

Week 4 Project Overview

Artifacts:

- Server Code with DAO using Sqlite using Spring Boot Data JPA with Hibernate
- React Client for interacting with the API
- Screen Shot of the React Client
- UML Diagram of the Application

```
=====
FILE: src\main\java\com\cu5448\pcb\PcbApplication.java
=====
```

```
package com.cu5448.pcb;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.scheduling.annotation.EnableAsync;

/**
 * Main Spring Boot Application with Configuration Properties Support
 *
 * <p>Demonstrates Dependency Injection design pattern implementation:
 * - @EnableConfigurationProperties enables property-driven configuration - REST API endpoints
 * available for running simulations on demand - All dependencies managed by Spring IoC container
 * - @EnableAsync enables asynchronous method execution for simulation processing
 *
 * <p>Server starts and waits for client API calls to run simulations and retrieve results.
 */
@SpringBootApplication
@EnableConfigurationProperties
@EnableAsync
public class PcbApplication {

    public static void main(String[] args) {
        SpringApplication.run(PcbApplication.class, args);
    }
}
```

```
=====
FILE: src\main\java\com\cu5448\pcb\api\SimulationApiController.java
=====
```

```
package com.cu5448.pcb.api;

import java.util.ArrayList;
import java.util.List;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import com.cu5448.pcb.controller.SimulationController;
import com.cu5448.pcb.dto.SimulationReportDto;
import com.cu5448.pcb.dto.SimulationReportMapper;
import com.cu5448.pcb.entity.SimulationResult;

import lombok.RequiredArgsConstructor;

/**
 * REST API Controller for PCB simulation operations. Implements the server-side requirement: run
 * simulations to gather failure results and persist them to SQLite database, then wait for client
 * API calls to retrieve the stored simulation results as JSON.
 *
 * <p>Provides separate endpoints for running simulations and retrieving persisted results from
 * database.
 */
@RestController
@RequestMapping("/api/simulation")
@CrossOrigin(origins = "http://localhost:3000")
@RequiredArgsConstructor
public class SimulationApiController {

    private final SimulationController simulationController;
    private final SimulationReportMapper reportMapper;

    /**
     * Start simulation for a specific PCB type asynchronously and store results in memory.
     *
     * @param pcbType the type of PCB to simulate (test, sensor, gateway)
     * @param quantity number of PCBs to process (default: 1000)
     * @return 201 Created status to indicate simulation started
     */
    @PostMapping("/run/{pcbType}")
    public ResponseEntity<String> runSimulation(
        @PathVariable String pcbType, @RequestParam(defaultValue = "1000") int quantity) {
```

```

// Start simulation asynchronously using @Async method
simulationController.runSimulationAsync(pcbType, quantity);

return ResponseEntity.status(201)
    .body(
        String.format(
            "Simulation started asynchronously for PCB type: %s with quantity: %d",
            pcbType, quantity));
}

/**
 * Start simulations for all three PCB types asynchronously and store results in memory.
 *
 * @param quantity number of PCBs to process for each type (default: 1000)
 * @return 201 Created status to indicate simulations started
 */
@PostMapping("/run/all")
public ResponseEntity<String> runAllSimulations(
    @RequestParam(defaultValue = "1000") int quantity) {

    // Start all simulations asynchronously using @Async method
    simulationController.runAllSimulationsAsync();

    return ResponseEntity.status(201)
        .body(
            String.format(
                "Simulations started asynchronously for all PCB types with quantity: %d",
                quantity));
}

/**
 * Retrieve stored simulation results for a specific PCB type.
 *
 * @param pcbType the type of PCB to get results for
 * @return simulation report as JSON, or 404 if not found
 */
@GetMapping("/results/{pcbType}")
public ResponseEntity<SimulationReportDto> getSimulationResults(@PathVariable String pcbType) {
    SimulationResult result = simulationController.getSimulationResults(pcbType);

    if (result == null) {
        return ResponseEntity.notFound().build();
    }

    SimulationReportDto report = reportMapper.toDto(result);
    return ResponseEntity.ok(report);
}

/**
 * Retrieve all stored simulation results (latest for each PCB type).
 *
 * @return list of simulation reports for all stored results
 */
@GetMapping("/results/all")
public ResponseEntity<List<SimulationReportDto>> getAllSimulationResults() {
    List<SimulationResult> allResults = simulationController.getAllSimulationResults();

    if (allResults.isEmpty()) {
        return ResponseEntity.notFound().build();
    }

    List<SimulationReportDto> reports = new ArrayList<>();
    for (SimulationResult result : allResults) {
        SimulationReportDto report = reportMapper.toDto(result);
        reports.add(report);
    }

    return ResponseEntity.ok(reports);
}

/**
 * Get available PCB types.
 *
 * @return list of supported PCB types
 */
@GetMapping("/types")
public ResponseEntity<List<String>> getPcbTypes() {
    List<String> types = List.of("Test Board", "Sensor Board", "Gateway Board");
    return ResponseEntity.ok(types);
}

/**
 * Clear all stored simulation results.
 *
 * @return success response
 */
@DeleteMapping("/results")
public ResponseEntity<String> clearAllResults() {
    simulationController.clearAllResults();
}

