← AI in Test Automation

Framework Design & Capstone Project

Table of Contents

Automation Framework Architecture

Framework Design Principles

SOLID Principles:

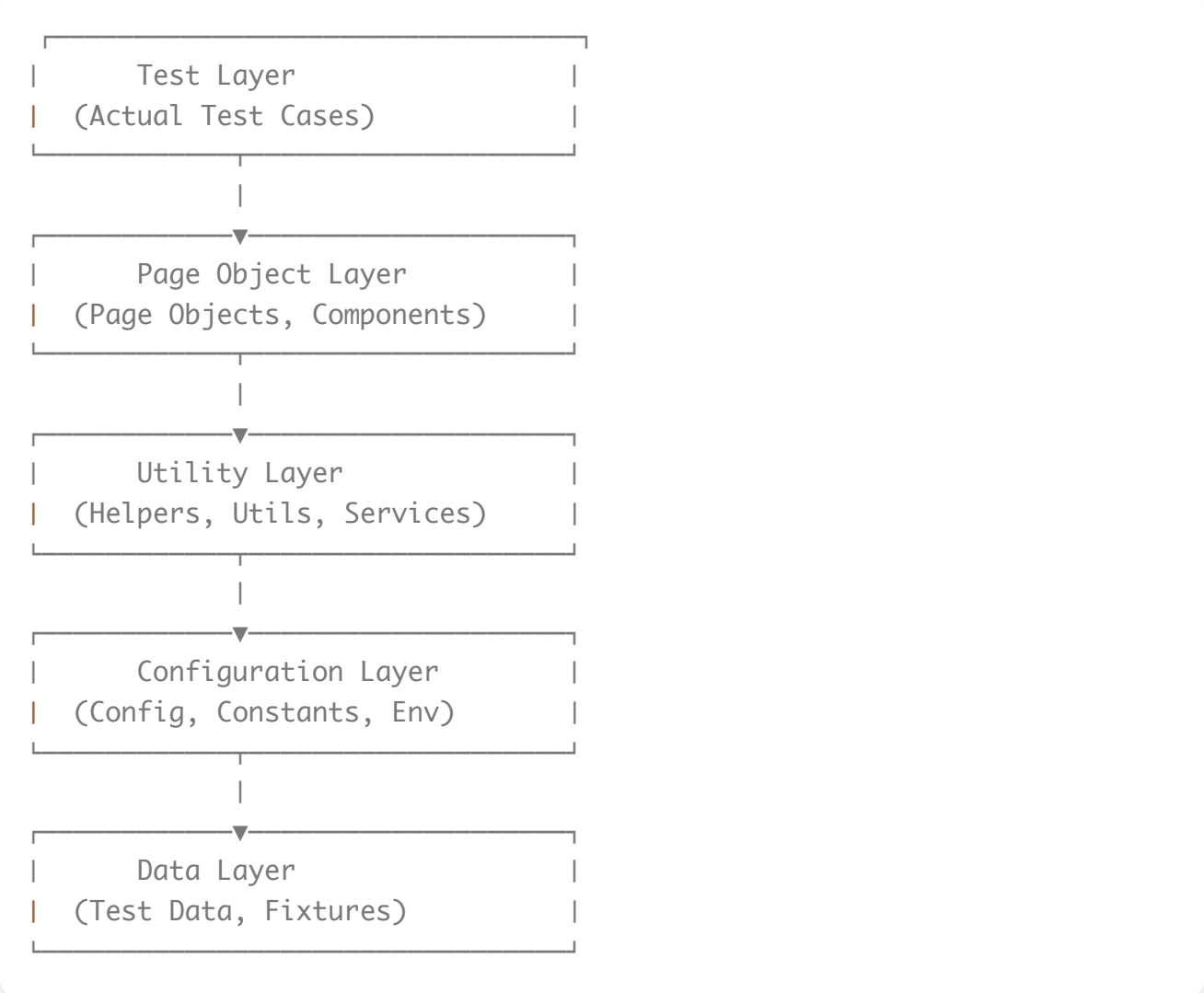
- **S**ingle Responsibility
- **O**pen/Closed

- **D**ependency Inversion

Testing Principles:

- DRY (Don't Repeat Yourself)
- Clear naming conventions
- Modular design
- Separation of concerns
- Maintainability first

