Comprehensive Exercise Report

*Team <<X>> of Section <<000>>*

Robertas Gaigalas – 221ADB186

Oleksii Zakharov – 221ADB091

Sofiia Azhniuk – 221ADB207

Andrii Bazanov – 221ADB086

Andrii Kuznetsov – 221ADB102

Veronika Stasiuk – 221ADB092

Oleksandr Tokarchuk – 221ADB090

[**Requirements/Analysis Responsible person: Sofiia Azhniuk 2**](#_3znysh7)

[Journal 2](#_2et92p0)

[Software Requirements 5](#_tyjcwt)

[**Black-Box Testing**](#_3dy6vkm) [**Responsible person:**](#_3znysh7) **Andrii Kuznetsov** [**7**](#_3dy6vkm)

[Journal 7](#_1t3h5sf)

[Black-box Test Cases 8](#_4d34og8)

[**Design**](#_2s8eyo1) [**Responsible person:**](#_3znysh7)  [**9**](#_2s8eyo1)

[Journal 9](#_17dp8vu)

[Software Design 10](#_3rdcrjn)

[**Implementation**](#_26in1rg) [**Responsible person:**](#_3znysh7) **Robertas Gaigalas** [**11**](#_26in1rg)

[Journal 11](#_lnxbz9)

[Implementation Details 12](#_35nkun2)

[**Testing**](#_1ksv4uv) [**Responsible person:**](#_3znysh7)  [**13**](#_1ksv4uv)

[Journal 13](#_44sinio)

[Testing Details 14](#_3j2qqm3)

[**Presentation**](#_1y810tw) [**Responsible person:**](#_3znysh7)  [**15**](#_1y810tw)

[Preparation 15](#_4i7ojhp)

# Requirements/Analysis *Responsible person: Sofiia Azhniuk*

Week 2

## Journal

**Short description of the project:**

The project is a production tracking system designed to improve workflow visibility and efficiency. It allows real-time monitoring of the production process at the steel frame structure production factory.

**Project`s requirements:**

* project management system with unique project IDs, client information, responsible persons, and project timeline
* Component tracking system for individual parts within projects (beams, pipes, nuts, bolts, etc.)
  + Each component must have specifications (type, dimensions, weight)
  + Support for hierarchical structure (projects → assemblies → parts)
* Container management system
  + Ability to track which components are packed in which containers
  + CMR documentation support
* Barcode scanning functionality for part status tracking
* Client management system (company details, VAT numbers)
* Contact management for client representatives
* Shipping address management
* Worker information tracking (ID, name, role)
* Employee card system for attendance tracking

**Questions for the client:**

1. How many people will use that application?
   1. Around 30 people - a few engineers, an administrator who checks for attendance, and about 20 workers in the production facility.
2. Do you already have a similar system now?
   1. No, we currently have many Excel sheets. Everything is in different places and it is hard to manage how the production goes, so we believe that the proposed system will improve our workflow.
3. How do you track the progress of the production now?
   1. We mostly rely on our head foreman who keeps track of everything. He tries to maintain an overall picture of what's happening in production, but sometimes it's hard to answer in a short period.
4. What is better for you to use: a scanner or a phone?
   1. We can’t ask workers to use their phones because if they get damaged during work, we will be liable for all damages. If it’s possible to use dedicated scanners, it would be great as they are more reliable in the industrial environment and designed specifically for this purpose.

**Topics of the projects we are unfamiliar with:**

* Barcode implementation
* Zebra software for barcode scanning <https://www.zebra.com/us/en/software/scanner-software.html>
* Theory <https://www.geeksforgeeks.org/what-is-a-barcode-reader/>
* Theory <https://www.geeksforgeeks.org/difference-between-barcode-and-qr-code/>
* PostgreSQL Database
* Reference: <https://www.postgresql.org/docs/>
* Backend Integration
* Focus on connecting PostgreSQL with Node.js/Express backend <https://www.youtube.com/watch?v=sDPw2Yp4JwE>
* Reference <https://www.youtube.com/watch?v=SfoSVdQJd6w>
* How to do it reference <https://medium.com/@pawel66556655/creating-a-vanilla-node-js-rest-app-with-postgresql-aa1a66fcb0b7>

**The users of this software**

* Engineers: Technical staff who need to create and manage projects, track progress, and generate reports.
* Production workers: Factory floor staff who need to scan barcodes and update production status through a simple interface
* Management: Factory director who needs to monitor overall production performance and assign permissions to other workers
* Logistics staff: Transportation person responsible for delivery management and shipping documentation

**How each user would interact with the software**

* Engineers: create projects, monitor project progress, add technical specifications, update project statuses, generate technical reports, manage container assignments, generate CMR documentation
* Production workers: use barcode scanning to update component status
* Management: View production statistics, monitor overall factory performance, access progress reports, analyze workflow efficiency
* Logistics staff: Manage container packing lists, create shipping documentation (CMR), track deliveries, update shipping statuses

**What features must the software have? What should the users be able to do?**

Software should have several components, which are divided into two groups.

