PARALLEL UNIVERSE: ARCHITECTING SCALABLE E2E TESTS WITH PLAYWRIGHT

From Flaky Failures to Lightning-Fast Feedback

OĞUZ CEYLAN

💼 Software Engineer @ Microsoft

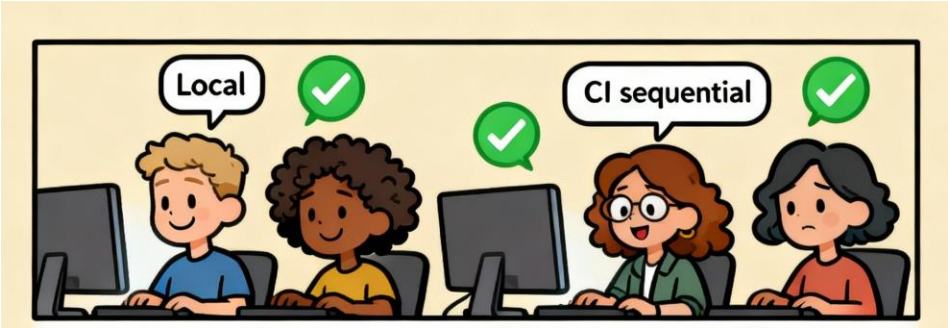
📍 Prague

⛹️ Football, Softball, Hiking, Running

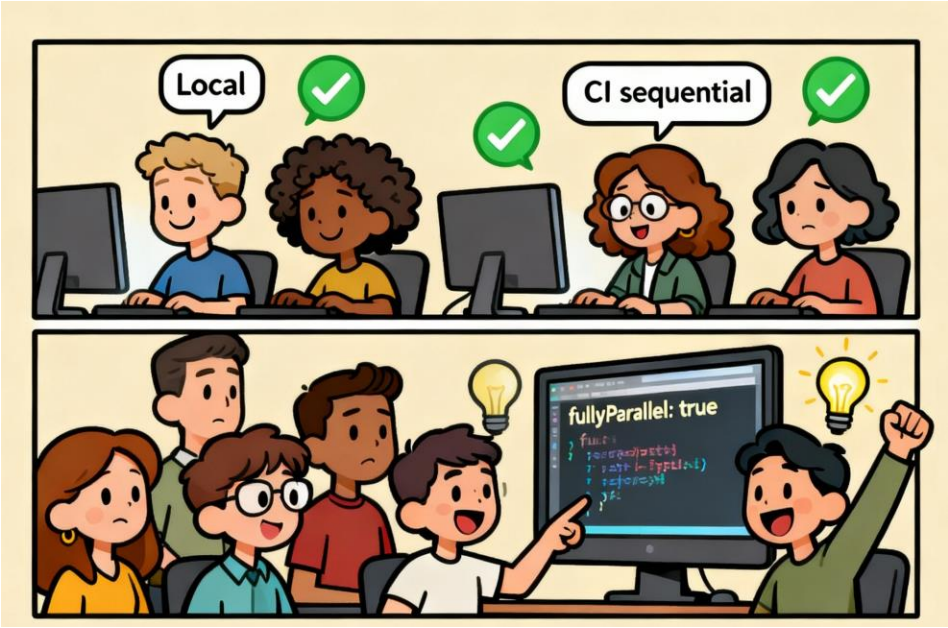
PLAYWRIGHT

- Open-source web automation framework
- Cross-browser testing (Chromium, Firefox, WebKit)
- Supports parallel and modern web testing features
- Integrates easily with CI/CD pipelines

SEQUENTIAL TESTING



PARALLEL TESTING




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

export default defineConfig({
        fullyParallel: true,
});
```

playwright.config.ts

PARALLEL TESTING TRAP

EXPECTATIONS VS REALITY

Parallel testing is an architecture problem, not a configuration problem.

WHY IT MATTERS

Flaky, slow tests cost teams time, confidence, and ultimately product quality.

BEFORE AND AFTER

Metric	Before Architecture	After Architecture	Improvement
⏱️ Execution Time	45 minutes	8 minutes	82% reduction
📊 Pass Rate	60% (flaky)	98% (stable)	38% improvement
🚀 Deployment Frequency	1x per week	Daily	Flexible/trusted release cycle
😊 Developer Confidence	Low	High	Restored
🐛 Time Spent Debugging	8 hours/week	1 hour/week	87% reduction

THE BIG FOUR PROBLEMS

- ⚠️ Test Data Contamination
- ⚠️ Shared Resource Conflicts
- ⚠️ Poor Test Isolation
- ⚠️ CI Environment Bottlenecks

⚠️ TEST DATA CONTAMINATION