Framework Layers



Directory Structure

```
|   |   ├── auth/
|   |   |   ├── login.spec.ts
|   |   |   └── signup.spec.ts
|   |   ├── products/
|   |   |   ├── search.spec.ts
|   |   |   └── checkout.spec.ts
|   |   └── profile/
|   |       └── settings.spec.ts
|   ├── api/
|   |   ├── users.spec.ts
|   |   └── products.spec.ts
|   └── mobile/
|       └── app.spec.ts
|
├── pages/
|   ├── BasePage.ts
|   ├── LoginPage.ts
|   ├── ProductPage.ts
|   └── CheckoutPage.ts
|
├── components/
|   ├── Header.ts
|   ├── Footer.ts
|   └── SearchBar.ts
|
├── utils/
|   ├── helpers.ts
|   ├── logger.ts
|   ├── database.ts
|   └── api-client.ts
|
├── config/
|   ├── environments/
|   |   ├── dev.config.ts
|   |   ├── staging.config.ts
|   |   └── prod.config.ts
|   └── constants.ts
|
├── fixtures/
|   ├── users.json
|   ├── products.json
|   └── test-data.ts
|
├── reports/
```

```
├── .github/
|   └── workflows/
|       ├── ci.yml
|       └── nightly.yml
|
├── playwright.config.ts
├── package.json
├── tsconfig.json
└── README.md
```

Base Page Class

```typescript
// pages/BasePage.ts
import { Page, Locator } from '@playwright/test'
import { Logger } from '../utils/logger'

export class BasePage {
  protected page: Page
  protected logger: Logger

  constructor(page: Page) {
    this.page = page
    this.logger = new Logger(this.constructor.name)
  }

  async goto(path: string = ''): Promise<void> {
    const url = `${process.env.BASE_URL}${path}`
    this.logger.info(`Navigating to: ${url}`)
    await this.page.goto(url)
  }

  async waitForPageLoad(): Promise<void> {
    await this.page.waitForLoadState('networkidle')
    this.logger.info('Page loaded')
  }

  async takeScreenshot(name: string): Promise<void> {
    await this.page.screenshot({
      path: `screenshots/${name}-${Date.now()}.png`,
      fullPage: true
    })
  }
```

```
  }

  async getCurrentUrl(): Promise<string> {
    return this.page.url()
  }

  protected async click(locator: Locator): Promise<void> {
    this.logger.info(`Clicking element`)
    await locator.click()
  }

  protected async fill(locator: Locator, text: string): Promise<void> {
    this.logger.info(`Filling text: ${text}`)
    await locator.fill(text)
  }

  protected async selectOption(locator: Locator, value: string): Promise<v
    this.logger.info(`Selecting option: ${value}`)
    await locator.selectOption(value)
  }
}
```

Page Object Example

```
 // pages/LoginPage.ts
import { Page, Locator } from '@playwright/test'
import { BasePage } from './BasePage'

export class LoginPage extends BasePage {
  // Locators
  private get usernameInput(): Locator {
    return this.page.getByLabel('Username')
  }

  private get passwordInput(): Locator {
    return this.page.getByLabel('Password')
  }

  private get loginButton(): Locator {
    return this.page.getByRole('button', { name: 'Login' })
  }
```

```
  private get rememberMeCheckbox(): Locator {
    return this.page.getByLabel('Remember me')
  }

  // Actions
  async login(username: string, password: string): Promise<void> {
    this.logger.info(`Logging in as: ${username}`)
    await this.fill(this.usernameInput, username)
    await this.fill(this.passwordInput, password)
    await this.click(this.loginButton)
    await this.waitForPageLoad()
  }

  async loginWithRememberMe(username: string, password: string): Promise<v
    await this.fill(this.usernameInput, username)
    await this.fill(this.passwordInput, password)
    await this.rememberMeCheckbox.check()
    await this.click(this.loginButton)
  }

  async getErrorMessage(): Promise<string> {
    return await this.errorMessage.textContent() || ''
  }

  async isErrorMessageVisible(): Promise<boolean> {
    return await this.errorMessage.isVisible()
  }

  async clearForm(): Promise<void> {
    await this.usernameInput.clear()
    await this.passwordInput.clear()
  }
}
```