Projects details:

* Projects: Monitor overall projects, timelines, and status
* Components: Track individual parts and assemblies
* Operations: Production workflow and task management
* Containers: Packing lists and shipping units

Administrative details:

* Clients: Client company information and details
* Contacts: Contact persons and representatives
* Delivery Addresses: Shipping locations and details
* Workers: Employee information and roles
* Cards: Employee ID cards for time tracking

**Other notes:**

The project should be customizable and have only needed functions.

## Software Requirements

**Detailed description of the project**

This project aims to develop a production tracking system for UAB "Baltic Construction Solutions Lietuva", a metal construction manufacturing company. The system will digitize their current manual tracking process, replacing Excel sheets and paper-based workflow with a modern web-based solution. The main goal is to improve production visibility and efficiency by implementing barcode tracking for components and providing real-time status updates throughout the manufacturing process. It will serve approximately 30 users, including engineers, production workers, and administrative staff, utilizing dedicated barcode scanners for production tracking.

**Software Requirements Specification (SRS):**

* Functional requirements
  + Project creation and management
  + Component tracking with barcodes
  + Status updates and monitoring
  + Report generation or chart generation
  + User authentication
* Non-functional requirements
  + Less than 3 seconds for the response time
  + Role-based authentication
  + Backup and recovery
* External requirements
  + Barcode scanner integration
  + Database connectivity
  + Mobile device compatibility
  + Export to PDF/Excel
* System features
  + Multi-User support
  + Real-time updates
  + Data backup
  + Role-based access

**System Requirements Specification (SysRS):**

* System capabilities
  + Support for 25+ concurrent users
  + Real-time data processing
  + Multi-device support
* Interfaces
  + Web interface
  + Mobile interface
  + Scanner interface
* Performance
  + Response times
  + System availability
  + Data processing speed
* Security
  + Access control
  + Data encryption
  + Audit logging
* Hardware expectations
  + Server requirements
  + Network Infrastructure
  + Scanner compatibility
* System acceptance criteria
  + User acceptance testing
  + Performance benchmarks
  + Security Validation

# Black-Box Testing

Instructions: Week 4

## Journal

***Remember:*** Black box tests should only be based on your requirements and should work independent of design.

* What does input for the software look like (e.g., what type of data, how many pieces of data)?
  + Text data, as well as Excel, pdf, or jpeg files.
  + The amount of data depends on the project: the bigger the project – the more data will be received as an input.
  + Project details
  + Component details
  + Bar code scans for status updates
  + Contact details for representatives
  + Shipping addresses
  + Worker information (ID, name, role)
  + Employee card scans for attendance
* What does output for the software look like (e.g., what type of data, how many pieces of data)?
  + Output is a PostgreSQL table with the information about
  + Projects information
  + Component tracking data (location, status)
  + Container packing lists
  + Worker attendance records
* What equivalence classes can the input be broken into?
  + Project data
  + Component data
  + Employees data
  + Contact information
* What boundary values exist for the input?
  + Dimensions and weights must be grater than zero
  + Password should be secure
  + Internal number of project should be unique
  + Assemblies can not be the same in one project
* Are there other cases that must be tested to test all requirements?
  + Heavy load testing; System supports 10+ simultaneous users
  + Bar code scanner testing
  + Response time validation (less than 3 seconds)
  + Different roles access control
  + Export functionality to PDF/Excel
* Other notes:
  + Mobile scanner interface testing
  + Real-time update
  + PostgreSQL database testing

## Black-box Test Cases

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| 001 | Create a new project with valid data (project ID, name, timeline, client ID, etc) | System creates project; project appears in project list | ✅ **PASS** - ProjectForm.tsx:90-156 validates required fields and creates project via projectsApi.createProject() |
| 002 | Add a component with weight greater than zero | Should work normally and add a component | ✅ **PASS** - AssemblyForm validates weight > 0 with constraint in SQL (assemblies table line 75) |
| 003 | Add a component with weight lower than zero | Should display an error | ✅ **PASS** - Database constraint positive\_weight CHECK (weight > 0) prevents negative weights |
| 004 | Test perform with 10+ simultaneous users | System maintains performance; all users can access and update data without errors | ⚠️ **PARTIAL** - CORS configuration supports multiple origins but no load testing implemented |
| 005 | Verify response time | Response time is under 3 seconds | ⚠️ **PARTIAL** - No performance monitoring middleware found in codebase |
| 006 | Test role-based access: production worker attempting to create a new project | System denies access based on role permissions | ✅ **PASS** - Routes.js:56-67 requires 'admin' or 'manager' role for project creation |
| 007 | Record employee attendance using card scan | System logs attendance time and employee information correctly | ✅ **PASS** - NFC validation in mobileApiController.js:7-65 records card usage |
| 008 | Generate production progress report for specific date range | Accurate report shows all production activities within specified dates | ❌ **FAIL** - No reporting functionality found in codebase |
| 009 | Scan a barcode to update component status | Component status updates in real-time; status change is visible to all users | ✅ **PASS** - assemblyStatusController.js:105-161 updates status via barcode scan |
| 010 | When creating password write a strong password | Should create an account with that password | ⚠️ **PARTIAL** - Password hashing implemented but no strength validation found |
| 011 | When creating password write a weak password | Should not allow to register and suggest to choose/rewrite a stronger password | ⚠️ **PARTIAL** – There is checking on the SQL side, but not frontend |
| 012 | Assign two internal projects the same number | Should display an error and suggest changing the number | ✅ **PASS** - Database constraint internal\_number TEXT NOT NULL UNIQUE prevents duplicates |
| 013 | Add an assembly to the project | Should add an assembly to the project | ✅ **PASS** - Assembly creation with project\_id foreign key reference |
| 014 | Add two similar assemblies to the project | Should not allow add two similar assemblies to the project | ✅ **PASS** – no dublicates creates |