```
// hardcoded-data.spec.ts
import { test, expect } from '@playwright/test';

test('update user profile', async ({ page }) => {
  await page.goto('/login');
  await page.fill('#user', 'test@example.com'); // Same user everywhere!
  await page.fill('#password', 'pass123');
  await page.click('#login');

  await page.goto('/profile');
  await page.fill('#name', 'New Name');
  await page.click('#save');
  // 💥 Random failures - another worker modified this user
});

test('delete user account', async ({ page }) => {
  await page.goto('/login');
  await page.fill('#user', 'test@example.com'); // Same user everywhere!
  await page.fill('#password', 'pass123');
  await page.click('#login');

  await page.goto('/settings');
  await page.click('#delete-account');
  await page.click('#confirm-delete');
  // 💥 Other tests break - user is gone!
});
```

✅ WORKER-SCOPED FIXTURES

```ts
// solution-example.spec.ts
import { test, expect } from './fixtures/user-fixture';

test('update user profile', async ({ page, uniqueUser }) => {
  await page.goto('/login');
  await page.fill('#user', uniqueUser.email); // ✅ Unique per worker!
  await page.fill('#password', uniqueUser.password);
  await page.click('#login');

  await page.goto('/profile');
  await page.fill('#name', 'New Name');
  await page.click('#save');
  // ✅ No conflicts - Worker 0 uses user-0@test.com
});

test('delete user account', async ({ page, uniqueUser }) => {
  await page.goto('/login');
  await page.fill('#user', uniqueUser.email); // ✅ Same worker, safe!
  await page.fill('#password', uniqueUser.password);
  await page.click('#login');

  await page.goto('/settings');
  await page.click('#delete-account');
  await page.click('#confirm-delete');
  // ✅ Tests run sequentially in same worker - safe!
});
```

```ts
// fixtures/user-fixture.ts
import { test as baseTest } from '@playwright/test';

type WorkerFixtures = {
  uniqueUser: {
    email: string;
    password: string;
  };
};

export const test = baseTest.extend<{}, WorkerFixtures>({
  uniqueUser: [
    async ({ }, use, workerInfo) => {
      const user = {
        email: `user-${workerInfo.workerIndex}@test.com`,
        password: 'pass123'
      };

      await createInDB(user);
      await use(user);
      await deleteFromDB(user);
    },
    { scope: 'worker' }
  ],
});

export { expect } from '@playwright/test';
```

⚠️ SHARED RESOURCE CONFLICTS

```ts
// problem-file-conflicts.spec.ts
import { test, expect } from '@playwright/test';
import fs from 'fs/promises';

test('generate sales report', async ({ page }) => {
  await page.goto('/dashboard');

  const [download] = await Promise.all([
    page.waitForEvent('download'),
    page.click('#export-csv')
  ]);

  await download.saveAs('./downloads/report.csv'); // Same file!

  const data = await fs.readFile('./downloads/report.csv', 'utf-8');
  expect(data).toContain('Total Sales');
  // 💥 Another worker overwrote this file!
});

test('generate invoice PDF', async ({ page }) => {
  await page.goto('/invoices');

  const [download] = await Promise.all([
    page.waitForEvent('download'),
    page.click('#export-pdf')
  ]);

  await download.saveAs('./downloads/invoice.pdf'); // Same file!

  const size = (await fs.stat('./downloads/invoice.pdf')).size;
  expect(size).toBeGreaterThan(0);
  // 💥 File conflicts with other workers!
});
```

✅ TEST-SCOPED FIXTURES

```ts
// solution-file-isolation.spec.ts
import { test, expect } from './fixtures/file-isolation';
import fs from 'fs/promises';

test('generate sales report', async ({ page, downloadPath }) => {
  await page.goto('/dashboard');

  const [download] = await Promise.all([
    page.waitForEvent('download'),
    page.click('#export-csv')
  ]);

  const filePath = downloadPath.getPath('report.csv'); // ✅ Unique directory!
  await download.saveAs(filePath);

  const data = await fs.readFile(filePath, 'utf-8');
  expect(data).toContain('Total Sales');
  // ✅ Isolated: downloads/a1b2c3d4-uuid/report.csv
});

test('generate invoice PDF', async ({ page, downloadPath }) => {
  await page.goto('/invoices');

  const [download] = await Promise.all([
    page.waitForEvent('download'),
    page.click('#export-pdf')
  ]);

  const filePath = downloadPath.getPath('invoice.pdf'); // ✅ Different
directory!
  await download.saveAs(filePath);

  const size = (await fs.stat(filePath)).size;
  expect(size).toBeGreaterThan(0);
  // ✅ Isolated: downloads/xyz789-uuid/invoice.pdf - Auto cleanup!
});
```

