# Test Strategy Document

**Project:** Full-Stack Todo Application (UI + API Automation)

---

## 1. Objective

To validate the end-to-end functionality of a simple Todo web application, focusing on UI (React frontend) and API (Node.js backend) test automation using Playwright and Postman.

---

## 2. Scope

**In-Scope:**

- UI Automation (Playwright)
- API Automation (Postman)
- Positive and Negative Scenarios
- Data-driven Testing using CSV

**Out of Scope:**

- Performance Testing
- Cross-browser Compatibility

---

## 3. Types of Testing

| Type | Tool | Purpose |
|---|---|---|
| UI Testing | Playwright | Simulate user behavior in browser |

|  |  |  |
|---|---|---|
| API Testing | Postman/Newman | Validate backend endpoints |
| Data-Driven | Postman + CSV | Run same test with varied input sets |

---

## 4. Test Environment Setup

- **Base URLs:**
    - `BASE_URL=http://localhost:3000` (Frontend)
    - `API_URL=http://localhost:8000` (Backend)
- **Env Variables:** Stored in `.env` file and Postman Environment
- **CSV Files:** For parameterization, placed in `/data` folder

---

## 5. Test Artifacts

**Playwright Folder Structure:**

- tests/
- ├── .env
- ├── constants.ts
- ├── fixtures.ts
- ├── login.spec.ts
- └── client.spec.ts

**Postman Collections:**

- `test.postman_collection_assignment.json`
- `Assignment variables.postman_environment.json`

- `Add_item_data.csv,delete_item_data.csv,edit_item_data.csv,login_data.csv`

---

## 6. Test Scenarios

**UI Tests (Playwright):**

- Signup - Valid/Invalid
- Login - Valid/Invalid
- Todo CRUD (Create, Read, Update, Delete)

**API Tests (Postman):**

- `POST /signup`: Verify user creation and error on duplicates
- `POST /login`: Status = 200, Token validation, Invalid login
- `GET /items`: Verify existing items
- `POST /items`: Add new item (valid/invalid)
- `PUT /items/:id`: Edit item text, status, ID match
- `DELETE /items/:id`: Validate success message, status, invalid ID

---

## 7. Assertions

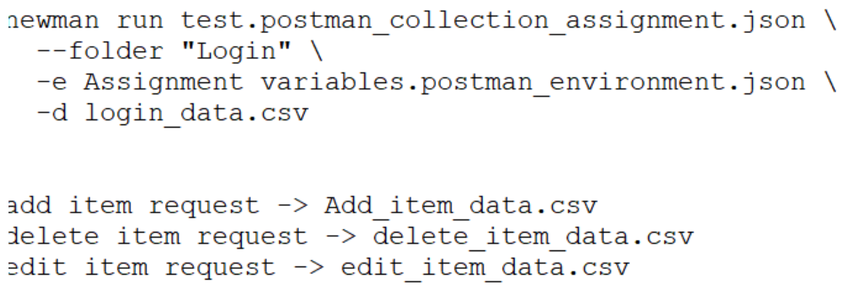
- **Status Codes:** 200, 400, 401 depending on test case
- **Field Validation:** Ensure correct fields in response (e.g., token, id, text)
- **Response Messages:** Match expected strings like "Item deleted successfully"
- **Data Persistence:** Added/edited item appears in `GET /items`

---

## 8. Execution

- **Playwright:** `npx playwright test client.spec.ts –headed –project=chromium`

- **Postman:** Logged in into Postman , imported the collection
  - The requests are in folder , so executed testcases by right clicking on folder then click on run and provide the script i.e the testdata and select the request and then click on run.
  - Below are the sample command and the mapping of request to the corresponding test data file



```
newman run test.postman_collection_assignment.json \
  --folder "Login" \
  -e Assignment variables.postman_environment.json \
  -d login_data.csv


add item request -> Add_item_data.csv
delete item request -> delete_item_data.csv
edit item request -> edit_item_data.csv
```

  - 
  - Only Create new user request is not having parameterize data and the positive and negative request is written separately.
  - Also for add , edit , delete and get item , we need a authtoken which we get on successful login so that is set as environment variable but if expired will have to login again,.

---

## 9. Reporting

- Playwright HTML report: `npx playwright show-report`

- Newman CLI summary or HTML reporters

---

## 10. Risks and Mitigation

| Risk | Mitigation |
|---|---|
| Hardcoded test data | Use environment variables and CSV files |
| Token expiry | Fetch token in pre-request or before suite |
| API changes | Add contract validation using schema-based testing |

---

## 11. Maintenance Plan

- Keep test data externalized
- Reuse selectors and helpers
- Group reusable logic into fixtures/helpers (Playwright)
- Use version control for test artifacts