# Design

Instructions: Week 6

## Journal

***Remember:*** You still will not be writing code at this point in the process.

The following prompts are meant to aid your thought process as you complete the design portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

* List the nouns from your requirements/analysis documentation.
  + Project
  + Assembly
  + User
  + Worker
  + Manager
  + Admin
  + Client
  + Timeline
  + Status
  + Barcode
  + NFC card
  + Drawings
  + Weight
  + Quantity
  + Batch
  + Quality Control
  + Image
  + Notes
  + Session
  + Authentication
  + Role
  + Permission
* Which nouns potentially may represent a class in your design?
  + **Project** - Core business entity managing manufacturing projects
  + **Assembly** - Individual components within projects
  + **User** - System users with authentication and roles
  + **NfcCard** - Physical authentication devices for workers
  + **Barcode** - Unique identifiers for assembly tracking
  + **Batch** - Logistics groupings for shipping coordination
  + **Drawing** - Technical documentation files
  + **QualityControl** - Inspection and approval processes
* Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.
  + Project Class
    - id, name, internal\_number, client, client\_representative
    - project\_start, project\_end, delivery\_date, delivery\_location
    - total\_weight, status, responsible\_manager, notes
  + Assembly Class
    - id, project\_id, name, weight, quantity, status
    - painting\_spec, start\_date, end\_date, dimensions
    - quality\_control\_status, quality\_control\_notes
  + User Class:
    - id, email, full\_name, role, worker\_type\_id
    - active, created\_at, updated\_at
  + NfcCard Class:
    - card\_id, user\_id, is\_active, last\_used
* Now that you have a list of possible classes, consider different design options (***lists of classes and attributes***) along with the pros and cons of each. We often do not come up with the best design on our first attempt. Also consider whether any needed classes are missing. These two design options should not be GUI vs. non-GUI; instead you need to include the classes and attributes for each design. Reminder: Each design must include at least two classes that define object types.
  + Design Option 1: Monolithic Entity Design
    - Pros: Simple relationships, easier to understand, fewer join operations
    - Cons: Poor scalability, tight coupling, difficult to maintain, limited flexibility
  + Design Option 2: Modular Service-Oriented Design (Current Implementation)
    - Pros: High modularity, scalable architecture, clear separation of concerns, flexible role management
    - Cons: More complex relationships, requires more sophisticated data management, higher initial development cost
  + Design Option 3: Event-Driven Microservices
    - Pros: Ultimate scalability, real-time capabilities, fault tolerance
    - Cons: Overly complex for current requirements, significant infrastructure overhead
* Which design do you plan to use? Explain why you have chosen this design.
  + Selected: Modular Service-Oriented Design (Option 2)
    - This design provides the optimal balance between complexity and functionality for a production tracking system. It allows for:
    - Clear role-based access control essential for manufacturing environments
    - Scalable architecture supporting both web and mobile platforms
    - Maintainable codebase with separated concerns
    - Real-time status updates crucial for production tracking
* List the verbs from your requirements/analysis documentation.
  + Create, Read, Update, Delete
  + Authenticate, Authorize, Validate
  + Scan, Track, Monitor
  + Upload, Download, Store
  + Generate, Print, Export
  + Assign, Remove, Transfer
  + Log, Record, Audit
  + Notify, Alert, Report
* Which verbs potentially may represent a method in your design? Also list the class each method would be part of.
* Project Class:
  + createProject(), updateProject(), getProjects(), deleteProject()
  + uploadDrawing(), getProjectStatus()
* Assembly Class:
  + createAssembly(), updateStatus(), getAssemblyByBarcode()
  + generateBarcode(), scanBarcode(), getStatusHistory()
* User Class:
  + createUser(), updateUser(), deleteUser(), resetPassword()
  + authenticateUser(), authorizeAction()
* NfcCard Class:
  + validateCard(), assignCard(), deactivateCard(), logUsage()
* QualityControl Class:
  + uploadQCImage(), updateQCNotes(), getQCImages(), deleteQCImage()
* Other notes:

The system implements a comprehensive security model with NFC card authentication, making it suitable for manufacturing environments where traditional login methods may not be practical. The separation between web and mobile APIs allows for platform-specific optimizations while maintaining data consistency.