```

File - C:\Users\lck\Documents\dev\source\CSCA\csca-java\5448\pcb\w4-code.txt

```
        return ResponseEntity.ok("All simulation results cleared");
    }
}

=====
FILE: src\main\java\com\cu5448\pcb\config\CorsConfig.java
=====

package com.cu5448.pcb.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

/**
 * CORS configuration to allow React client (localhost:3000) to communicate with the Spring Boot
 * server (localhost:8080).
 */
@Configuration
public class CorsConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**")
            .allowedOrigins("http://localhost:3000")
            .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
            .allowedHeaders("*")
            .allowCredentials(true);
    }

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.addAllowedOrigin("http://localhost:3000");
        configuration.addAllowedMethod("*");
        configuration.addAllowedHeader("*");
        configuration.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/api/**", configuration);
        return source;
    }
}

=====
FILE: src\main\java\com\cu5448\pcb\config\PCBProperties.java
=====

package com.cu5448.pcb.config;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import com.cu5448.pcb.model.DefectRates;

import lombok.Data;

/**
 * PCB Configuration Properties that directly create DefectRates instances. This approach eliminates
 * nested property classes and provides direct access to DefectRates objects for each PCB type.
 */
@Data
@Component
@ConfigurationProperties(prefix = "pcb")
public class PCBProperties {

    private DefectRates testboard = new DefectRates();

    private DefectRates sensorboard = new DefectRates();

    private DefectRates gatewayboard = new DefectRates();
}

=====
FILE: src\main\java\com\cu5448\pcb\config\PCBSimulationConfig.java
=====

package com.cu5448.pcb.config;
```

File - C:\Users\lck\Documents\dev\source\CSCA\csc-java\5448\pcb\lw4-code.txt

```
import java.util.List;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.cu5448.pcb.factory.StationFactory;
import com.cu5448.pcb.station.*;

import lombok.RequiredArgsConstructor;

/**
 * Spring Configuration Class using Abstract Factory Pattern for PCB Assembly Line Stations. The
 * configuration delegates station creation to a StationFactory, maintaining the factory pattern
 * while leveraging Spring's dependency injection.
 */
@Configuration
@RequiredArgsConstructor
public class PCBSimulationConfig {

    private final StationFactory stationFactory;

    /**
     * Creates ordered list of stations for the assembly line using the abstract factory pattern.
     * This ensures consistent station creation and proper manufacturing process flow.
     */
    @Bean
    public List<Station> createAssemblyLineStations() {
        return List.of(
            stationFactory.createStation("ApplySolderPaste"),
            stationFactory.createStation("PlaceComponents"),
            stationFactory.createStation("ReflowSolder"),
            stationFactory.createStation("OpticalInspection"),
            stationFactory.createStation("HandSoldering"),
            stationFactory.createStation("Cleaning"),
            stationFactory.createStation("Depanelization"),
            stationFactory.createStation("Test"));
    }
}

=====
FILE: src\main\java\com\cu5448\pcb\config\StationProperties.java
=====

package com.cu5448.pcb.config;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import lombok.Data;

/** Station Configuration Properties using Lombok @Data generates all necessary boilerplate code */
@Data
@Component
@ConfigurationProperties(prefix = "station")
public class StationProperties {

    private double failureRate = 0.002;
}

=====
FILE: src\main\java\com\cu5448\pcb\controller\SimulationController.java
=====

package com.cu5448.pcb.controller;

import java.util.List;
import java.util.concurrent.CompletableFuture;

import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Component;

import com.cu5448.pcb.entity.SimulationResult;
import com.cu5448.pcb.repository.SimulationResultRepository;
import com.cu5448.pcb.service.AssemblyLine;
import com.cu5448.pcb.service.StatisticsCollector;

import lombok.RequiredArgsConstructor;

/**
 * Main Simulation Controller using Spring Dependency Injection and Lombok @RequiredArgsConstructor
 * generates constructor for final fields. This controller orchestrates PCB simulations and manages
 * results storage in SQLite database via JPA for REST API access.
 *
 * <p>Supports the server-side requirement: run simulations to gather failure results and persist
 * them to database, then wait for client API calls to retrieve the stored simulation results.

