Project Title: Plum Goodness Test Automation Suite

Phase: I

---

1. Overview

This project delivers a comprehensive end-to-end test automation suite for the Plum Goodness website (https://www.plumgoodness.com/), built using Playwright with TypeScript. The suite focuses on:

- UI validation across core pages (Home, Product Listing, Product Details, Cart, Login, Profile).
- Key user workflows: browsing, searching, filtering, adding to cart, order management.
- Regression testing to ensure platform stability across browsers (Chromium, Firefox, WebKit).

---

2. Prerequisites & Setup

1. **Node.js**: v16 or later
2. **npm**: Bundled with Node.js
3. **TypeScript**: Installed as a project dependency
4. **Browsers**: Chromium, Firefox, WebKit (auto-installed via Playwright)

To initialize:

```
npm install
npx playwright install
```

---

3. Project Structure

```
├── .github/workflows/        # CI/CD workflows (GitHub Actions)
├── fixtures/                 # Custom Playwright fixtures & hooks
│   └── plum.fixtures.ts      # Browser context, login-state fixtures
├── pages/                    # Page Object Model (POM) classes
│   ├── HomePage.ts
│   ├── ProductsPage.ts
│   ├── ProductDetailsPage.ts
│   ├── CartPage.ts
│   ├── LoginPage.ts
│   └── ProfilePage.ts
├── tests/                    # Test suites organized by page
│   ├── home.spec.ts
│   ├── products.spec.ts
│   ├── productDetails.spec.ts
│   ├── cart.spec.ts
│   ├── login.spec.ts
```

```
│   └── profile.spec.ts
├── utils/                    # Utility helpers
│   └── Screenshot.util.ts    # Full-page screenshot capture & attachment
├── screenshots/              # Captured screenshots & videos
├── playwright.config.ts      # Global Playwright configuration
└── package.json              # Project metadata & scripts
```

---

4. Page Object Model (POM)

Each UI page is encapsulated in its own class under `pages/`, e.g., `HomePage.ts`. Classes expose:

- **Selectors**: Centralized locators.
- **Actions**: Methods for user interactions (navigation, clicks, data entry).
- **Assertions**: Convenience methods to verify UI elements and states.

Benefits:

- High reusability.
- Clear separation between test logic and page internals.
- Easier maintenance when locators change.

---

5. Fixtures & Hooks

- **Custom fixtures** in `plum.fixtures.ts` manage browser contexts and authentication state.
- **Hooks** (`beforeAll`, `beforeEach`, `afterEach`) handle setup and cleanup:
  - Navigate to home.
  - Close pop-ups.
  - Capture and attach full-page screenshots after each test.

---

6. Test Scenarios

6.1 Home Page Tests (`home.spec.ts`)

- Verify home page loads and displays key elements (logo, banners).
- Navigate to Login and Products pages.
- Validate containers: Offers, Trending Products (Best of Plums), Spotlight, Routine Essentials, New Launches, Science-backed Solutions.
- Add items from each container to cart and verify cart quantity updates.

6.2 Products Page Tests (`products.spec.ts`)

- Search workflow with multiple terms.

- Page title and breadcrumb validation.
- Filter by tags, product type, concern, price.
- Sort orders (ascending/descending) and clear filters.
- Navigate back to Home via logo click.

6.3 Product Details Tests (`productDetails.spec.ts`)

- Verify details page opens and displays title, product name, price, size, "best suited for you" container.
- Add to cart and validate quantity.
- Reviews section:
    - Navigate to reviews.
    - Submit multiple review variations.
    - Like and dislike reviews.
- Pincode entry to view expected delivery dates.

6.4 Cart Page Tests (`cart.spec.ts`)

- Empty cart validation.
- Add single/multiple products to cart from listing and details pages.
- Remove individual items and clear all items.
- Page title verification.

6.5 Login Page Tests (`login.spec.ts`)

- Navigation from Home to Login.
- Page title verification.
- Successful login with valid credentials and OTP flow.
- Invalid login attempts with various numbers to validate error handling.

6.6 Profile Page Tests (`profile.spec.ts`)

- Access Profile (Order History) post-login.
- Logout flow validation.
- Combined login and add-to-cart scenario as end-to-end test.

---

7. Playwright Features Utilized

- **Isolated browser contexts** for clean state per test.
- **Auto-waiting** on elements.
- **Cross-browser testing**: Chromium, Firefox, WebKit.
- **Screenshots & Videos** for debugging failures.
- **Custom fixtures** to share state.
- **Parallel execution** and **test sharding** for speed.
- **Playwright Inspector** for interactive debugging.
- **Built-in assertions & locators** for robust checks.

---

8. Continuous Integration

- **GitHub Actions** automated pipeline defined in `.github/workflows/`:
  - Install dependencies.
  - Run `npx playwright test --reporter=html`.
  - Publish HTML reports as artifacts.
  - Fail build on any test failure.

---

9. Reporting & Artifacts

- **HTML Reports**: Detailed pass/fail summary, logs, screenshots, and videos.
- **Artifacts** in CI: Accessible from GitHub Actions for triage.

---

10. How to Run Locally

```
# Install dependencies
npm ci

# Execute full suite
npx playwright test

# Run a single file or test
npx playwright test tests/home.spec.ts

# Generate HTML report
npx playwright show-report
```

---

11. Maintenance & Next Steps

- Expand coverage to Checkout and Payment workflows.
- Integrate API-level tests for backend validation.
- Add accessibility checks (a11y).
- Enhance data-driven testing using external test data files.
- Schedule nightly regression runs and integrate Slack notifications.

End of Phase I Documentation.