Component Class

```
 // components/Header.ts
import { Page, Locator } from '@playwright/test'

export class Header {
  private page: Page
```

```
  }

  private get logo(): Locator {
    return this.page.getByRole('img', { name: 'Logo' })
  }

  private get searchBox(): Locator {
    return this.page.getByPlaceholder('Search...')
  }

  private get cartIcon(): Locator {
    return this.page.getByRole('link', { name: 'Cart' })
  }

  private get userMenu(): Locator {
    return this.page.getByRole('button', { name: 'Account' })
  }

  async search(query: string): Promise<void> {
    await this.searchBox.fill(query)
    await this.searchBox.press('Enter')
  }

  async goToCart(): Promise<void> {
    await this.cartIcon.click()
  }

  async logout(): Promise<void> {
    await this.userMenu.click()
    await this.page.getByRole('menuitem', { name: 'Logout' }).click()
  }

  async getCartItemCount(): Promise<number> {
    const text = await this.page.locator('.cart-count').textContent()
    return parseInt(text || '0')
  }
}
```

Configuration Management

```
// config/environments/base.config.ts
export interface Environment {
  baseUrl: string
  apiUrl: string
  timeout: number
  retries: number
  headless: boolean
}

// config/environments/dev.config.ts
export const devConfig: Environment = {
  baseUrl: 'http://localhost:3000',
  apiUrl: 'http://localhost:8080/api',
  timeout: 30000,
  retries: 0,
  headless: false
}

// config/environments/staging.config.ts
export const stagingConfig: Environment = {
  baseUrl: 'https://staging.example.com',
  apiUrl: 'https://api-staging.example.com',
  timeout: 30000,
  retries: 2,
  headless: true
}

// config/environments/prod.config.ts
export const prodConfig: Environment = {
  baseUrl: 'https://example.com',
  apiUrl: 'https://api.example.com',
  timeout: 30000,
  retries: 3,
  headless: true
}
```

Configuration loader:

```
// config/config.ts
import { devConfig } from './environments/dev.config'
import { stagingConfig } from './environments/staging.config'
import { prodConfig } from './environments/prod.config'
import { Environment } from './environments/base.config'
```

```
  switch (env) {
    case 'dev':
      return devConfig
    case 'staging':
      return stagingConfig
    case 'prod':
      return prodConfig
    default:
      throw new Error(`Unknown environment: ${env}`)
  }
}

export const config = getConfig()
```

Playwright Configuration

```
 // playwright.config.ts
import { defineConfig, devices } from '@playwright/test'
import { config } from './config/config'

export default defineConfig({
  testDir: './tests',
  timeout: config.timeout,
  retries: config.retries,
  workers: process.env.CI ? 4 : 2,

  reporter: [
    ['html'],
    ['allure-playwright'],
    ['junit', { outputFile: 'test-results/junit.xml' }]
  ],

  use: {
    baseURL: config.baseUrl,
    headless: config.headless,
    screenshot: 'only-on-failure',
    video: 'retain-on-failure',
    trace: 'on-first-retry',
  },

  projects: [
```

```
    },
    {
      name: 'firefox',
      use: { ...devices['Desktop Firefox'] },
    },
    {
      name: 'webkit',
      use: { ...devices['Desktop Safari'] },
    },
    {
      name: 'mobile-chrome',
      use: { ...devices['Pixel 5'] },
    },
    {
      name: 'mobile-safari',
      use: { ...devices['iPhone 12'] },
    },
  ],
})
```

Constants and Enums

```
 // config/constants.ts
export const TIMEOUTS = {
  SHORT: 5000,
  MEDIUM: 10000,
  LONG: 30000,
  EXTRA_LONG: 60000
}

export const ROUTES = {
  HOME: '/',
  LOGIN: '/login',
  SIGNUP: '/signup',
  PRODUCTS: '/products',
  CART: '/cart',
  CHECKOUT: '/checkout',
  PROFILE: '/profile'
}

export const API_ENDPOINTS = {
  USERS: '/users',
```

```
}

export enum UserRole {
  ADMIN = 'admin',
  USER = 'user',
  GUEST = 'guest'
}

export enum OrderStatus {
  PENDING = 'pending',
  PROCESSING = 'processing',
  SHIPPED = 'shipped',
  DELIVERED = 'delivered',
  CANCELLED = 'cancelled'
}
```