```ts
// fixtures/file-isolation.ts
import { test as baseTest } from '@playwright/test';
import { randomUUID } from 'crypto';
import path from 'path';
import fs from 'fs/promises';

type FileFixtures = {
  downloadPath: {
    getPath: (filename: string) => string;
  };
};

export const test = baseTest.extend<FileFixtures>({
  downloadPath: async ({}, use) => {
    const testId = randomUUID();
    const baseDir = path.join('./downloads', testId);

    await fs.mkdir(baseDir, { recursive: true });

    await use({
      getPath: (filename: string) => path.join(baseDir, filename)
    });

    await fs.rm(baseDir, { recursive: true, force: true });
  },
});

export { expect } from '@playwright/test';
```

⚠ POOR TEST ISOLATION

```
// problem-auth-repetition.spec.ts
import { test, expect } from '@playwright/test';

test('user views dashboard', async ({ page }) => {
  await page.goto('/login');
  await page.fill('#username', 'user@test.com');
  await page.fill('#password', 'password123');
  await page.click('#login-button');
  await page.waitForURL('/dashboard');

  await expect(page.locator('h1')).toHaveText('Welcome Back');
  // 3 seconds wasted on login
});

test('user updates settings', async ({ page }) => {
  await page.goto('/login');
  await page.fill('#username', 'user@test.com');
  await page.fill('#password', 'password123');
  await page.click('#login-button');
  await page.waitForURL('/dashboard');

  await page.goto('/settings');
  await page.fill('#timezone', 'UTC');
  await page.click('#save');
  // Another 3 seconds wasted on login
});
```

✅ STORAGE STATE PATTERN

```ts
// solution-storage-state.spec.ts
import { test, expect } from '@playwright/test';

test('user views dashboard', async ({ page }) => {
  await page.goto('/dashboard');
// Already logged in! ✅

  await expect(page.locator('h1')).toHaveText('Welcome Back');
  // No login needed - instant!
});

test('user updates settings', async ({ page }) => {
  await page.goto('/settings');
// Already logged in! ✅

  await page.fill('#timezone', 'UTC');
  await page.click('#save');
  // Saved 3 seconds!
});
```

```ts
// auth.setup.ts
import { test as setup, expect } from '@playwright/test';

setup('authenticate as user', async ({ page }) => {
  await page.goto('/login');
  await page.fill('#username', 'user@test.com');
  await page.fill('#password', 'password123');
  await page.click('#login-button');
  await page.waitForURL('/dashboard');

  // Save authentication state
  await page.context().storageState({ path: 'auth/user.json' });
});
```

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

export default defineConfig({
  projects: [
    { name: 'setup', testMatch: /.*\.setup\.ts/ },
    {
      name: 'chromium',
      use: {
        storageState: 'auth/user.json'  // Load saved auth state
      },
      dependencies: ['setup'],
    },
  ],
});
```

✅ CI ENVIRONMENT OPTIMIZATION

```
// playwright.config.ts - Environment-aware
import { defineConfig } from '@playwright/test';

const isCI = !!process.env.CI;

export default defineConfig({
  testDir: './tests',

  workers: isCI ? 2 : undefined, // CI: 2 workers (stable), Local: Use all cores (fast)

  retries: isCI ? 2 : 0, // CI: Retry flaky tests, Local: Fail fast

  timeout: isCI ? 60000 : 30000, // CI: Longer timeout (slower resources), Local: Quick timeout

  maxFailures: isCI ? 10 : undefined, // CI: Stop after 10 failures (save costs), Local: Run all

  forbidOnly: isCI, // CI: Strict mode, Local: Flexible

  fullyParallel: true,

  },
});
```

 THE PARALLEL TESTING LANDSCAPE

MORE PATTERNS

- Factory Pattern (test data builders)
- Builder Pattern (fluent APIs)
- Database Per Worker (infrastructure isolation)
- API Mocking (external dependencies)
- Sharding (distribute across machines)
- Setup Dependencies (orchestration)
- Serial Mode (ordered execution)
- Web-First Assertions (auto-wait)

TECHNIQUES

- Port Allocation
- File System Isolation
- API Mocking
- Storage State
- UUID Generation
- Webhook Isolation
- Temporary Data Creation
- Multiple Auth States
- .....

