CarParts.com

Project description:

This website is a car parts showcase platform that allows users to browse, view, and interact with detailed information about various car parts. Each part is displayed in a card view containing key details such as name, brand, and price. When a user clicks on a card, they are taken to a detailed page showing full information about the selected part, including images, specifications, and user feedback.

Tech Stack:

Frontend:

- HTML
- CSS
- Django Templates

Backend:

- Python
- Django

Database:

- PostgreSQL

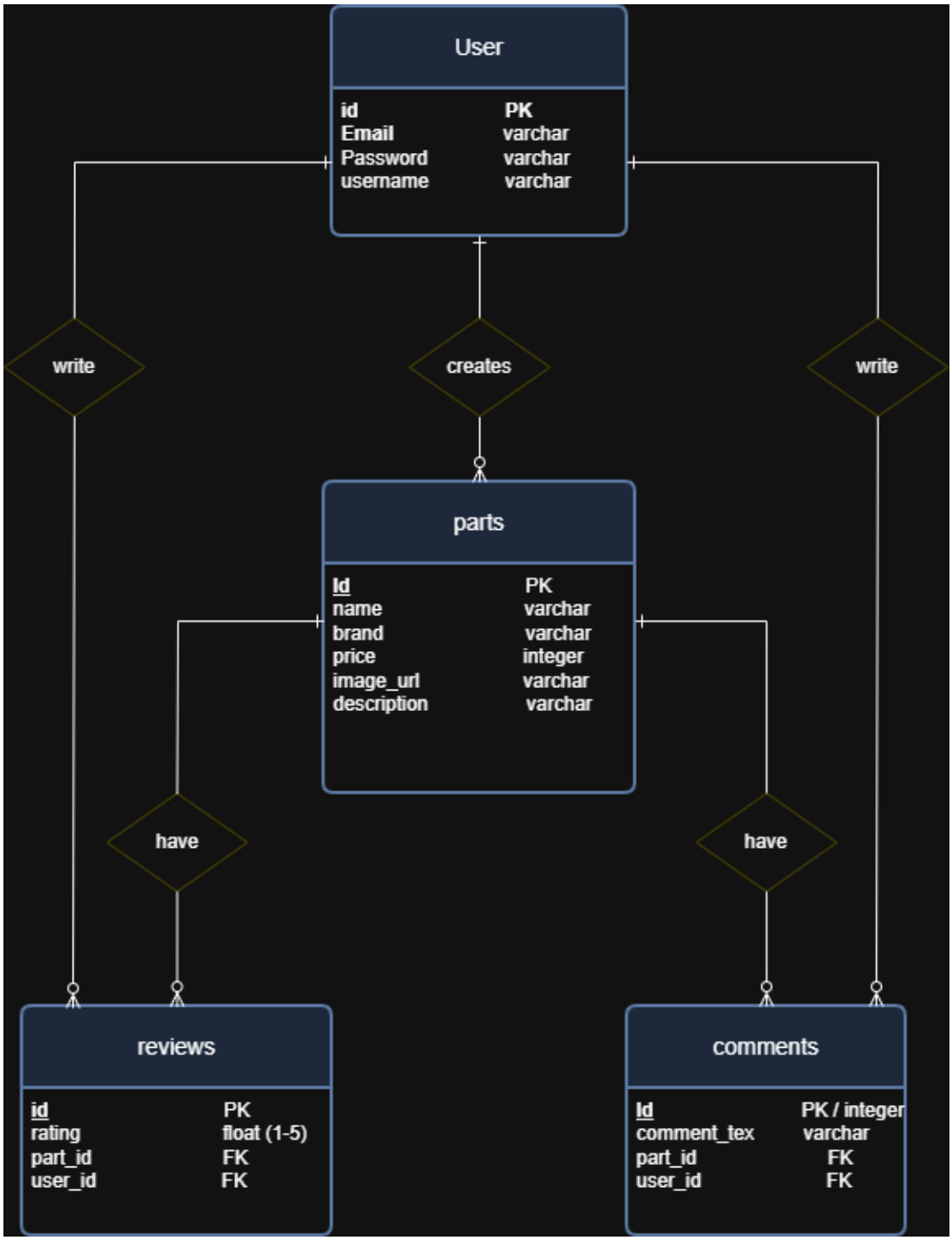
Other Tools:

- Django Authentication (built-in)
- Static Files Management (django.contrib.staticfiles)

User Stories:

- User can register and log in to access the website's features and interact with car parts
- User can browse all car parts in card view to explore available car parts easily
- User can click on a car part card to view complete details and specifications about that part
- User can submit a review (1–5 stars) for a car part to rate the part based on my experience
- User can add comments under a car part discuss and share thoughts with other users
- User can delete his own comments to manage his contributions to the site

ERD:


User
id PK
Email varchar
Password varchar
username varchar
write
creates
write
parts
Id PK
name varchar
brand varchar
price integer
image_url varchar
description varchar
have
have
reviews
id PK
rating float (1-5)
part_id FK
user_id FK
comments
Id PK / integer
comment_tex varchar
part_id FK
user_id FK

Installation and Setup Guide:

Follow these steps to set up and run the project locally:

1- Clone the Repository

git clone git@github.com:mohannad-sallam/Capstone-Project.git

cd carparts

2- Activate a Virtual Environment

python -m venv venv

venv\Scripts\activate

3- Install Dependencies

pip install -r requirements.txt

If you don’t have a requirements.txt, create one with:

pip freeze > requirements.txt

4- Set Up the Database

In your settings.py, update the database configuration:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'carparts',
        'USER': 'postgres',
        'PASSWORD': 'yourpassword',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Then run the migrations:

python manage.py makemigrations

python manage.py migrate

5- Create a Superuser (Admin)

python manage.py createsuperuser

6- Add Static Files and Images

Place your CSS and images inside:

main_app/static/css/

main_app/static/images/

7- Run the Development Server

python manage.py runserver

Visit your app at http://127.0.0.1:8000/

Summary of challenges encountered and solutions applied:

1- User Authentication and Access Control

- Challenge: Ensuring only logged-in users can access the home page and part details.
- Solution: Implemented Django's built-in authentication system and used the LoginRequiredMixin for class-based views.

2- Responsive Card Layout for Parts

- Challenge: When there was only one part, the card expanded too much and the image stretched.
- Solution: Used CSS Grid with grid-template-columns: repeat (auto-fit, minmax (280px, 1fr)) and a max-width on cards, keeping images properly scaled with object-fit: cover.

3- Logout Button in Navbar

- Challenge: Combining a logout form inside a navbar without breaking HTML semantics or styling.
- Solution: Created a standalone <form> for logout with a styled button and applied CSS for alignment with other navbar items, avoiding invalid HTML like <a><form>…</form>.

4- Database Schema Changes (Removing Fields)

- Challenge: Removing unnecessary fields like user_type from Profile while keeping the database consistent.
- Solution: Updated models.py, then ran makemigrations and migrate to reflect the changes in the PostgreSQL database safely.

5- Form Validation and UX

- Challenge: Users could log in with wrong credentials or see errors without feedback.
- Solution: Leveraged Django's LoginView and UserCreationForm along with proper messages to provide clear feedback on invalid logins or signup errors.