```

```

*/
@Component
@RequiredArgsConstructor
public class SimulationController {

    private final AssemblyLine assemblyLine;
    private final SimulationResultRepository simulationResultRepository;

    /**
     * Run simulation for a specific PCB type asynchronously and persist results to database.
     *
     * @param pcbType the type of PCB to simulate
     * @param quantity number of PCBs to process
     * @return CompletableFuture with the simulation result entity
     */
    @Async
    public CompletableFuture<SimulationResult> runSimulationAsync(String pcbType, int quantity) {
        StatisticsCollector stats = assemblyLine.runSimulation(pcbType, quantity);
        SimulationResult result = createAndSaveSimulationResult(stats, pcbType, quantity);
        return CompletableFuture.completedFuture(result);
    }

    /**
     * Run simulation for a specific PCB type synchronously and persist results to database.
     *
     * @param pcbType the type of PCB to simulate
     * @param quantity number of PCBs to process
     * @return the simulation result entity
     */
    public SimulationResult runSimulation(String pcbType, int quantity) {
        StatisticsCollector stats = assemblyLine.runSimulation(pcbType, quantity);
        return createAndSaveSimulationResult(stats, pcbType, quantity);
    }

    /**
     * Run simulation for a specific PCB type with default quantity (1000).
     *
     * @param pcbType the type of PCB to simulate
     * @return the simulation result entity
     */
    public SimulationResult runSimulation(String pcbType) {
        return runSimulation(pcbType, 1000);
    }

    /**
     * Helper method to create and save SimulationResult from StatisticsCollector.
     *
     * @param stats the statistics collector with simulation data
     * @param pcbType the PCB type that was simulated
     * @param quantity the quantity of PCBs processed
     * @return the saved simulation result entity
     */
    private SimulationResult createAndSaveSimulationResult(
        StatisticsCollector stats, String pcbType, int quantity) {
        SimulationResult result =
            new SimulationResult(
                pcbType,
                stats.getPcbSubmitted(),
                stats.getCompletedPCBs(),
                stats.getStationFailures(),
                stats.getDefectFailures(),
                stats.generateReport(pcbType),
                quantity);
        return simulationResultRepository.save(result);
    }

    /**
     * Run simulations for all three PCB types asynchronously and persist results to database.
     *
     * @return CompletableFuture that completes when all simulations are done
     */
    @Async
    public CompletableFuture<List<SimulationResult>> runAllSimulationsAsync() {
        SimulationResult testResult = runSimulation("test");
        SimulationResult sensorResult = runSimulation("sensor");
        SimulationResult gatewayResult = runSimulation("gateway");
        return CompletableFuture.completedFuture(List.of(testResult, sensorResult, gatewayResult));
    }

    /**
     * Retrieve the most recent simulation result for a specific PCB type.
     *
     * @param pcbType the type of PCB to get results for
     * @return the most recent simulation result, or null if not found
     */
    public SimulationResult getSimulationResults(String pcbType) {
        return simulationResultRepository
            .findFirstByPcbTypeOrderByCreatedAtDesc(pcbType)
            .orElse(null);
    }

