## Backend Wireframe and Models for Full Stack Web Development Project

### Part 1: Project Overview

**Project Title:** PCBuilder

**Project Description:** A website that will have users compare and choose which computer components to create their custom PC build. They will be able to sort each component by categories (price, brand, performance, ratings, etc..). Users will be given the ability to share their builds, save configurations, and make sure that each component is compatible with each other. They will be able to see price breakdowns of each component, as well as a final balance, and get the best price from a certain company.

### Part 2: API Endpoints

https://github.com/docyx/pc-part-dataset

https://github.com/AnthonyTsui/Buildapcscraper

https://github.com/nynhex/PCPartPicker-API

https://apify.com/matyascimbulka/pcpartpicker-scraper/api

This is a list of the primary API endpoints for our backend that we will be using:

| Endpoint URL | HTTP Method | Description | Request Parameters | Response Structure |
|---|---|---|---|---|
| /api/auth/register | POST | Register a new user | {username, email, password, first_name, last_name} | {success: true, user: {...}} |
| /api/auth/login | POST | Authenticate user and return token | {email, password} | {success: true, token: “jwt_token”, user: {...}} |
| /api/users/{id} | GET | Get current user profile | None (Auth token in header) | {id, username, email, first_name, last_name, …} |
| /api/users/{id} | PUT | Update user profile | {first_name, last_name, email} | {success: true} |

| | | | | |
|---|---|---|---|---|
| /api/users/{id} | DELETE | Delete a user | None | {success: true} |
| /api/components | GET | Get filtered components | Component_id, brand, prince_min, price_max, in_stock | [{id, name, price, specs..}] |
| /api/components/{id} | GET | Get component details | None | {id, name, price, description, specs, images, compatibility,...} |
| /api/builds | GET | Get user's saved builds | None (Auth token in header) | [{id, name, components, total_price,...}] |
| /api/builds/{id} | PUT | Update existing build | {name, components: [{component_id, quantity}], …} | {success: true, {...}} |
| /api/builds/compatibility | POST | Check compatibility | {components; [component_ids]} | {compatible: true/false, issues:[...]} |
| /api/price-history/{component_id} | GET | Get price history for a component | Path: component_id, query:days | {"component_id":id, "data":[{ date, price, retailer}]} |

## Part 3: Data Models

Model Name: User

- **Attributes:**
  - id: int, primary key, auto-increment
  - name: str, required
  - email: str, required, unique
  - Is_active: boolean, default to True
  - Is_admn: boolean, default to False
  - hashed_password: str, required
  - Last_login: datetime, nullable
  - Email_verified: boolean, default to False

    - created_at: datetime, default to current time
    - Verification_token: str, nullable, unique
- **Relationships:** User → build, review, resetToken, AuditLogs

Model Name: Components

**Attributes:**

    - id: int, primary key, auto-increment
    - name: str, required
    - category: str, required
    - brand: str, required
    - model: str, required
    - price: float, required
    - specs: JSON, required
    - Image_url: str, nullable
    - In_stock: boolean, default to True
    - Updated_at: datetime, default to current time
    - created_at: datetime, default to current time
- **Relationships:** Components are checked for compatibility with the compatibility model, price history, and reviews
- Components → Compatibility, price history, and reviews

Model Name: Compatibility Check

**Attributes:**

    - id: int, primary key, auto-increment
    - category1: int, foreign key -> Products model id
    - category2: int, foreign key -> Products model id
    - Rule_type: str, nullable
    - Rule_logic: JSON, nullable
- **Relationships:** The compatibility model checks if products from the product model are compatible
- When the user choses a component, the next component will only return parts that are compatible with the parts previously selected

Model Name: Build

**Attributes:**

    - id: int, primary key, auto-increment
    - User_id:int foreign key, required
    - name: str, required

    - Description: text, nullable
    - Is_public: boolean, default to True
    - Total_price: float, nullable
    - created_at: datetime, default to current time
    - Updated_at: DateTime, default to current time
- **Relationships:** Build → User, Build Component

Model Name: Build Component

**Attributes:**

    - id: int, primary key, auto-increment
    - Build_id: int, foreign key, required
    - Component_id: int, foreign key, required
    - Quantity: int, default to 0, required
- **Relationships:** Build  Component→  Build

Model Name: Price History

**Attributes:**

    - id: int, primary key, auto-increment
    - Component_id: int, foreign key, required
    - Price: float, required
    - Retailer: str, required
    - Url: str, nullable
    - Date: DateTime, default to current time
- **Relationships:** Price History→ Component

Model Name: Reviews

**Attributes:**

    - id: int, primary key, auto-increment
    - User_id:int foreign key, required
    - Component_id: int, foreign key, required
    - Rating: int, required
    - Title: str, nullable
    - Content: text, nullable
    - Verified: boolean, default to False
    - Status: str
    - created_at: datetime, default to current time
    - Updated_at: DateTime, default to current time
- **Relationships:** Reviews → User, Component

Model Name: Admin

**Attributes:**

- id: int, primary key, auto-increment
- Username: str, required, unique
- Permissions: JSON, nullable
- email: str, required, unique
- hashed_password: str, required
- created_at: DateTime, default to current time
- Last_login: DateTime, nullable
- Updated_at: DateTime, default to current time
- Is_active: boolean, default to True
- Role: str nullable

- **Relationships:** Admin → ResetTokens, Audit Logs, Admin Settings

Model Name: Admin Settings

**Attributes:**

- id: int, primary key, auto-increment
- Key: str, required, unique
- Value: JSON, required
- Description: str, nullable
- Updated_at: DateTime, default to current time
- Updated_by: int, foreign key, required

