SECTIONS 1-3: Enterprise Framework, Branching & Daily Workflow

SECTION 1 — Enterprise Framework Overview

1.1 What is an Enterprise Automation Framework?

An **enterprise automation framework** is a structured system where multiple employees work together to automate software testing across different application modules.

```
┌──────────────────────────────────────────────────────────────────┐
│                      ENTERPRISE FRAMEWORK                          │
├──────────────────────────────────────────────────────────────────┤
│                                                                    │
│                                                                  │
│   ┌─────────────┐    ┌─────────────┐    ┌─────────────┐          │
│   │  MODULE A   │    │  MODULE B   │    │  MODULE C   │          │
│   │   (Login)   │    │  (Payment)  │    │  (Reports)  │          │
│   │             │    │             │    │             │          │
│   │  Team A     │    │  Team B     │    │  Team C     │          │
│   └─────────────┘    └─────────────┘    └─────────────┘          │
│          │                 │                  │                    │
│          └─────────────────┼──────────────────┘                    │
│                            │                                       │
│                   ┌────────▼────────┐                              │
│                   │  SHARED REPO    │                              │
│                   │  (GitHub)       │                              │
│                   └─────────────────┘                              │
└──────────────────────────────────────────────────────────────────┘
```

1.2 How Multiple Employees Work on Different Modules

In an enterprise, the application is divided into **modules**:

Module	Example	Team Assigned
Authentication	Login, Logout, Password Reset	Team A
User Management	Registration, Profile, Settings	Team B
Payments	Checkout, Billing, Refunds	Team C
Reports	Dashboard, Analytics, Export	Team D

How It Works:

1. Each team owns specific modules
2. All teams share the same code repository
3. Each team creates their tests in separate folders
4. Everyone follows the same coding standards
5. CI/CD runs all tests together

1.3 Key Roles and Responsibilities

1.3.1 Automation Engineer (Junior/Mid Level)

Primary Responsibilities:

- Write automated test cases
- Maintain existing tests
- Execute tests locally
- Report bugs found during automation
- Follow coding standards

Daily Activities:

```
Morning:
  └── Pull latest code from repository
  └── Check if any assigned tests need updates

During Day:
  └── Write new test cases
  └── Fix failing tests
  └── Run tests locally before pushing

End of Day:
  └── Push code to feature branch
  └── Raise Pull Request if work is complete
```

1.3.2 Senior Automation Engineer

Primary Responsibilities:

- Design framework structure
- Review code from junior engineers
- Handle complex automation challenges
- Mentor team members
- Optimize test execution

Additional Duties:

- Create reusable utilities
- Define page objects structure
- Establish coding standards
- Troubleshoot flaky tests

1.3.3 Automation Lead

Primary Responsibilities:

- Plan automation strategy
- Assign work to team members
- Track progress and metrics
- Coordinate with other teams
- Make architectural decisions

Key Metrics Tracked:

Metric	Description
Test Coverage	% of features automated
Pass Rate	% of tests passing
Execution Time	Total time to run all tests
Flaky Rate	% of unstable tests

1.3.4 CI/CD Owner (DevOps Engineer)

Primary Responsibilities:

- Set up and maintain CI/CD pipelines
- Configure GitHub Actions/Jenkins
- Manage cloud runners
- Handle secrets and credentials
- Monitor pipeline health

Key Tasks:

- Configure parallel execution
- Set up test environments
- Manage artifact storage
- Optimize pipeline speed

1.4 Team Collaboration Model

```
┌──────────────────────────────────────────────────────────────┐
│                     COLLABORATION FLOW                         │
├──────────────────────────────────────────────────────────────┤
│                                                                │
│  AUTOMATION ENGINEER                                           │
│         │                                                      │
│         │ 1. Writes tests                                      │
│         │ 2. Raises PR                                         │
│         ▼                                                      │
│  SENIOR ENGINEER (Reviewer)                                    │
│         │                                                      │
│         │ 3. Reviews code                                      │
│         │ 4. Approves/Requests changes                         │
│         ▼                                                      │
│  AUTOMATION LEAD                                               │
│         │                                                      │
│         │ 5. Final review (if needed)                          │
│         │ 6. Merges to develop                                 │
│         ▼                                                      │
│  CI/CD PIPELINE (Automatic)                                    │
│         │                                                      │
│         │ 7. Runs all tests                                    │
│         │ 8. Generates reports                                 │
│         ▼                                                      │
│  TEAM NOTIFICATION                                             │
│         │                                                      │
│         └── 9. Success/Failure alerts sent                     │
│                                                                │
└──────────────────────────────────────────────────────────────┘
```

SECTION 2 — Repository & Branching Strategy

2.1 What is Git?

Git is a version control system that tracks changes to your code over time.