IS THERE AN EASIER WAY?

The Challenge

- ⚠ Manual test architecture requires deep expertise
- ⚠ Time-consuming to implement isolation patterns
- ⚠ Difficult to debug parallel execution issues

The Solution

- 🤖 Large Language Models (LLMs) for test generation and debugging
- 🔧 MCP (Model Context Protocol) servers for orchestration
- 🛠️ Automated tools for parallel-safe architecture generation

PLAYWRIGHT MCP SERVER

Model Context Protocol server for browser automation with Playwright.

- Fast accessibility tree based
- LLM-friendly structured data
- Deterministic tool application

Playwright MCP

INTRODUCING PLAYWRIGHT WIZARD MCP SERVER

🧙‍♂️ An intelligent Model Context Protocol (MCP) server that guides you through creating professional Playwright test suites with best practices built in.

Features

🧙‍♂️ Step-by-step wizard workflow for creating comprehensive test suites
📚 Comprehensive prompts covering analysis, planning, setup, and implementation
🎯 Best practices for selectors, fixtures, and parallel execution
🔧 Optional enhancements for accessibility and API testing
📖 Reference documentation for advanced patterns
🌐 MCP Registry integration for easy discovery and installation

Download package on NPM

https://github.com/oguzc/playwright-wizard-mcp

INTRODUCING PLAYWRIGHT WIZARD MCP SERVER

Core Workflow

Tool	Purpose
analyze-app	Analyzes your application structure and tech stack
generate-test-plan	Creates comprehensive test scenarios and acceptance criteria
setup-infrastructure	Sets up Playwright config, fixtures, and folder structure
generate-page-objects	Generates type-safe page object models with optimal selectors
implement-test-suite	Writes actual tests with assertions and error handling

Optional Enhancements

Tool	Purpose
setup-ci-cd	Adds GitHub Actions for automated testing
add-accessibility	Integrates axe-core for WCAG 2.1 AA compliance
add-api-testing	Adds REST/GraphQL/tRPC API testing
advanced-optimization	Deep performance optimization and auth state reuse

Download package on NPM

https://github.com/oguzc/playwright-wizard-mcp

TestMart

Home | Products

Demo User | Sign Out

Premium Products Collection

Discover our carefully curated selection of high-quality products designed to enhance your lifestyle