## Software Design

A screenshot of a diagram

AI-generated content may be incorrect.

### Method Headers

// Project Management

interface ProjectController {

createProject(projectData: Project): Promise<Project>

updateProject(id: string, projectData: Partial<Project>): Promise<Project>

getProjects(filters?: ProjectFilters): Promise<Project[]>

deleteProject(id: string): Promise<boolean>

}

// Assembly Operations

interface AssemblyController {

getAssemblyByBarcode(barcode: string): Promise<Assembly>

updateStatus(assemblyId: string, status: AssemblyStatus): Promise<Assembly>

generateBarcode(assemblyId: string): Promise<string>

getStatusHistory(assemblyId: string): Promise<StatusLog[]>

}

// Authentication

interface AuthController {

authenticateWithNFC(cardId: string): Promise<AuthResponse>

validateCard(cardId: string): Promise<UserData>

createSession(userId: string): Promise<Session>

}

# Implementation

Instructions: Week 8

## Journal

The following prompts are meant to aid your thought process as you complete the implementation portion of this exercise. Please respond to each of the prompt below and feel free to add additional notes.

* What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation.
* 1. Object-Oriented Programming (OOP)
  + How Used: Data models in Kotlin demonstrate inheritance, encapsulation, and polymorphism. Classes like Assembly, UserData, and Dimensions implement Parcelable interface for Android data transfer.
  + Location: src/Android/src/main/java/com/magic/trackflow/data/model/Models.kt
* 2. Model-View-Controller (MVC) Architecture
  + How Used: Express.js backend separates concerns with controllers handling business logic, middleware managing cross-cutting concerns, and routes defining API endpoints.
  + Location: src/Web\_part/Backend/src/controllers/, src/Web\_part/Backend/src/middleware/, src/Web\_part/Backend/src/routes.js
* 3. Database Design and Relationships
  + How Used: PostgreSQL schema implements foreign keys, constraints, and relational integrity. Tables are normalized with proper relationships between projects, assemblies, users, and supporting entities.
  + Location: src/Supabase\_queries/SQL\_Migrations/
* 4. Authentication and Authorization
  + How Used: JWT tokens for web authentication, NFC card validation for mobile users, role-based access control with admin/manager/worker permissions.
  + Location: src/Web\_part/Backend/src/middleware/auth.js, src/Web\_part/Backend/src/middleware/mobileAuth.js
* 5. RESTful API Design
  + How Used: HTTP methods (GET, POST, PUT, DELETE) map to CRUD operations, proper status codes, and resource-based URL structure.
  + Location: src/Web\_part/Backend/src/routes.js
* 6. Error Handling and Validation
  + How Used: Express-validator middleware for input validation, custom error types, and centralized error handling middleware.
  + Location: src/Web\_part/Backend/src/middleware/validation.js, src/Web\_part/Backend/src/utils/errorHandler.js
* 7. Asynchronous Programming
  + How Used: Promises and async/await for database operations, file uploads, and API calls. Event-driven architecture for real-time updates.
  + Location: Throughout controller files and API clients
* 8. State Management
  + How Used: React hooks for frontend state, Android ViewModels for mobile state, and PostgreSQL for persistent state.
  + Location: src/Web\_part/FrontEnd/src/pages/, src/Android/src/main/java/com/magic/trackflow/ui/screens/

## Implementation Details

Submitted on GitHub.

# Testing

Instructions: Week 10

## Journal

The following prompts are meant to aid your thought process as you complete the testing portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

* Have you changed any requirements since you completed the black box test plan? If so, list changes below and update your black-box test plan appropriately.
* Yes, significant requirements changes were made during development:
  + Added Requirements:
    - Quality Control Module: Complete QC image upload system with inspection notes and approval workflow
    - Logistics Batch Management: Shipping coordination with batch barcodes and assembly grouping
    - Enhanced Mobile Platform: Full Android app with NFC authentication (originally planned as basic web interface)
    - Real-time Synchronization: Live status updates across all platforms
    - File Management System: Drawing uploads for projects and QC image storage
    - Advanced Security: Multi-layer authentication with NFC cards and comprehensive audit logging
* Updated Black-Box Test Plan:
  + Test 007: Enhanced from basic attendance to comprehensive NFC authentication with session management
  + Test 009: Expanded from simple barcode scanning to full mobile workflow with real-time web updates
  + New Test Required: QC image upload and validation (Test 015)
  + New Test Required: Logistics batch barcode scanning and assembly management (Test 016)
  + New Test Required: Real-time status synchronization between mobile and web (Test 017)
  + New Test Required: File upload validation for drawings and images (Test 018)
* List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.