```

```

    }

    /**
     * Retrieve the most recent simulation result for each PCB type.
     *
     * @return list of the latest simulation results for each PCB type
     */
    public List<SimulationResult> getAllSimulationResults() {
        return simulationResultRepository.findLatestResultForEachPcbType();
    }

    /** Clear all stored simulation results from database. */
    public void clearAllResults() {
        simulationResultRepository.deleteAll();
    }
}

=====
FILE: src\main\java\com\cu5448\pcb\dto\SimulationReportDto.java
=====

package com.cu5448.pcb.dto;

import java.util.Map;

import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * Data Transfer Object for simulation report data used in REST API communication between server and
 * client applications.
 */
@Data
@NoArgsConstructor
@AllArgsConstructor
public class SimulationReportDto {

    @JsonProperty("pcbType")
    private String pcbType;

    @JsonProperty("pcbsRun")
    private int pcbsRun;

    @JsonProperty("stationFailures")
    private Map<String, Integer> stationFailures;

    @JsonProperty("defectFailures")
    private Map<String, Integer> defectFailures;

    @JsonProperty("totalFailedPcbs")
    private int totalFailedPcbs;

    @JsonProperty("totalPcbsProduced")
    private int totalPcbsProduced;

    @JsonProperty("formattedReport")
    private String formattedReport;
}

=====
FILE: src\main\java\com\cu5448\pcb\dto\SimulationReportMapper.java
=====

package com.cu5448.pcb.dto;

import org.springframework.stereotype.Component;

import com.cu5448.pcb.entity.SimulationResult;
import com.cu5448.pcb.service.StatisticsCollector;

/**
 * Mapper class to convert SimulationResult entities and StatisticsCollector data to
 * SimulationReportDto for API communication.
 */
@Component
public class SimulationReportMapper {

    /**
     * Convert SimulationResult entity to DTO.
     *
     * @param result the simulation result entity
     * @return the DTO for API response
     */
}

```

```

    */
    public SimulationReportDto toDto(SimulationResult result) {
        SimulationReportDto dto = new SimulationReportDto();

        dto.setPcbType(result.getPcbType());
        dto.setPcbsRun(result.getPcbsSubmitted());
        dto.setStationFailures(result.getStationFailures());
        dto.setDefectFailures(result.getDefectFailures());
        dto.setTotalFailedPcbs(result.getTotalFailedPcbs());
        dto.setTotalPcbsProduced(result.getPcbsCompleted());
        dto.setFormattedReport(result.getFormattedReport());

        return dto;
    }

    /**
     * Convert StatisticsCollector data to DTO (for backward compatibility).
     *
     * @param stats the statistics collector
     * @param pcbType the PCB type
     * @return the DTO for API response
     */
    public SimulationReportDto toDto(StatisticsCollector stats, String pcbType) {
        SimulationReportDto dto = new SimulationReportDto();

        dto.setPcbType(pcbType);
        dto.setPcbsRun(stats.getPcbsSubmitted());
        dto.setStationFailures(stats.getStationFailures());
        dto.setDefectFailures(stats.getDefectFailures());
        dto.setTotalFailedPcbs(stats.getPcbsSubmitted() - stats.getCompletedPCBs());
        dto.setTotalPcbsProduced(stats.getCompletedPCBs());
        dto.setFormattedReport(stats.generateReport(pcbType));

        return dto;
    }
}

=====
FILE: src\main\java\com\cu5448\pcb\entity\MapToJsonConverter.java
=====

package com.cu5448.pcb.entity;

import java.util.HashMap;
import java.util.Map;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;

import jakarta.persistence.AttributeConverter;
import jakarta.persistence.Converter;

/**
 * JPA AttributeConverter to store Map<String, Integer> as JSON string in database. This allows us
 * to store complex data structures in SQLite which doesn't have native JSON support.
 */
@Converter
public class MapToJsonConverter implements AttributeConverter<Map<String, Integer>, String> {

    private static final ObjectMapper objectMapper = new ObjectMapper();

    @Override
    public String convertToDatabaseColumn(Map<String, Integer> attribute) {
        if (attribute == null || attribute.isEmpty()) {
            return "{}";
        }
        try {
            return objectMapper.writeValueAsString(attribute);
        } catch (JsonProcessingException e) {
            throw new RuntimeException("Error converting Map to JSON", e);
        }
    }

    @Override
    public Map<String, Integer> convertToEntityAttribute(String dbData) {
        if (dbData == null || dbData.trim().isEmpty()) {
            return new HashMap<>();
        }
        try {
            return objectMapper.readValue(dbData, new TypeReference<Map<String, Integer>>() {});
        } catch (JsonProcessingException e) {
            throw new RuntimeException("Error converting JSON to Map", e);
        }
    }
}