End-to-End Framework Integration

Complete Test Example

```
 // tests/e2e/checkout.spec.ts
import { test, expect } from '@playwright/test'
import { LoginPage } from '../../pages/LoginPage'
import { ProductPage } from '../../pages/ProductPage'
import { CartPage } from '../../pages/CartPage'
import { CheckoutPage } from '../../pages/CheckoutPage'
import { Header } from '../../components/Header'
import { testData } from '../../fixtures/test-data'
import { ROUTES } from '../../config/constants'

test.describe('E2E Checkout Flow', () => {
  let loginPage: LoginPage
  let productPage: ProductPage
  let cartPage: CartPage
  let checkoutPage: CheckoutPage
  let header: Header

  test.beforeEach(async ({ page }) => {
```

```
  checkoutPage = new CheckoutPage(page)
  header = new Header(page)

  // Login
  await loginPage.goto(ROUTES.LOGIN)
  await loginPage.login(
    testData.users.standard.username,
    testData.users.standard.password
  )
})

test('should complete checkout successfully', async ({ page }) => {
  // Search and add product
  await header.search('iPhone 15')
  await productPage.selectProduct('iPhone 15 Pro')
  await productPage.addToCart()

  // Verify cart
  expect(await header.getCartItemCount()).toBe(1)

  // Go to cart
  await header.goToCart()
  await expect(page).toHaveURL(/.*cart/)

  // Proceed to checkout
  await cartPage.proceedToCheckout()

  // Fill shipping info
  await checkoutPage.fillShippingInfo({
    firstName: 'John',
    lastName: 'Doe',
    address: '123 Main St',
    city: 'New York',
    zipCode: '10001',
    phone: '555-1234'
  })

  // Fill payment info
  await checkoutPage.fillPaymentInfo({
    cardNumber: '4111111111111111',
    expiryDate: '12/25',
    cvv: '123'
  })

  // Place order
```

```
    await expect(page.getByText('Order placed successfully')).toBeVisible(
    const orderNumber = await checkoutPage.getOrderNumber()
    expect(orderNumber).toMatch(/^ORD-\d+$/)
  })

  test('should show error for invalid payment', async ({ page }) => {
    await header.search('MacBook Pro')
    await productPage.selectProduct('MacBook Pro 16"')
    await productPage.addToCart()
    await header.goToCart()
    await cartPage.proceedToCheckout()

    await checkoutPage.fillShippingInfo(testData.shippingInfo.default)
    await checkoutPage.fillPaymentInfo({
      cardNumber: '0000000000000000',
      expiryDate: '12/25',
      cvv: '123'
    })

    await checkoutPage.placeOrder()

    await expect(page.getByText('Invalid payment information')).toBeVisibl
  })
})
```

API Integration

```
// utils/api-client.ts
import axios, { AxiosInstance } from 'axios'
import { config } from '../config/config'

export class ApiClient {
  private client: AxiosInstance
  private token?: string

  constructor() {
    this.client = axios.create({
      baseURL: config.apiUrl,
      timeout: 10000,
      headers: {
        'Content-Type': 'application/json'
      }
```

```
    this.client.interceptors.request.use((config) => {
      if (this.token) {
        config.headers.Authorization = `Bearer ${this.token}`
      }
      return config
    })

    // Response interceptor
    this.client.interceptors.response.use(
      (response) => response,
      (error) => {
        console.error('API Error:', error.response?.data || error.message)
        return Promise.reject(error)
      }
    )
  }

  async login(username: string, password: string): Promise<void> {
    const response = await this.client.post('/auth/login', {
      username,
      password
    })
    this.token = response.data.token
  }

  async createUser(userData: any): Promise<any> {
    const response = await this.client.post('/users', userData)
    return response.data
  }

  async getUser(id: string): Promise<any> {
    const response = await this.client.get(`/users/${id}`)
    return response.data
  }

  async createProduct(productData: any): Promise<any> {
    const response = await this.client.post('/products', productData)
    return response.data
  }

  async deleteProduct(id: string): Promise<void> {
    await this.client.delete(`/products/${id}`)
  }
}
```

```
// utils/database.ts
import { Pool } from 'pg'

export class Database {
  private pool: Pool

  constructor() {
    this.pool = new Pool({
      host: process.env.DB_HOST,
      port: parseInt(process.env.DB_PORT || '5432'),
      database: process.env.DB_NAME,
      user: process.env.DB_USER,
      password: process.env.DB_PASSWORD
    })
  }

  async query(sql: string, params?: any[]): Promise<any> {
    const client = await this.pool.connect()
    try {
      const result = await client.query(sql, params)
      return result.rows
    } finally {
      client.release()
    }
  }

  async getUserByEmail(email: string): Promise<any> {
    const result = await this.query(
      'SELECT * FROM users WHERE email = $1',
      [email]
    )
    return result[0]
  }

  async createUser(userData: any): Promise<any> {
    const result = await this.query(
      'INSERT INTO users (email, name, password) VALUES ($1, $2, $3) RETUR
      [userData.email, userData.name, userData.password]
    )
    return result[0]
  }

  async deleteUser(email: string): Promise<void> {
    await this.query('DELETE FROM users WHERE email = $1', [email])
  }
```

```
  }
}
```