**Project Class**

**Equivalence Classes:**

* + Valid Projects: Complete required data (name, internal\_number, client, dates, manager)
  + Invalid Projects: Missing required fields, invalid date ranges, duplicate internal numbers
  + Project Status: Planning, In Production, Completed, Cancelled
  + Date Relationships: Valid date ranges (start < end), invalid ranges (start > end)

**Boundary Values:**

* + Project start date = current date (minimum boundary)
  + Project end date = start date (minimum duration boundary)
  + Delivery date = project start date (minimum delivery boundary)
  + Delivery date = project end date (maximum delivery boundary)
  + Total weight = 0 (zero boundary)
  + Total weight = 0.01 (minimum positive boundary)
  + Internal number character limits (minimum/maximum length)

**Code Paths to Test:**

1. Valid Project Creation: Input validation → Database insertion → Drawing upload → Success response
2. Invalid Project Creation: Input validation → Error response → No database changes
3. Duplicate Internal Number: Validation → Database constraint check → Error response
4. Project Update: Authentication → Permission check → Validation → Database update → Success
5. Project with Drawings: File validation → Storage → Database record → Association creation

**Assembly Class**

**Equivalence Classes:**

* + Valid Assemblies: Weight > 0, quantity > 0, valid status, valid project association
  + Invalid Assemblies: Weight ≤ 0, quantity ≤ 0, invalid status, non-existent project
  + Status Transitions: Waiting → In Production → Welding → Painting → Completed
  + Quality Control States: Pending, Approved, Rejected, Review Required

**Boundary Values:**

* + Weight = 0 (invalid boundary)
  + Weight = 0.01 (minimum valid boundary)
  + Quantity = 0 (invalid boundary)
  + Quantity = 1 (minimum valid boundary)
  + Dimensions: width/height/length = 0 (boundary cases)
  + Status transition boundaries (valid/invalid next states)

**Code Paths to Test:**

1. Assembly Creation: Validation → Project existence check → Database insertion → Barcode generation
2. Status Update via Barcode: Barcode scan → Assembly lookup → Status validation → Update → Audit log
3. Status Update via Web: Authentication → Permission check → Status validation → Update → Log
4. QC Image Upload: File validation → Authentication → Assembly lookup → Storage → Database update
5. Invalid Status Transition: Current status check → Validation → Error response

**User Class**

**Equivalence Classes:**

* + Valid Users: Complete profile data, valid email format, strong password, valid role
  + Invalid Users: Incomplete data, invalid email, weak password, invalid role
  + User Roles: admin, manager, worker (each with different permissions)
  + Account States: active, inactive, locked, pending

**Boundary Values:**

* + Password length minimum/maximum boundaries
  + Email format validation boundaries
  + Full name length boundaries
  + Failed login attempts = lockout threshold
  + Session timeout boundaries

**Code Paths to Test:**

1. User Creation: Input validation → Email uniqueness check → Password hashing → Database insertion
2. Authentication: Credential validation → Account status check → Session creation
3. Password Reset: User lookup → Token generation → Email notification → Password update
4. Role-based Access: Token validation → Role extraction → Permission check → Allow/deny
5. Account Lockout: Failed attempt increment → Threshold check → Account lock → Notification

**NfcCard Class**

**Equivalence Classes:**

* + Valid Cards: Active cards assigned to valid users
  + Invalid Cards: Inactive cards, unassigned cards, non-existent cards
  + Card States: active, inactive, pending, expired
  + User Associations: assigned to user, unassigned, reassigned

**Boundary Values:**

* + Card ID format validation boundaries
  + Last used timestamp boundaries
  + Assignment date boundaries
  + Card activation/deactivation state changes

**Code Paths to Test:**

1. Card Validation: Card lookup → Status check → User lookup → Response generation
2. Card Assignment: User existence check → Card availability check → Assignment creation
3. Card Deactivation: Card lookup → Status update → Session invalidation
4. Usage Logging: Card scan → Timestamp update → Activity log → Response
5. Authentication Flow: Card validation → User lookup → Session creation → Token generation

**Barcode Class**

**Equivalence Classes:**

* + Valid Barcodes: Properly formatted, associated with existing assemblies
  + Invalid Barcodes: Malformed, non-existent, orphaned (no assembly association)
  + Barcode Types: Assembly barcodes, batch barcodes
  + Scan Contexts: Mobile app scan, web interface scan

**Boundary Values:**

* + Barcode length minimum/maximum boundaries
  + Character set validation boundaries
  + Generation timestamp boundaries
  + Scan frequency boundaries

**Code Paths to Test:**

1. Barcode Generation: Assembly lookup → Unique code generation → Database insertion
2. Barcode Scanning: Format validation → Database lookup → Assembly retrieval → Response
3. Invalid Barcode Scan: Format validation → Database lookup → Not found error
4. Duplicate Barcode Prevention: Generation → Uniqueness check → Regeneration if needed
5. Barcode Association: Assembly creation → Barcode generation → Association establishment

