# 🎭 Playwright UI Automation Notes (Beginner to Advanced)

---

## 🧩 1. What is Playwright?

- **Playwright** is an end-to-end browser automation framework developed by **Microsoft**.
- It's used for automating UI tests for web applications across **Chromium**, **Firefox**, and **WebKit**.
- **Features**:
    - Cross-browser support.
    - Headless and headful execution.
    - Auto-waiting for elements to be ready.
    - Parallel test execution.
    - Test retries, trace viewer, and video recording.
    - Support for modern locators (e.g., by role, label, text, etc.).
    - API testing and network mocking (not covered here).

---

## 🛠️ 2. Installation

### Step-by-Step:

```
npm init -y                    # Initialize a new Node.js project
npm install -D @playwright/test      # Install Playwright test runner
npx playwright install               # Install required browsers (Chromium, Firefox, WebKit)
```

---

## 🗂️ 3. Recommended Project Structure

```
playwright-project/
├── tests/                  # Folder containing all test cases
│   └── login.spec.ts       # Example test file
├── playwright.config.ts    # Global config file
├── package.json            # Project config
```

---

## ⚙️ 4. playwright.config.ts (Configuration File)

```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  testDir: './tests',
  timeout: 30 * 1000,
  retries: 1,
  use: {
    headless: true,
    screenshot: 'only-on-failure',
    video: 'retain-on-failure',
    baseURL: 'https://example.com',
  },
});
```

**Explanation:**

- `testDir`: Folder containing test files.
- `timeout`: Max timeout for each test.
- `retries`: Retry failed tests.
- `use`: Global browser settings.

---

## ✍️ 5. Writing Your First Test

```
import { test, expect } from '@playwright/test';

test('Verify homepage title and Docs link', async ({ page }) => {
  await page.goto('https://playwright.dev/');
  await expect(page).toHaveTitle(/Playwright/);
  await page.getByRole('link', { name: 'Docs' }).click();
  await expect(page).toHaveURL(/.*docs/);
```

});

---

## 🚀 6. Running Tests

```
npx playwright test                    # Run all tests
npx playwright test tests/login.spec.ts   # Run specific file
npx playwright test --debug            # Debug mode with UI
npx playwright test -g "test name"     # Run by title
```

---

## 🧠 7. Understanding Selectors in Playwright

| Selector Type | Usage Example | Description |
|---|---|---|
| CSS | `page.locator('.btn-primary')` | Standard CSS selector |
| Text | `page.getByText('Login')` | Matches visible text |
| Role | `page.getByRole('button', { name: 'Submit' })` | Accessibility role |
| Label | `page.getByLabel('Email')` | Associated label text |
| Placeholder | `page.getByPlaceholder('Search')` | Input placeholder |
| Test ID | `page.getByTestId('login-button')` | Custom data-testid attribute |
| XPath | `page.locator('//button[text()="Submit"]')` | XPath expression |

---

## 💡 8. Useful Page Actions

```
await page.goto('https://example.com');
await page.fill('#username', 'admin');
await page.fill('#password', '1234');
await page.click('button[type="submit"]');
await page.check('#accept');
await page.selectOption('#country', 'IN');
await page.press('#password', 'Enter');
await page.waitForSelector('.success-message');
```

---

## 🎯 9. Assertions with `expect()`

```
await expect(page).toHaveTitle(/Dashboard/);
await expect(page.locator('h1')).toHaveText('Welcome');
await expect(page.locator('#submit')).toBeDisabled();
await expect(page.locator('.alert')).toContainText('Success');
await expect(page.locator('#logout')).toBeVisible();
```

---

## 🖼️ 10. Screenshots and Video Capture

### Screenshots:

```
await page.screenshot({ path: 'home.png' });
```

### Configure globally:

```
use: {
  screenshot: 'only-on-failure',
  video: 'retain-on-failure',
}
```

---

## 🔁 11. Reusable Setup with Hooks

```
test.beforeEach(async ({ page }) => {
  await page.goto('https://example.com/login');
});
```

---

## 🧪 12. Grouping Tests with `test.describe()`

```
test.describe('Login Scenarios', () => {
  test('Valid login', async ({ page }) => {
    // logic
  });

  test('Invalid login', async ({ page }) => {
    // logic
  });
});
```

---

## 🏷️ 13. Tagging and Filtering Tests

```
test('@smoke login test', async ({ page }) => {
  // steps
});
```

Run with tag:

npx playwright test --grep @smoke

---

## 🧠 14. Auto-Waiting Features

Playwright automatically waits for:

- Element to be attached to DOM
- Element to be visible
- Element to be stable (no animation)

No need for manual `waitForTimeout`.

---

## 🪪 15. Saving Login Session

### Create login session:

```
await page.context().storageState({ path: 'auth.json' });
```

### Use in config:

```
use: {
  storageState: 'auth.json'
}
```

Avoids logging in before every test.

---

## 🔄 16. Parallel Execution

Playwright supports parallelism by default.

Control with:

npx playwright test --workers=4

---

## 🖱️ 17. Keyboard and Mouse

```
await page.keyboard.press('Enter');
await page.mouse.move(100, 150);
await page.mouse.click(120, 200);
```

---

## 📏 18. Mobile & Responsive Testing

```
import { devices } from '@playwright/test';

use: devices['iPhone 13'];
```

---

## 🔐 19. Handling Alerts/Popups

```
page.on('dialog', async dialog => {
  console.log(dialog.message());
  await dialog.accept();
});
```

---

## 🕵️ 20. Codegen Tool

npx playwright codegen https://example.com

Generates test code as you interact with the browser.

---

## 📊 21. HTML Report

npx playwright show-report

After test execution, an HTML report opens with result details.

---

## ✅ 22. Best Practices

- Use `getByRole()` for accessibility.
- Avoid fixed waits like `waitForTimeout`.
- Use `beforeEach` for repeated setup.
- Group related tests using `describe`.
- Keep tests atomic and stateless.
- Reuse authentication with `storageState`.
- Tag tests logically (`@regression`, `@smoke`, `@login`).
- Use clear assertion messages.

---