```

```
=====
FILE: src\main\java\com\cu5448\pcb\entity\SimulationResult.java
=====

package com.cu5448.pcb.entity;

import java.time.LocalDateTime;
import java.util.Map;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * JPA Entity for persisting simulation results to SQLite database. Stores all statistical data from
 * StatisticsCollector along with metadata.
 */
@Entity
@Table(name = "simulation_results")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class SimulationResult {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "pcb_type", nullable = false, length = 50)
    private String pcbType;

    @Column(name = "pcbs_submitted", nullable = false)
    private int pcbsSubmitted;

    @Column(name = "pcbs_completed", nullable = false)
    private int pcbsCompleted;

    @Column(name = "total_failed_pcbs", nullable = false)
    private int totalFailedPcbs;

    @Column(name = "station_failures", columnDefinition = "TEXT")
    @Convert(converter = MapToJsonConverter.class)
    private Map<String, Integer> stationFailures;

    @Column(name = "defect_failures", columnDefinition = "TEXT")
    @Convert(converter = MapToJsonConverter.class)
    private Map<String, Integer> defectFailures;

    @Column(name = "formatted_report", columnDefinition = "TEXT")
    private String formattedReport;

    @Column(name = "created_at", nullable = false)
    private LocalDateTime createdAt;

    @Column(name = "simulation_quantity", nullable = false)
    private int simulationQuantity;

    /** Constructor for creating SimulationResult from StatisticsCollector data. */
    public SimulationResult(
        String pcbType,
        int pcbsSubmitted,
        int pcbsCompleted,
        Map<String, Integer> stationFailures,
        Map<String, Integer> defectFailures,
        String formattedReport,
        int simulationQuantity) {
        this.pcbType = pcbType;
        this.pcbsSubmitted = pcbsSubmitted;
        this.pcbsCompleted = pcbsCompleted;
        this.totalFailedPcbs = pcbsSubmitted - pcbsCompleted;
        this.stationFailures = stationFailures;
        this.defectFailures = defectFailures;
        this.formattedReport = formattedReport;
        this.simulationQuantity = simulationQuantity;
        this.createdAt = LocalDateTime.now();
    }

    @PrePersist
    protected void onCreate() {
        if (createdAt == null) {
            createdAt = LocalDateTime.now();
        }
        // Calculate total failed PCBs if not set
        if (totalFailedPcbs == 0 && pcbsSubmitted > 0) {
            totalFailedPcbs = pcbsSubmitted - pcbsCompleted;
        }
    }
}
```



```
}  
}
```

```
=====br/>FILE: src\main\java\com\cu5448\pcb\factory\PCBFactory.java  
=====
```

```
package com.cu5448.pcb.factory;  
  
import org.springframework.stereotype.Component;  
  
import com.cu5448.pcb.config.PCBProperties;  
import com.cu5448.pcb.model.GatewayBoard;  
import com.cu5448.pcb.model.PCB;  
import com.cu5448.pcb.model.SensorBoard;  
import com.cu5448.pcb.model.TestBoard;  
  
import lombok.RequiredArgsConstructor;  
  
/**  
 * Factory Pattern Implementation using Spring Dependency Injection and  
 * Lombok @RequiredArgsConstructor generates constructor for final fields This factory creates PCB  
 * instances with configuration-driven defect rates.  
 */  
@Component  
@RequiredArgsConstructor  
public class PCBFactory {  
  
    private final PCBProperties pcbProperties;  
  
    public PCB createPCB(String type) {  
        return switch (type.toLowerCase()) {  
            case "testboard", "test", "test board" -> new TestBoard(pcbProperties.getTestboard());  
            case "sensorboard", "sensor", "sensor board" ->  
                new SensorBoard(pcbProperties.getSensorboard());  
            case "gatewayboard", "gateway", "gateway board" ->  
                new GatewayBoard(pcbProperties.getGatewayboard());  
            default -> throw new IllegalArgumentException("Unknown PCB type: " + type);  
        };  
    }  
}
```

```
=====br/>FILE: src\main\java\com\cu5448\pcb\factory\StationFactory.java  
=====
```

```
package com.cu5448.pcb.factory;  
  
import java.util.Map;  
import java.util.function.Function;  
  
import org.springframework.stereotype.Component;  
  
import com.cu5448.pcb.config.StationProperties;  
import com.cu5448.pcb.station.*;  
  
import lombok.RequiredArgsConstructor;  
  
/**  
 * Abstract Factory for creating PCB manufacturing stations using Spring Dependency Injection. This  
 * factory uses a registry pattern to eliminate the need for individual creation methods for each  
 * station type, making it more extensible and following the Abstract Factory pattern.  
 */  
@Component  
@RequiredArgsConstructor  
public class StationFactory {  
  
    private final StationProperties stationProperties;  
  
    // Registry of station constructors using method references  
    private final Map<String, Function<Double, Station>> stationRegistry =  
        Map.of(  
            "ApplySolderPaste", ApplySolderPasteStation::new,  
            "PlaceComponents", PlaceComponentsStation::new,  
            "ReflowSolder", ReflowSolderStation::new,  
            "OpticalInspection", OpticalInspectionStation::new,  
            "HandSoldering", HandSolderingStation::new,  
            "Cleaning", CleaningStation::new,  
            "Depanelization", DepanelizationStation::new,  
            "Test", TestStation::new);  
  
    /**  
     * Creates a station by type name using the Abstract Factory pattern. This method uses a  
     * registry of constructor method references to eliminate the need for individual creation  
     * methods.  
     */
```

```

*
* @param stationType the type of station to create (e.g., "ApplySolderPaste", "Test")
* @return Station instance of the specified type
* @throws IllegalArgumentException if station type is unknown
*/
public Station createStation(String stationType) {
    Function<Double, Station> constructor = stationRegistry.get(stationType);
    if (constructor == null) {
        throw new IllegalArgumentException("Unknown station type: " + stationType);
    }
    return constructor.apply(stationProperties.getFailureRate());
}
}