- **Relationships:** Admin Settings → Admin

Model Name: Reset Tokens

**Attributes:**

- id: int, primary key, auto-increment
- Token: str, required, unique
- User_id: int, foreign key, nullable
- Admin_id:  int, foreign key, nullable
- created_at: DateTime, default to current time
- Is_revoked: boolean, default to False
- Revoked_at: DateTime, nullable

- **Relationships:** Reset Tokens → Admin, User

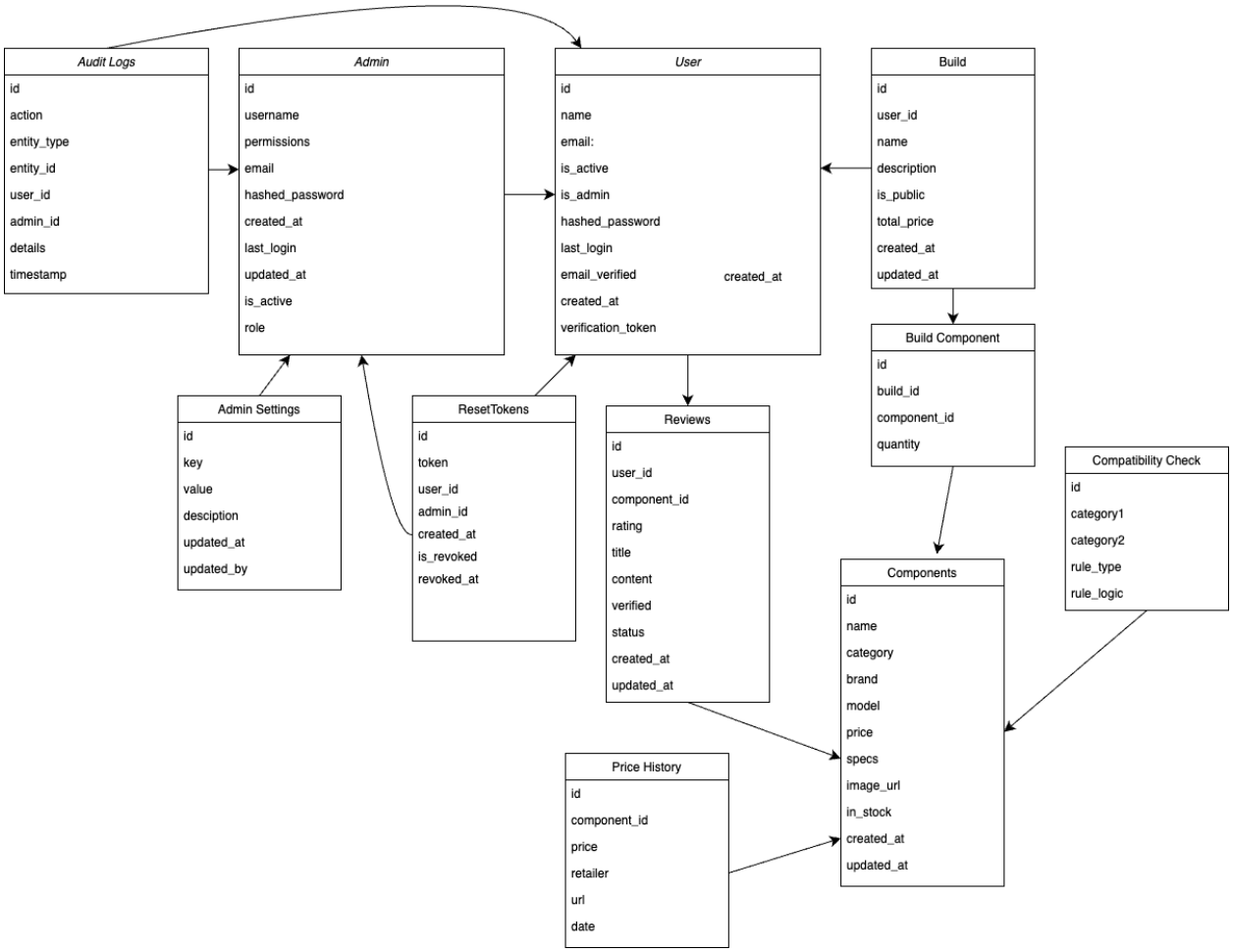
Model Name: Audit Logs

**Attributes:**

- id: int, primary key, auto-increment
- Action: str, required

    - Entity_type: str, required
    - Entity_id: int,  nullable
    - User_id: int, foreign key, nullable
    - Admin_id: int, foreign key, nullable
    - Details: JSON, nullable
    - Timestamp: datetime, default to current time
- **Relationships:** Audits Logs → User, Admin

## Part 4: Database Schema




## Part 5: Additional Considerations

## Authentication

- JWT (JSON Web Tokens) will be used for authentication
- Tokens will expire after 24 hours
- Password hashing will be implemented using bcrypt

- Refresh tokens will be implemented for seamless re-authentication

## Middleware

- CORS middleware to allow cross-origin requests from the frontend
- Authentication middleware to verify JWT tokens
- Rate limiting middleware to prevent abuse
- Request logging middleware for monitoring and debugging

## Error Handling

- Standard HTTP status codes will be used (200, 201, 400, 401, 403, 404, 500)
- Error responses will follow a consistent format:

```
{
  "success": false,
  "error": {
    "code": "ERROR_CODE",
    "message": "Appropriate Error message for user",
    "details": {} // Optional details
  }
}
```

- Detailed error logging for server-side issues
- Validation errors will include specific field information

## Testing

- Unit tests for models and utility functions using pytest
- Integration tests for API endpoints
- Automated testing using GitHub Actions CI/CD
- Manual testing using Postman collections
- Performance testing for high-traffic endpoints