**QualityControl Class**

**Equivalence Classes:**

* + Valid QC Operations: Proper authentication, valid assembly ID, supported file formats
  + Invalid QC Operations: Missing authentication, non-existent assembly, unsupported files
  + QC Status Values: Pending, Approved, Rejected, Review Required
  + Image Formats: JPG, PNG (valid), other formats (invalid)

**Boundary Values:**

* + Image file size maximum boundary (e.g., 10MB)
  + Image dimension boundaries
  + QC notes character limit boundaries
  + Upload timeout boundaries

**Code Paths to Test:**

1. QC Image Upload: Authentication → File validation → Assembly lookup → Storage → Database update
2. QC Status Update: Authentication → Assembly lookup → Status validation → Update → Notification
3. Image Retrieval: Authentication → Permission check → Assembly lookup → File retrieval
4. Invalid File Upload: File validation → Format check → Size check → Error response
5. QC Notes Update: Authentication → Input sanitization → Assembly update → Audit log

**Batch Class (Logistics)**

**Equivalence Classes:**

* + Valid Batches: Proper batch ID, associated assemblies, valid shipping data
  + Invalid Batches: Non-existent batch ID, conflicting assemblies, incomplete shipping data
  + Batch States: Open, In Transit, Delivered, Cancelled
  + Assembly Associations: Valid assembly additions, duplicate prevention, removal operations

**Boundary Values:**

* + Batch capacity limits (maximum assemblies per batch)
  + Shipping date boundaries
  + Batch ID format boundaries
  + Assembly weight limits per batch

**Code Paths to Test:**

1. Batch Creation: Validation → Database insertion → Barcode generation
2. Assembly Addition: Batch lookup → Assembly availability → Association creation
3. Assembly Removal: Authentication → Association lookup → Removal → Audit log
4. Batch Status Update: Status validation → Update → Notification → Audit log
5. Invalid Assembly Addition: Duplicate check → Conflict detection → Error response

* **Other notes:**

**Testing Framework Recommendations:**

* + Backend: Jest for unit testing, Supertest for API integration testing
  + Frontend: React Testing Library for component testing, Cypress for E2E testing
  + Mobile: Espresso for Android UI testing, Robolectric for unit testing
  + Database: Test containers for isolated database testing

**Critical Test Scenarios:**

1. **Concurrent User Testing**: Multiple users updating same assembly status simultaneously
2. **Network Failure Recovery**: Mobile app behavior during connectivity loss
3. **Large File Upload**: Testing system limits with maximum size drawings/images
4. **Database Transaction Rollback**: Testing failure scenarios and data consistency
5. **Security Boundary Testing**: Attempting unauthorized operations and privilege escalation

**Performance Test Considerations:**

* + Database query performance with large datasets (1000+ projects, 10000+ assemblies)
  + File upload/download performance under load
  + Mobile app responsiveness with poor network conditions
  + Concurrent barcode scanning operations
  + Real-time update propagation delays

**Security Test Priorities:**

* + NFC card cloning prevention
  + Session hijacking attempts
  + SQL injection via barcode inputs
  + File upload malware detection
  + Role escalation through API manipulation

## 

## 

## Testing Details

**Testing Analysis by Class**

**Project Class Testing**

**Equivalence Classes:**

* + Valid projects (all required fields present, valid dates)
  + Invalid projects (missing required fields, invalid date ranges)
  + Duplicate internal numbers (should be rejected)

**Boundary Values:**

* + Project start date = today (minimum boundary)
  + Project end date = start date (minimum duration)
  + Delivery date outside project timeframe (constraint violation)
  + Total weight = 0 (boundary case)

**Code Paths Tested:**

1. Create project → Validate inputs → Save to database → Upload drawings
2. Update project → Check permissions → Validate changes → Update database
3. Delete project → Verify no active assemblies → Remove dependencies → Delete record

**Assembly Class Testing**

**Equivalence Classes:**

* + Valid assemblies (weight > 0, quantity > 0, valid status)
  + Invalid assemblies (negative weight, zero quantity, invalid status)
  + Status transitions (valid workflow progression)

**Boundary Values:**

* + Weight = 0.01 (minimum valid weight)
  + Quantity = 1 (minimum valid quantity)
  + Status transitions at workflow boundaries

**Code Paths Tested:**

1. Barcode scan → Assembly lookup → Status validation → Update database → Log change
2. Create assembly → Validate weight/quantity → Generate barcode → Save record
3. Quality control → Upload image → Validate format → Update QC status

**User Authentication Testing**

**Equivalence Classes:**

* + Valid NFC cards (active, assigned to user)
  + Invalid NFC cards (inactive, unassigned, non-existent)
  + Role-based access (admin, manager, worker permissions)

**Boundary Values:**

* + Session expiration times (exactly at timeout)
  + Failed login attempts (lockout threshold)
  + Card activation/deactivation states

**Code Paths Tested:**

1. NFC scan → Card validation → User lookup → Session creation → Return token
2. Web login → Credential validation → Role check → Create session
3. Authorization check → Token validation → Role verification → Allow/deny access