Logger Utility

```
// utils/logger.ts
import winston from 'winston'

export class Logger {
  private logger: winston.Logger

  constructor(context: string) {
    this.logger = winston.createLogger({
      level: process.env.LOG_LEVEL || 'info',
      format: winston.format.combine(
        winston.format.timestamp(),
        winston.format.printf(({ timestamp, level, message }) => {
          return `[${timestamp}] [${context}] ${level.toUpperCase()}: ${me
        })
      ),
      transports: [
        new winston.transports.Console(),
        new winston.transports.File({ filename: 'logs/test.log' })
      ]
    })
  }

  info(message: string): void {
    this.logger.info(message)
  }

  error(message: string): void {
    this.logger.error(message)
  }

  warn(message: string): void {
    this.logger.warn(message)
  }

  debug(message: string): void {
    this.logger.debug(message)
  }
```

Capstone Project Guidelines

Project Requirements

Functional Requirements:

1. Web automation (Playwright)
2. API automation (minimum 5 endpoints)
3. Mobile automation (WebDriverIO - bonus)
4. Cross-browser testing
5. Data-driven tests
6. Page Object Model
7. Fixtures and test data
8. CI/CD integration (GitHub Actions)
9. Test reporting (Allure or ReportPortal)
10. BrowserStack integration

Technical Requirements:

1. TypeScript/JavaScript
2. Proper directory structure
3. Configuration management
4. Error handling
5. Logging
6. Documentation (README)
7. Git version control
8. Code quality (ESLint)

Project Ideas

- Product search and filtering
- Shopping cart operations
- Checkout process
- Order history
- Profile management

Banking Application:

- Account creation
- Login/logout
- Fund transfer
- Transaction history
- Bill payment
- Account statements

Social Media Platform:

- User registration
- Profile creation
- Post creation
- Comments and likes
- Friend requests
- Messaging

Project Deliverables

1. Source Code

- Complete test framework
- All test cases
- Configuration files
- Documentation

- Test execution summary
- Screenshots/videos of failures
- Coverage metrics

3. Documentation

- README with setup instructions
- Architecture documentation
- Test case documentation
- API documentation

4. CI/CD Pipeline

- GitHub Actions workflow
- Automated test execution
- Report generation
- Notifications

5. Presentation

- Project overview (10 mins)
- Demo (10 mins)
- Q&A (5 mins)

Project Presentation

Presentation Structure

1. Introduction (2 mins)

- Project overview
- Technology stack

- Directory structure
- Design patterns used
- Key components

3. Test Demonstrations (10 mins)

- Web automation demo
- API automation demo
- Cross-browser execution
- Mobile automation (if included)

4. CI/CD Integration (3 mins)

- GitHub Actions workflow
- Automated execution
- Report generation

5. Test Reports (2 mins)

- Allure/ReportPortal dashboard
- Test metrics
- Coverage statistics

6. Challenges & Solutions (3 mins)

- Technical challenges faced
- Solutions implemented
- Lessons learned

7. Q&A (5 mins)

- Answer questions
- Demonstrate additional features

- Practice beforehand
- Keep slides minimal
- Focus on live demo
- Highlight unique features
- Show actual test execution
- Demonstrate reports

Don't:

- Read from slides
- Show too much code
- Skip error handling
- Ignore questions
- Rush through demo

Career Guidance

QE Career Path

```
 Entry Level:
├── Manual QA Tester
└── Junior Test Automation Engineer
    |
Mid Level:
├── Test Automation Engineer
├── SDET (Software Development Engineer in Test)
└── QA Engineer
    |
Senior Level:
├── Senior SDET
├── Test Automation Architect
├── QA Lead
```

```
├── QA Manager
├── Test Automation Manager
└── Director of Quality Engineering
```