Simple Analogy: Think of Git like "Track Changes" in Microsoft Word, but much more powerful for code.

Why Git is Essential:

- Track all changes made to code
- See who made what change and when
- Revert to previous versions if needed
- Work on multiple features simultaneously
- Collaborate without overwriting others' work

2.2 What is GitHub?

GitHub is a cloud platform that stores your Git repositories online.

Simple Analogy: If Git is like a photo album, GitHub is like Google Photos where you store and share your albums.

Why Enterprises Use GitHub:

Reason	Explanation
Central Storage	All code in one accessible place
Backup	Code is safe even if laptop crashes
Collaboration	Multiple people can work together
Code Review	Pull Requests for quality control
CI/CD	Built-in GitHub Actions
Access Control	Control who can see/edit code
Audit Trail	Complete history of all changes

2.3 Repository Structure

```
playwright-automation-framework/          ← Root Directory
│
├── .github/                              ← GitHub Configuration
│   └── workflows/
│       └── playwright.yml                ← CI/CD Pipeline
│
├── pages/                                ← Page Object Models
│   ├── LoginPage.ts
│   ├── RegisterPage.ts
│   └── DashboardPage.ts
│
├── tests/                                ← Test Files
│   ├── login/
│   │   └── login.spec.ts
│   ├── register/
│   │   └── register.spec.ts
│   └── dashboard/
│       └── dashboard.spec.ts
│
├── test-data/                            ← Test Data
│   ├── loginData.ts
│   └── registerData.ts
│
├── utils/                                ← Utilities
│   ├── helpers.ts
│   └── excelReader.ts
│
├── playwright.config.ts                  ← Playwright Config
├── package.json                          ← Dependencies
├── tsconfig.json                         ← TypeScript Config
└── README.md                             ← Documentation
```

2.4 Branch Types Explained

2.4.1 Main/Master Branch

```
PURPOSE: Production-ready code only
```

Rules:

- ❌ Never commit directly
- ❌ Never push untested code
- ✅ Only merge from develop after full testing
- ✅ Always stable and deployable

Protection: This branch should be protected - no one can push directly.

2.4.2 Develop Branch

```
PURPOSE: Integration branch for all features
```

Rules:

- All feature branches merge here first
- Must pass all tests before merging to main
- Represents "next release" code

2.4.3 Feature Branches (feature/*)

```
PURPOSE: New test cases or features
```

Examples:

- `feature/login-tests`
- `feature/payment-validation`
- `feature/add-excel-reader`

Lifecycle:

```
1. Created FROM: develop
2. Merged INTO: develop
3. Deleted AFTER: merge is complete
```

2.4.4 Bugfix Branches (bugfix/*)

```
PURPOSE: Fix failing tests or broken code
```

Examples:

- `bugfix/login-timeout-issue`

- `bugfix/locator-update-dashboard`

When to Use:

- Test started failing after recent changes
- Locator changed due to UI update
- Logic error found in existing test

