# Documentation – Feb_16 Lab

## End-to-End Integration Testing Using CI

### Personal Finance Tracker – Full Stack Implementation

---

# 1. Scenario Overview

### Scenario Title

**End-to-End Integration Testing Using CI**

### Scenario Description

The objective of this assignment was to enhance an existing Continuous Integration (CI) pipeline by introducing End-to-End (E2E) integration testing. The goal was to validate the complete workflow of the Personal Finance Tracker application, starting from user interface interaction and extending to backend data processing and persistence.

The implementation ensures that:

- UI navigation functions correctly
- API communication is stable
- Backend logic processes data correctly
- UI updates reflect backend changes
- The entire application flow is automatically validated within CI

---

# 2. Application Context

The Personal Finance Tracker is a full stack web application that enables users to:

- View dashboard summary (Income, Expenses, Balance)
- Add new expenses
- Add new income records
- Observe automatic balance recalculation

Balance Calculation:

```
Balance = Total Income – Total Expenses
```

---

# 3. Task Instructions Implementation

## 3.1 Setup

### Requirement:

Use Cypress or a similar E2E testing tool.

### Implementation:

Cypress was selected as the E2E automation framework due to:

- Browser automation capability
- Real user simulation
- Headless execution support
- CI integration compatibility

Cypress was installed as a development dependency in the root project.

---

## 3.2 Test Scenarios Implemented

The following E2E test cases were implemented as required:

### 1️⃣ Navigate to Dashboard and Load Summary Data

Validation Steps:

- Open application at http://localhost:3000
- Confirm dashboard title is visible
- Confirm income, expenses, and balance elements exist
- Validate numeric summary values are displayed

---

### 2️⃣ Add a New Expense and Verify It Appears in the List

Validation Steps:

- Enter expense title
- Enter expense amount
- Submit form
- Confirm new expense appears in expense list
- Validate that total expenses increase

---

### 3️⃣ Add a New Income Record and Verify It Reflects on Dashboard

Validation Steps:

- Enter income source
- Enter income amount
- Submit form
- Confirm income appears in list
- Validate total income increases
- Confirm balance recalculates correctly

---

# 4. CI Pipeline Configuration

## 4.1 CI Enhancement Objective

The CI pipeline was extended to automate:

- Backend testing
- Frontend startup
- End-to-end testing
- Automated service shutdown

---

## 4.2 CI Pipeline Responsibilities

The pipeline performs:

1. Install dependencies
2. Build or prepare frontend
3. Start backend server
4. Start frontend server
5. Wait until frontend is accessible
6. Execute Cypress headless tests
7. Shut down all services
8. Report test results

---

## 4.3 Service Orchestration Strategy

To ensure correct execution order:

- concurrently → Runs backend and frontend simultaneously
- wait-on → Ensures frontend server is ready before running tests

- Cypress headless mode → Executes automated tests
- Automatic termination after completion

**Root-level script:**

```
npm run test:e2e
```

Execution Flow:

```
Start Backend
      ↓
Start Frontend
      ↓
Wait for Frontend Availability
      ↓
Run Cypress E2E Tests
      ↓
Validate Results
      ↓
Gracefully Shut Down Services
```

---

# 5. Test Flow (As Required in Scenario)

The implemented test flow exactly matches the scenario instruction:

```
Start Backend → Start Frontend → Run E2E Tests → Validate Results
```

Detailed Flow:

1. Backend Express server runs on Port 5001
2. Frontend static server runs on Port 3000
3. Cypress connects to frontend
4. User interactions are simulated
5. API calls are triggered
6. Backend processes requests
7. UI updates dynamically
8. Cypress validates output
9. Process terminates automatically

---

# 6. Bonus Challenge Implementation

The scenario specifies:

Capture screenshots on test failure
Store test results and screenshots as CI artifacts

### Implementation Status:

✔ Cypress is configured to automatically capture screenshots on failure.

✔ Headless execution supports screenshot and video capture.

✔ These artifacts can be uploaded within GitHub Actions workflow if required.

Cypress default behavior:

- On test failure → Screenshot saved under cypress/screenshots
- Can be archived as CI artifacts

---

# 7. Expected Output Validation

The scenario expected:

### 1️⃣ Full application flow validated automatically

Achieved through:

- Cypress E2E automation
- Backend API integration
- CI execution

---

### 2️⃣ CI detects UI–API integration issues

Achieved through:

- Cypress failing if API fails
- Backend tests validating endpoints
- Automatic CI failure on error

---

### 3️⃣ Reliable end-to-end application verification

Achieved through:

- Deterministic testing strategy
- Numeric comparison instead of hardcoded values
- Service orchestration stability
- Headless execution reliability

---

# 8. Technical Implementation Summary

## Backend Layer

- Express server
- RESTful API endpoints
- In-memory data storage
- Jest integration tests

## Frontend Layer

- Static HTML + JavaScript
- Fetch API for communication
- Dynamic UI updates
- Dashboard summary rendering

## Testing Layer

### Integration Testing

- Jest
- Supertest

### End-to-End Testing

- Cypress
- Headless Electron execution

## CI Layer

- GitHub Actions
- Automated workflow trigger
- Service orchestration
- Automated validation

---

# 9. Engineering Justification

The chosen architecture ensures:

- Clear separation of concerns
- Independent backend validation
- Full application validation through E2E tests

- Automated quality assurance
- Reproducible execution
- CI-driven development workflow

This approach simulates a real-world DevOps lifecycle within an academic environment.

---

# 10. Learning Outcomes

Through this implementation, the following competencies were strengthened:

- Full stack architecture understanding
- RESTful API integration
- Automated integration testing
- End-to-end browser automation
- CI pipeline enhancement
- Multi-service orchestration
- Headless test execution
- Deterministic validation strategies

---

# 11. Conclusion

This assignment successfully enhances the CI pipeline by integrating end-to-end testing for the Personal Finance Tracker application.

The final system validates the complete application lifecycle:

```
User Action → UI → API → Backend Processing → Data Update → UI Reflection → Automated Validation
```

The implementation meets all scenario requirements:

✔ Cypress-based E2E testing

✔ CI pipeline extension

✔ Automated service orchestration

✔ Complete application flow validation

✔ Reliable failure detection

This reflects a production-aligned DevOps workflow adapted for academic evaluation.

---