Skills to Develop

Technical Skills:

- Programming (JavaScript/TypeScript, Python, Java)
- Test frameworks (Playwright, Cypress, Selenium)
- API testing (REST, GraphQL)
- Mobile testing (Appium, Detox)
- CI/CD (GitHub Actions, Jenkins)
- Cloud platforms (AWS, Azure, GCP)
- Performance testing (JMeter, k6)
- Security testing basics
- Database knowledge (SQL, NoSQL)

Soft Skills:

- Communication
- Problem-solving
- Attention to detail
- Time management
- Team collaboration
- Continuous learning

Resume Tips

Highlight:

- Technical skills
- Frameworks and tools

- Defect detection rate
- CI/CD implementation

Example Achievement:

```
"Designed and implemented end-to-end test automation framework using Playwright and TypeScript, achieving 85% test coverage across 3 platforms (Web, Mobile, API), reducing regression testing time by 70%"
```

Interview Preparation

Common Questions:

1. Explain your automation framework
2. How do you handle dynamic elements?
3. What are Page Objects?
4. How do you manage test data?
5. Explain your CI/CD setup
6. How do you handle flaky tests?
7. What's your approach to API testing?
8. How do you prioritize test cases?
9. Explain cross-browser testing strategy
10. How do you report bugs?

Continuous Learning

Resources:

- Playwright documentation
- Cypress documentation
- Test automation blogs
- GitHub repositories
- YouTube tutorials

- Tech conferences

Practice:

- Build personal projects
- Contribute to open source
- Solve coding challenges
- Create portfolio website
- Write technical blogs
- Participate in hackathons

Certification and Next Steps

Certification

Congratulations! 🎉

You have successfully completed the **QE - Automation Training Program**.

You have mastered: ✅ TypeScript & JavaScript ✅ Asynchronous Programming ✅ Playwright Automation ✅ API Testing ✅ Cypress Framework ✅ Mobile Testing (WebDriverIO) ✅ BrowserStack Integration ✅ AWS for Testing ✅ CI/CD with GitHub Actions ✅ Test Reporting ✅ AI in Test Automation ✅ Framework Design

Next Steps

Immediate (Week 1-2):

1. Complete capstone project
2. Update resume
3. Create LinkedIn profile
4. Build portfolio website

1. Apply for QA positions
2. Practice interviews
3. Contribute to open source
4. Build advanced projects
5. Network with QA professionals

Long-term (Year 1):

1. Gain work experience
2. Obtain certifications
3. Master additional tools
4. Mentor others
5. Speak at meetups

Recommended Certifications

Test Automation:

- ISTQB Test Automation Engineer
- Selenium Certification
- Playwright Certification (community)

Cloud:

- AWS Certified Cloud Practitioner
- Azure Fundamentals

Programming:

- JavaScript certification
- TypeScript certification

Final Checklist

- [ ] Presented project
- [ ] Resume updated
- [ ] LinkedIn profile created
- [ ] GitHub portfolio ready
- [ ] Applied for jobs
- [ ] Continued learning

Summary

In Day 15, you mastered:

✅ Framework Architecture

- Design principles
- Layer organization
- Directory structure
- Component design

✅ Configuration Management

- Environment configs
- Constants and enums
- Playwright configuration
- Multi-environment support

✅ Complete Integration

- End-to-end tests
- API integration
- Database utilities
- Logging system

- Deliverables
- Presentation guidelines
- Best practices

✅ Career Development

- Career path
- Skill development
- Resume tips
- Interview preparation

✅ Certification

- Program completion
- Next steps
- Continuous learning
- Job search strategy

Final Key Takeaways

- **Good architecture** ensures maintainability
- **Configuration** enables flexibility
- **Page Objects** reduce duplication
- **Utilities** improve efficiency
- **Documentation** helps collaboration
- **CI/CD** ensures quality
- **Continuous learning** is essential

You Are Now Ready!

Congratulations on completing the 120-hour QE Automation Training Program!

You now have the skills to:

- Integrate with CI/CD pipelines
- Generate comprehensive reports
- Use AI in test automation
- Work professionally as QA Engineer

Thank You! 🎓

Keep learning, keep testing, and best of luck in your QA career!

End of Day 15 Documentation

End of QE Automation Training Program

🎊 **PROGRAM COMPLETE!** 🎊

← AI in Test Automation