2.4.5 Hotfix Branches (hotfix/*)

```
PURPOSE: Emergency fixes for production
```

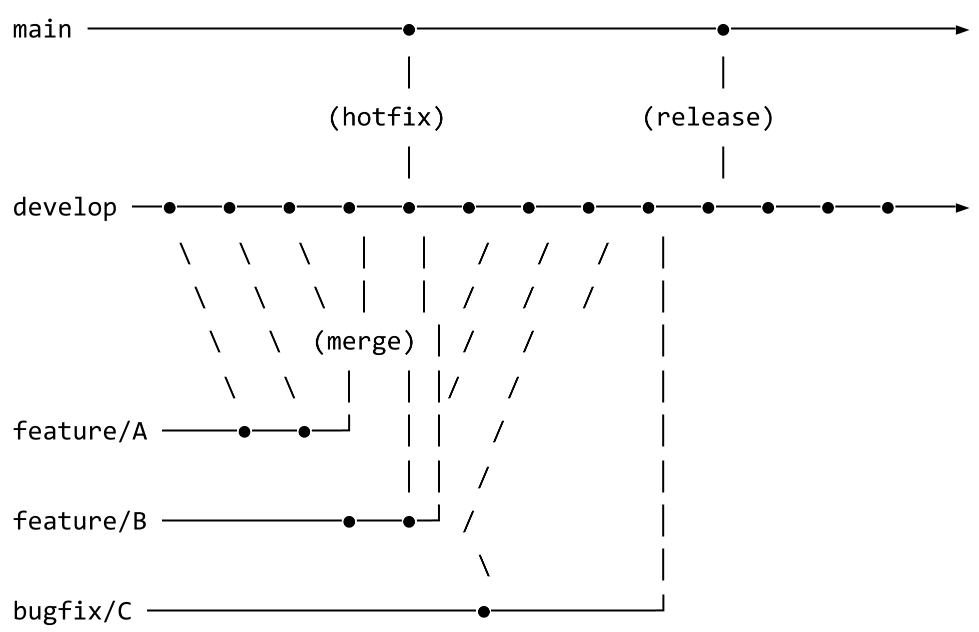
Examples:

- `hotfix/critical-login-failure`
- `hotfix/ci-pipeline-broken`

Special Rules:

- Created from: main (not develop)
- Merged into: BOTH main AND develop
- Used only for urgent issues

2.5 Branch Flow Diagram



```
LEGEND:
● = commit
─ = time progression
/ = merge
```

2.6 Branch Naming Conventions

Type	Format	Example
Feature	`feature/<description>`	`feature/login-page-tests`
Bugfix	`bugfix/<issue>`	`bugfix/fix-flaky-checkout`
Hotfix	`hotfix/<critical-issue>`	`hotfix/ci-broken`
Release	`release/<version>`	`release/v1.2.0`

Naming Rules:

1. Use lowercase letters only
2. Use hyphens (-) not underscores
3. Keep it short but descriptive
4. Include ticket number if available (e.g., `feature/JIRA-123-login-tests`)

2.7 When to Create Which Branch?

Scenario	Branch Type	Example
Writing new tests for a feature	`feature/*`	`feature/checkout-tests`
Fixing a broken test	`bugfix/*`	`bugfix/login-locator-fix`
Urgent fix needed in production	`hotfix/*`	`hotfix/critical-failure`
Preparing a new release	`release/*`	`release/v2.0.0`

SECTION 3 — Daily Tester Workflow (Step-by-Step)

3.1 Complete Workflow Overview

```
┌─────────────────────────────────────────────────────────────────┐
│                     DAILY TESTER WORKFLOW                        │
├─────────────────────────────────────────────────────────────────┤
│                                                                  │
│   START                                                          │
│     │                                                            │
│     ▼                                                            │
│  ┌────────────────┐                                              │
│  │ 1. Clone Repo  │  ◄─── First time only                        │
│  └───────┬────────┘                                              │
│          │                                                       │
│          ▼                                                       │
│  ┌────────────────┐                                              │
│  │ 2. Pull Latest │  ◄─── Every day                              │
│  └───────┬────────┘                                              │
│          │                                                       │
│          ▼                                                       │
│  ┌────────────────────┐                                          │
│  │ 3. Create Branch   │                                          │
│  └───────┬────────────┘                                          │
│          │                                                       │
│          ▼                                                       │
│  ┌────────────────────┐                                          │
│  │ 4. Write/Modify    │                                          │
│  │    Test Cases      │                                          │
│  └───────┬────────────┘                                          │
│          │                                                       │
│          ▼                                                       │
│  ┌────────────────────┐                                          │
│  │ 5. Run Tests       │                                          │
│  │    Locally         │                                          │
│  └───────┬────────────┘                                          │
│          │                                                       │
│      ┌───┴───┐                                                   │
│      │ Pass? │                                                   │
│      └───┬───┘                                                   │
│     No   │   Yes                                                 │
│    ◄─────┴─────►                                                 │
│    │           │                                       │
│    │           ▼                                       │
│    │    ┌────────────────┐                             │
│    │    │ 6. Stage       │                             │
│    │    │    Changes     │                             │
│    │    └───────┬────────┘                             │
│    │            │                                      │
│    │            ▼                                      │
│ Fix │    ┌────────────────┐                            │
│    │    │ 7. Commit      │                             │
│    │    └───────┬────────┘                             │
│    │            │                                      │
│    │            ▼                                      │
│    │    ┌────────────────┐                             │
│    │    │ 8. Push Code   │                             │
│    │    └───────┬────────┘                             │
│    │            │                                      │
│    │            ▼                                      │
│    │    ┌────────────────┐                             │
│    │    │ 9. Raise PR    │                             │
│    │    └───────┬────────┘                             │
│    │            │                                      │
│    │            ▼                                      │
│    │    ┌────────────────────┐                         │
│    └───►│ 10. Code Review    │                         │
│         └───────┬────────────┘                         │
│                 │                                      │
│                 ▼                                      │
│         ┌────────────────┐                             │
│         │ 11. Merge      │                             │
│         └───────┬────────┘                             │
│                 │                                      │
│                 ▼                                      │
│         ┌────────────────────┐                         │
│         │ 12. Delete Branch  │                         │
│         └────────────────────┘                         │
│                                                        │
│  END                                                   │
└──────────────────────────────────────────────────────┘
```