Premium Wireless Headphones

High-quality wireless headphones with noise cancellation

$299.99 | 10 in stock

Add to Cart

Smart Fitness Watch

Track your fitness goals with this smart watch

$199.99 | 5 in stock

Add to Cart

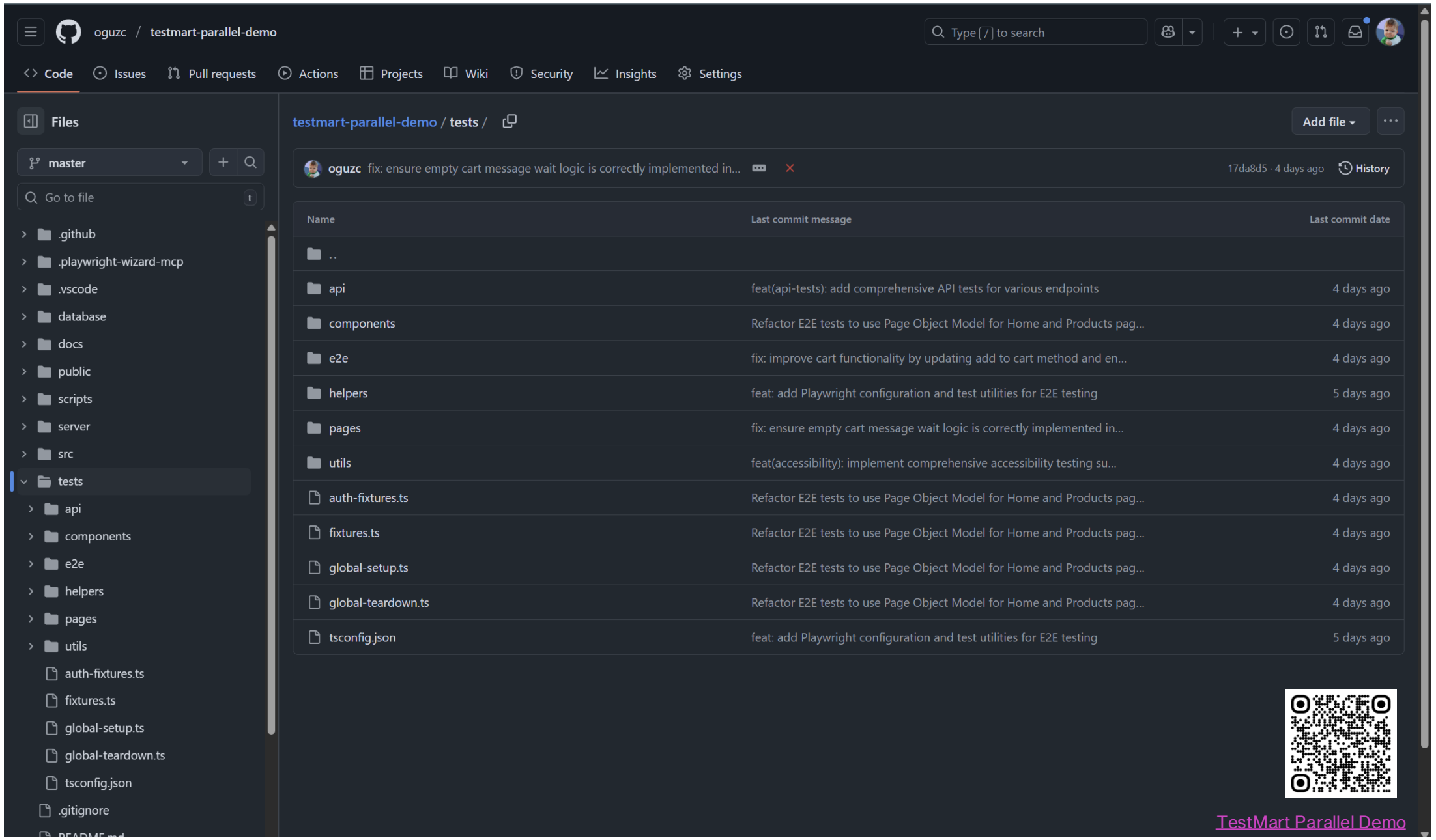
Laptop Backpack

Durable backpack with laptop compartment

$79.99 | 20 in stock

Add to Cart

TestMart Parallel Demo


oguzc / testmart-parallel-demo
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Files
master
Go to file
t
.github
.playwright-wizard-mcp
.vscode
database
docs
public
scripts
server
src
tests
api
components
e2e
helpers
pages
utils
auth-fixtures.ts
fixtures.ts
global-setup.ts
global-teardown.ts
tsconfig.json
.gitignore
testmart-parallel-demo / tests /
Add file
oguzc fix: ensure empty cart message wait logic is correctly implemented in...
17da8d5 · 4 days ago
History
Name
Last commit message
Last commit date
..
api
feat(api-tests): add comprehensive API tests for various endpoints
4 days ago
components
Refactor E2E tests to use Page Object Model for Home and Products pag...
4 days ago
e2e
fix: improve cart functionality by updating add to cart method and en...
4 days ago
helpers
feat: add Playwright configuration and test utilities for E2E testing
5 days ago
pages
fix: ensure empty cart message wait logic is correctly implemented in...
4 days ago
utils
feat(accessibility): implement comprehensive accessibility testing su...
4 days ago
auth-fixtures.ts
Refactor E2E tests to use Page Object Model for Home and Products pag...
4 days ago
fixtures.ts
Refactor E2E tests to use Page Object Model for Home and Products pag...
4 days ago
global-setup.ts
Refactor E2E tests to use Page Object Model for Home and Products pag...
4 days ago
global-teardown.ts
Refactor E2E tests to use Page Object Model for Home and Products pag...
4 days ago
tsconfig.json
feat: add Playwright configuration and test utilities for E2E testing
5 days ago
TestMart Parallel Demo

PLAYWRIGHT TEST AGENTS

Playwright recently announced Playwright agents:

- 🎭 **planner** explores the app and produces a Markdown test plan
- 🎭 **generator** transforms the Markdown plan into the Playwright Test files
- 🎭 **healer** executes the test suite and automatically repairs failing tests

