Demo 2: Coding Standards Document

GreenCart

Client: BBD Software

Team member	Student Number
Nikhil Govender	u22504193
Shayden Naidoo	u23599236
Corné de Lange	u23788862
Tshegofatso Mahlase	u22580833
Samvit Prakash	u23525119

1. Coding Standards

1.1 Variable Naming

- **CamelCase** for variables and functions in JavaScript/TypeScript
  Example: `userId`, `addToCart`, `isLoggedIn`
- **snake_case** for Python
  Example: `user_id`, `add_to_cart()`
- **UPPER_SNAKE_CASE** for constants
  Example: `MAX_CART_ITEMS`

1.2 File Naming

- **Python/Backend:** `snake_case`
  - Example: `cart_service.py`
- **React/Frontend:**
  - **Components:** `PascalCase.jsx` (e.g. `ProductCard.jsx`)
  - **Files:** `kebab-case.js` (e.g. `user-service.js`)
- **CSS/Assets:** `kebab-case.css`, `image-name.png`

1.3 Functions & Classes

- **Functions:**
  - **Python:** `def fetch_user_data():`
  - **JS/TS:** `function fetchUserData()`

- **Classes:**
  PascalCase: `class ProductManager`

Type Casing Conventions

To maintain consistency across the GreenCart codebase, the following casing styles must be used:

Element Type	Naming Convention	Example
Variables	`snake_case`	`cart_total`, `user_id`
Functions / Methods	`snake_case`	`calculate_total()`, `get_user_by_id()`
Class Names	`PascalCase`	`CartItem`, `OrderManager`

Constants	`UPPER_SNAKE_CASE`	`MAX_RETRIES`, `API_VERSION`
Database Table Names	`snake_case`	`cart_items`, `sustainability_ratings`
File and Module Names	`snake_case`	`cart_routes.py`, `order_utils.py`
React Component Names (frontend)	`PascalCase`	`ProductCard`, `CheckoutForm`

2. Project Structure

```
GreenCart/
│
├── app/                    # FastAPI backend
│   ├── api/                # API routes
│   ├── models/             # SQLAlchemy/Pydantic models
│   ├── schemas/            # Data schemas
│   ├── services/           # Business logic
│   └── main.py             # FastAPI entry point
│
├── frontend/               # React frontend
│   ├── components/         # Reusable UI components
│   ├── pages/              # Route-based page components
│   ├── services/           # API functions
│   └── App.jsx             # Main React app
│
├── tests/                  # Backend tests using pytest
├── documents/              # Documentation and specs
├── greencart_dump.sql      # SQL schema and seed data
├── init_db.py              # Script to initialize DB
├── .github/workflows/      # CI/CD configuration
├── README.md
└── requirements.txt
```

3. Error Handling

3.1 Use of Try-Except Blocks

- Used around database operations, file I/O, and external API calls.
- Avoid wrapping large blocks—catch only relevant exceptions.

3.2 Logging Errors

- Backend uses Python’s logging module or a structured logger.
- Frontend should avoid exposing stack traces to users.
- Do not log sensitive data.

3.3 Error Propagation

- Re-throw with context where necessary.
- Use custom error classes if applicable.

3.4 User-Facing Error Handling

- Show simple, non-technical messages in frontend (e.g., “Something went wrong”).
- Implement retry logic for timeouts or 5xx errors where appropriate.

3.5 Error Codes

- Follow standard HTTP status codes:
  - 400 for bad requests
  - 401 for unauthorized
  - 404 for not found
  - 500 for server errors

3.6 Validations

- All form inputs and API payloads must be validated:

  - Check for nulls, types, and logical errors.
  - Use Pydantic schemas in FastAPI and Yup/Zod in React.

4. Testing & Debugging

4.1 Types of Tests

- **Unit Tests (Backend)**
    - Tool: `pytest`
    - Files live in `tests/`
    - Naming: `test_<module>.py`
- **Unit & Integration Tests (Frontend)**
    - Tool: `Jest` with React Testing Library
    - Naming: `<Component>.test.jsx`
- **End-to-End Testing (Planned)**
    - Tool: `Cypress`

4.2 Code Coverage

- Backend: `pytest-cov`, minimum 80% target
- Frontend: `Jest --coverage`, minimum 80% target

5. Git Repository & Strategy

5.1 Git Flow

We use the **Git Flow** strategy to manage parallel development safely.

Branch	Description
`main`	Stable production code
`dev`	Aggregation of completed features
`feature/*`	New features, always merge back to `dev`
`hotfix/*`	Emergency fixes, merge to both `main` and `dev`
`documentation`	Documentation updates
`config`	Linting, CI/CD, environment-related changes
`69-*` `71-*`	UI/API separation groups (not used in final prod structure)

5.2 Branch Naming Rules

- Use lowercase + hyphens

6. CI/CD

6.1 Linting

- **Backend**:
    - Tool: `flake8`
    - Config: `setup.cfg`
- **Frontend**:
    - Tool: `eslint`
    - Config: `.eslintrc.json`
- Linting is enforced via GitHub Actions on each push/PR.

6.2 Testing in CI

- **Backend**: Run all `pytest` tests
- **Frontend**: Run `Jest` tests with coverage
- Both sets of tests are automatically triggered on PR to `dev` or `main`

6.3 Future Deployment

- CI/CD deployment pipelines will be introduced using:
    - Supabase for managed PostgreSQL hosting
    - Vercel or similar for frontend deployments