3.2 Step 1: Clone Repository (First Time Only)

What is Cloning?

Cloning creates a copy of the GitHub repository on your local computer.

Command:

```
git clone https://github.com/company/automation-framework.git
```

What Happens Internally:

1. Git contacts GitHub server
2. Downloads all files and history
3. Creates a folder with the repository name
4. Sets up connection to GitHub (remote)

After Cloning:

```
cd automation-framework    # Enter the folder
npm install                 # Install dependencies
```

3.3 Step 2: Pull Latest Code (Every Day)

What is Pulling?

Getting the latest changes from GitHub that others have pushed.

Command:

```
# Make sure you're on develop branch
git checkout develop

# Pull latest changes
git pull origin develop
```

What Happens Internally:

```
GitHub (Remote)                   Your Computer (Local)
┌──────────────────┐              ┌──────────────────┐
│ Latest Code      │              │ Your Code        │
│ (10 new commits) │  ──────→     │ (gets updated)   │
└──────────────────┘   git pull   └──────────────────┘
```

Common Mistake:
❌ Forgetting to pull before starting work = working on outdated code

3.4 Step 3: Create Feature Branch

Command:

```
# Create and switch to new branch
git checkout -b feature/login-tests
```

What Happens Internally:

1. Git creates a new branch pointer
2. Copies current develop state
3. Switches your working directory to new branch

Verify Branch:

```
git branch    # Shows all branches, * marks current
```

Output:

```
  develop
* feature/login-tests    ← You are here
  main
```

3.5 Step 4: Add or Modify Test Cases

Example: Creating a new test file

```
// tests/login/login-validation.spec.ts
import { test, expect } from '@playwright/test';
import { LoginPage } from '../../pages/LoginPage';

test.describe('Login Validation Tests', () => {
    test('should show error for empty username', async ({ page }) => {
        const loginPage = new LoginPage(page);
        await loginPage.navigate();
        await loginPage.fillLoginForm('', 'password123');
        await loginPage.submitForm();

        await expect(loginPage.errorMessage).toBeVisible();
    });
});
```

3.6 Step 5: Run Tests Locally

Why Run Locally First?

- Catch errors before pushing
- Faster feedback loop
- Don't break the pipeline for others

Commands:

```
# Run all tests
npx playwright test

# Run specific test file
npx playwright test tests/login/login-validation.spec.ts

# Run with browser visible (headed mode)
npx playwright test --headed

# Run with debug mode
npx playwright test --debug
```

Expected Output:

```
Running 5 tests using 4 workers

  ✓ login-validation.spec.ts:5:5 › should show error for empty username (2s)
  ✓ login-validation.spec.ts:15:5 › should show error for empty password (1s)
  ✓ login-validation.spec.ts:25:5 › should login with valid credentials (3s)
  ✓ login-validation.spec.ts:35:5 › should show error for invalid credentials (2s)
  ✓ login-validation.spec.ts:45:5 › should redirect after successful login (2s)

  5 passed (10s)
```

3.7 Step 6: Stage Changes

What is Staging?

Marking specific files to be included in the next commit.

Commands:

```
# See what files changed
git status

# Stage specific file
git add tests/login/login-validation.spec.ts

# Stage all changed files
git add .

# Stage specific folder
git add tests/
```

What Happens Internally:

```
Working Directory        Staging Area           Repository
┌───────────────────┐    ┌───────────────────┐   ┌───────────────────┐
│ Modified files    │    │ Files ready       │   │ Committed         │
│                   │ ─→ │ for commit        │──→│ history           │
│ - file1.ts        │    │                   │   │                   │
│ - file2.ts        │    │ - file1.ts        │   │ (permanent)       │
└───────────────────┘    └───────────────────┘   └───────────────────┘
       git add                git commit
```

3.8 Step 7: Commit Changes

What is a Commit?