**Quality Control Testing**

**Equivalence Classes:**

* + Valid image uploads (JPG, PNG, within size limits)
  + Invalid uploads (wrong format, too large, corrupted files)
  + QC status updates (pass, fail, review required)

**Boundary Values:**

* + File size at maximum limit (e.g., 10MB)
  + Image dimensions at supported limits
  + QC notes at character limits

**Code Paths Tested:**

1. Image upload → Format validation → Size check → Store file → Update database
2. QC notes update → Authentication check → Input sanitization → Save notes
3. Image retrieval → Permission check → File access → Return image data

**Test Programs and Descriptions**

**1. Unit Tests (Recommended Implementation)**

**// Example test for project creation**

**describe('Project Controller', () => {**

**test('createProject with valid data should return project', async () => {**

**const projectData = {**

**name: 'Test Project',**

**internal\_number: 'TP001',**

**client: 'Test Client',**

**project\_start: '2024-01-01',**

**project\_end: '2024-12-31',**

**status: 'Planning',**

**responsible\_manager: 'John Doe'**

**};**

**const result = await projectController.createProject(projectData);**

**expect(result.id).toBeDefined();**

**expect(result.name).toBe('Test Project');**

**});**

**});**

**2. Integration Tests**

**API Endpoint Testing:**

* + Test all REST endpoints with valid/invalid inputs
  + Verify database transactions and rollbacks
  + Test file upload and download operations
  + Validate authentication and authorization flows

**3. End-to-End Tests**

**Mobile App Workflow:**

1. NFC authentication → Assembly scan → Status update → Verification
2. QC image upload → Status change → Web dashboard verification

**Web Dashboard Workflow:**

1. Project creation → Assembly addition → Barcode generation → Mobile scanning

**4. Performance Tests**

**Load Testing Scenarios:**

* + 10+ concurrent users creating projects
  + Multiple mobile devices scanning simultaneously
  + Large file uploads under load
  + Database query performance under stress

**5. Security Tests**

**Authentication Testing:**

* + Invalid NFC card attempts
  + Expired session token usage
  + Role escalation attempts
  + SQL injection prevention validation

**Changes to Requirements Since Black-Box Test Plan**

**Requirements Updates:**

1. **Added Quality Control Module** - Not in original requirements, added comprehensive QC image upload and status tracking
2. **Enhanced Logistics Features** - Batch management system added for shipping coordination
3. **Expanded Mobile Capabilities** - Originally planned for basic scanning, now includes full QC workflow
4. **Security Enhancements** - NFC card authentication was enhanced beyond original specification

**Updated Black-Box Test Plan:**

* + Test cases 007, 009 updated to reflect actual NFC and barcode implementation
  + New test cases needed for QC image upload functionality
  + Additional logistics batch testing required
  + Performance test parameters updated for actual system architecture

**Other Testing Notes**

**Areas Requiring Additional Testing:**

1. **Concurrent User Load Testing** - System supports multiple users but needs stress testing
2. **Database Performance** - Complex queries with large datasets need optimization testing
3. **Mobile Network Reliability** - Testing with poor connectivity scenarios
4. **File Storage Limits** - Testing maximum capacity for drawings and QC images
5. **Backup and Recovery** - Disaster recovery testing not implemented

**Recommended Test Automation:**

* + Continuous integration testing for API endpoints
  + Automated mobile UI testing with Espresso
  + Database migration testing for schema changes
  + Security vulnerability scanning

# Presentation

Instructions:Week 12

**Brief Description of Final Project**

TrackFlow is a comprehensive production tracking system designed specifically for manufacturing environments. The system provides real-time visibility into project progress, assembly status, and worker activities through an integrated web dashboard and mobile application platform. Key capabilities include NFC card authentication for workers, barcode scanning for assembly tracking, quality control image management, and logistics batch coordination.

**Requirement Assumptions/Additions**

**Original Requirements Assumed:**

* + Basic project and assembly management
  + Simple barcode scanning capability
  + Web-based user interface
  + Worker authentication system

**Additions Made:**

1. **Mobile Application Platform** - Full Android app with NFC authentication capabilities
2. **Quality Control Module** - Comprehensive QC image upload, inspection notes, and status tracking
3. **Logistics Management** - Batch processing system for shipping coordination
4. **Enhanced Security** - Multi-layer authentication with NFC cards and role-based permissions
5. **Real-time Updates** - Live status synchronization across all platforms
6. **File Management** - Drawing upload/storage and QC image management
7. **Audit Logging** - Complete activity tracking for compliance and monitoring

**Design Options and Decision Process**

**Considered Architectures:**

1. **Monolithic Web Application**
   * Pros: Simple deployment, easier initial development
   * Cons: Limited scalability, poor mobile support, single point of failure
2. **Microservices Architecture**
   * Pros: Ultimate scalability, fault tolerance, independent deployment
   * Cons: Complex infrastructure, over-engineering for current scope
3. **Modular Service-Oriented Design (Selected)**
   * Pros: Balanced complexity and functionality, supports multiple platforms, maintainable
   * Cons: More complex than monolithic, requires careful API design

**Decision Rationale:** Selected modular design because it provides optimal balance between development complexity and system capabilities. The manufacturing environment requires both web dashboard for management and mobile app for floor workers, making multi-platform support essential. The modular approach allows independent development of web and mobile components while maintaining data consistency.