FURTHER RESOURCES

🛠️ Tools & Resources

- Playwright Wizard MCP
- Playwright MCP
- Playwright Agents
- TestMart Parallel Demo
- Playwright Parallel Testing

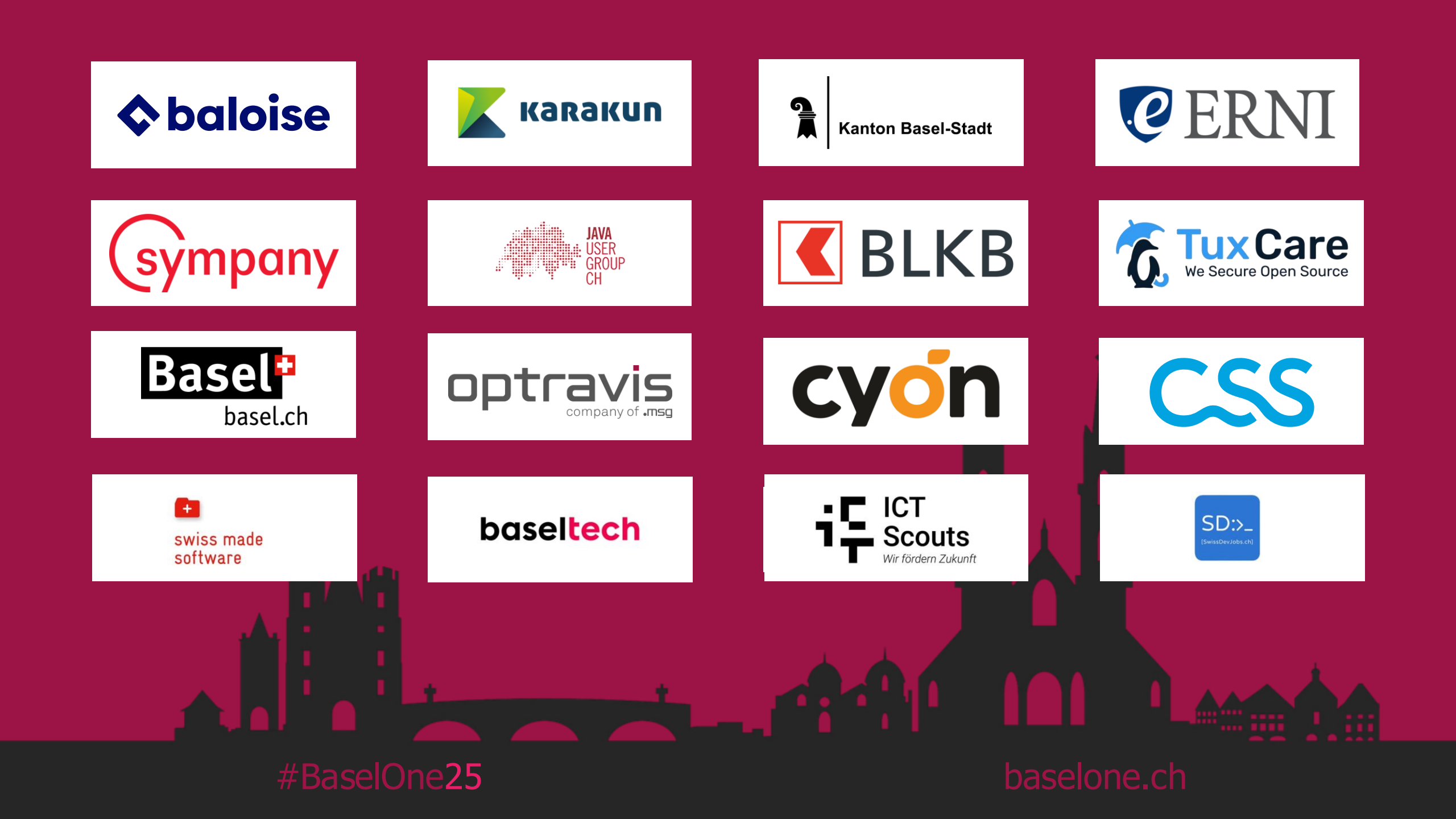
CONTACT

📧 Let's Connect

therealoguzceylan@gmail.com

linkedin.com/in/oguzceylan1

github.com/oguzc



#BaselOne25

baselone.ch