A snapshot of your staged changes with a message describing what you did.

Command:

```
git commit -m "Add login validation test cases"
```

Commit Message Standards (Enterprise):

Type	Description	Example
`feat:`	New feature/test	`feat: add login validation tests`
`fix:`	Bug fix	`fix: update login button locator`
`refactor:`	Code improvement	`refactor: simplify page object methods`
`docs:`	Documentation	`docs: update README with setup steps`
`chore:`	Maintenance	`chore: update dependencies`

Good vs Bad Commit Messages:

❌ Bad	✅ Good
`update`	`feat: add checkout page tests`
`fix bug`	`fix: resolve timeout in payment test`
`changes`	`refactor: extract common login helper`
`test`	`feat: add data-driven login tests`

3.9 Step 8: Push Code

What is Pushing?

Sending your local commits to GitHub.

Command:

```
git push origin feature/login-tests
```

What Happens Internally:

```
Your Computer (Local)              GitHub (Remote)
┌───────────────────┐              ┌───────────────────┐
│ Your commits      │              │ Repository        │
│                   │   ───────►   │ (updated)         │
│ 3 new commits     │   git push   │                   │
└───────────────────┘              └───────────────────┘
```

First Time Push (new branch):

```
git push -u origin feature/login-tests
```

The `-u` sets upstream tracking, so future pushes only need `git push`.

3.10 Step 9: Raise Pull Request (PR)

What is a Pull Request?
A request to merge your branch into another branch (usually develop).

Steps to Create PR:

1. Go to GitHub repository
2. Click "Pull Requests" tab
3. Click "New Pull Request"
4. Select: base = `develop`, compare = `feature/login-tests`
5. Fill in title and description
6. Click "Create Pull Request"

PR Title Format:

```
[JIRA-123] feat: Add login validation test cases
```

PR Description Template:

```
## Summary
Added 5 new test cases for login validation.

## Changes
- Added login-validation.spec.ts
- Added new test data for edge cases
- Updated LoginPage with error message locator

## Testing
- All tests pass locally
- Tested on Chrome and Firefox

## Checklist
- [ ] Tests pass locally
- [ ] Code follows standards
- [ ] No hardcoded values
- [ ] Added appropriate test tags
```

3.11 Step 10: Code Review Process

What Happens:

1. PR is created
2. Reviewers are notified
3. Reviewers check the code
4. Comments/suggestions are added
5. Author makes changes if needed
6. Reviewers approve

Responding to Review Comments:

```
# Make requested changes in your code
# Then commit and push

git add .
git commit -m "fix: address review comments"
git push origin feature/login-tests
```

Common Review Feedback:

Feedback	Action
"Add assertion for URL"	Add `expect(page).toHaveURL(...)`
"Remove hardcoded wait"	Replace `waitForTimeout` with proper wait
"Use test data file"	Move data to test-data folder
"Add test tags"	Add `@smoke` or `@regression` tags

3.12 Step 11: Merge Strategy

After Approval:

1. Click "Merge Pull Request"
2. Choose merge type (usually "Squash and merge" for clean history)
3. Confirm merge

Merge Types:

Type	When to Use
Merge commit	Preserve all commit history
Squash and merge	Combine all commits into one (recommended)
Rebase and merge	Linear history, no merge commits

3.13 Step 12: Branch Cleanup

After Merge:

```
# Switch back to develop
git checkout develop

# Pull the merged changes
git pull origin develop

# Delete local feature branch
git branch -d feature/login-tests

# Delete remote branch (if not auto-deleted)
git push origin --delete feature/login-tests
```

3.14 Common Mistakes and Solutions

Mistake	Impact	Solution
Not pulling before starting	Merge conflicts later	Always `git pull` first
Committing to wrong branch	Changes in wrong place	Use `git stash` and switch
Forgetting to run tests	Broken pipeline	Make it a habit before push
Vague commit messages	Hard to track changes	Follow commit message standards
Large PRs with many changes	Hard to review	Keep PRs small and focused
Not responding to reviews	PR gets stale	Respond within 24 hours

Continue to Section 4-6...