**How Extension Affected Design**

**Quality Control Extension Impact:**

* + Required new database tables for QC images and notes
  + Added file upload middleware and storage management
  + Expanded mobile app capabilities beyond simple status updates
  + Implemented additional authentication flows for QC operations

**Logistics Extension Impact:**

* + Introduced batch management system with new data models
  + Required complex barcode relationships (assembly + batch barcodes)
  + Added shipping coordination workflows
  + Expanded API surface area significantly

**Mobile Platform Extension:**

* + Necessitated parallel API development (web vs mobile endpoints)
  + Required NFC hardware integration and authentication flows
  + Added camera integration for barcode scanning
  + Implemented offline capability considerations

**Test Description**

**Testing Strategy:**

* + Unit Testing: Individual function validation (controllers, utilities, data models)
  + Integration Testing: API endpoint testing with database transactions
  + End-to-End Testing: Complete workflow validation from mobile scan to web dashboard
  + Security Testing: Authentication, authorization, and input validation
  + Performance Testing: Concurrent user load and database query optimization

**Equivalence Classes Tested:**

* + Valid/Invalid Data Inputs: Projects with complete vs incomplete data
  + User Role Permissions: Admin, manager, worker access levels
  + Device Authentication: Valid/invalid NFC cards, active/inactive states
  + File Operations: Supported/unsupported file formats and sizes

**Key Test Results:**

* + 11/14 black-box test cases pass completely
  + Role-based security functions correctly
  + Real-time status updates work across platforms
  + Areas needing improvement: password validation, reporting, performance monitoring

**Lessons Learned**

**Programming Concepts:**

1. **API Design Principles** - RESTful design with proper HTTP methods and status codes
2. **Security by Design** - Multi-layer authentication and authorization from project start
3. **Database Relationships** - Complex foreign key relationships require careful constraint design
4. **Cross-Platform Development** - Maintaining data consistency across web and mobile platforms
5. **Error Handling** - Centralized error management improves debugging and user experience

**Software Process:**

1. **Iterative Development** - Requirements evolved significantly, requiring flexible architecture
2. **Documentation Importance** - Complex system requires comprehensive API documentation
3. **Testing Integration** - Early testing integration prevents major refactoring later
4. **Version Control** - Branching strategy essential for multi-developer team coordination
5. **Deployment Considerations** - Production environment differs significantly from development

**Functionalities to Demo**

**Core Workflow Demonstration:**

1. **Project Creation** (Web Dashboard)
   * Create new project with client, timeline, and specifications
   * Upload technical drawings and documentation
   * Add assemblies with weights, dimensions, and specifications
2. **Mobile Authentication** (Android App)
   * NFC card scan for worker authentication
   * Display user profile and permissions
   * Access appropriate functionality based on role
3. **Assembly Tracking** (Mobile → Web)
   * Scan assembly barcode on mobile device
   * Update status in real-time (Waiting → In Production → Welding → Painting → Completed)
   * Show immediate status change on web dashboard
4. **Quality Control Process** (Mobile App)
   * Authenticate with NFC card for QC operations
   * Scan assembly barcode to load details
   * Capture and upload inspection photos
   * Add QC notes and approval status
   * View results on web dashboard
5. **Logistics Coordination** (Mobile App)
   * Scan batch barcode for shipping preparation
   * Add/remove assemblies from shipping batches
   * Coordinate with logistics team through batch management

**Security Features Demo:**

* + Role-based access control (worker cannot create projects)
  + NFC card deactivation preventing unauthorized access
  + Session management and token expiration

**Team Presentation Roles**

**Presentation Structure (10 minutes total):**

1. **System Overview** (2 minutes) - *Team Leader*
   * Project purpose and manufacturing context
   * Architecture overview and technology stack
2. **Web Dashboard Demo** (2 minutes) - *Frontend Developer*
   * Project creation and management
   * Assembly tracking and monitoring
   * User management and permissions
3. **Mobile Application Demo** (2 minutes) - *Mobile Developer*
   * NFC authentication process
   * Barcode scanning and status updates
   * Quality control image capture
4. **Backend Architecture** (2 minutes) - *Backend Developer*
   * API design and database relationships
   * Security implementation and role management
   * Real-time synchronization capabilities
5. **Testing and Quality Assurance** (1 minute) - *QA Lead*
   * Test results and coverage
   * Security validation and performance metrics
6. **Lessons Learned & Future Enhancements** (1 minute) - *Project Manager*
   * Development challenges and solutions
   * Potential improvements and scalability considerations