

Endpoints were created to perform the four CRUD operations (Create, Read, Update, and Delete) using different HTTP methods (GET, POST, PUT, PATCH, DELETE), enabling interaction with specific data utilizing the Insomnia Request Collection.

online_store/postgres@nucamp

Query Editor | Query History

```
1  SELECT
2      indexname,
3      indexdef
4  FROM
5      pg_indexes
6  WHERE
7      tablename = 'customers';
```

Data Output | Explain | Messages | Notifications

	indexname name	indexdef text
1	customers_pkey	CREATE UNIQUE INDEX customers_pkey ON public.customers USING btree (customer_id)
2	customers_username_key	CREATE UNIQUE INDEX customers_username_key ON public.customers USING btree (username)
3	customers_email_id_key	CREATE UNIQUE INDEX customers_email_id_key ON public.customers USING btree (email_id)
4	first_name_index	CREATE INDEX first_name_index ON public.customers USING btree (first_name)

online_store/postgres@nucamp

Query Editor | Query History

```
1  SELECT customer_id, username, password, email_id, first_name, last_name
2      FROM public.customers
3      WHERE first_name LIKE 'Je%';
```

Data Output | Explain | Messages | Notifications

	customer_id [PK] integer	username text	password text	email_id text	first_name text	last_name text
1	8	jeff52	653279f3b2398c4401d873d705c90a7401834681...	jeff@example.com	Jeff	Thomson
2	11	jeff523	fb4e3e33b6122b28fcf2d2e9a542928b60a02e808...	jeff@examplee.com	Jeff	Thomson

Hash index is implemented on the first_name column.

This project is an **Online Retail Store Database Management System** designed to optimize customer, product, order, and inventory management. It incorporates key entities such as customers, their contact addresses, products, orders, and order items, with clear relationships between them.To adhere to database normalization principles, **customers** and **contact addresses** are separated into distinct entities, reducing data redundancy and ensuring efficient data organization. The system uses **Flask** for the backend, with **SQLAlchemy ORM** for seamless database interactions, and **PostgreSQL** as the relational database, with **psycopg2** as the database adapter for Python. **Alembic** is utilized for database migrations, ensuring smooth schema changes over time. The project also employs **Insomnia** for testing RESTful APIs and **pgAdmin** for managing the PostgreSQL database. Key features include customer registration, order processing, and inventory tracking, with future enhancements possible through payment integrations, expanding shipping functionalities and advanced analytics.