```

=====

FILE: src\main\java\com\cu5448\pcb\model\DefectRates.java

=====

```

package com.cu5448.pcb.model;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * DefectRates encapsulates defect rates for different manufacturing stations. This class replaces
 * the Map<String, Double> approach with a type-safe, immutable object.
 *
 * <p>Only four stations can detect defects: PlaceComponents, OpticalInspection, HandSoldering, and
 * Test.
 */
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class DefectRates {

    private double placeComponentsDefectRate;
    private double opticalInspectionDefectRate;
    private double handSolderingDefectRate;
    private double testDefectRate;

    /**
     * Gets the defect rate for a specific station type.
     *
     * @param stationType the station type name
     * @return the defect rate for the station, or 0.0 if the station doesn't detect defects
     */
    public double getDefectRate(String stationType) {
        return switch (stationType) {
            case "PlaceComponents" -> placeComponentsDefectRate;
            case "OpticalInspection" -> opticalInspectionDefectRate;
            case "HandSoldering" -> handSolderingDefectRate;
            case "Test" -> testDefectRate;
            default -> 0.0; // Stations that don't detect defects
        };
    }
}

```

=====

FILE: src\main\java\com\cu5448\pcb\model\GatewayBoard.java

=====

```

package com.cu5448.pcb.model;

import lombok.EqualsAndHashCode;

/** Gateway Board PCB Implementation using Lombok */
@EqualsAndHashCode(callSuper = true)
public class GatewayBoard extends PCB {

    private final DefectRates defectRates;

    public GatewayBoard(DefectRates defectRates) {
        super("GatewayBoard");
        this.defectRates = defectRates;
    }

    @Override
    public double getDefectRate(String stationType) {
        return defectRates.getDefectRate(stationType);
    }

    @Override

```

File - C:\Users\lck\Documents\dev\source\CSCA\csca-java\5448\pcb\w4-code.txt

```
    public DefectRates getDefectRates() {
        return defectRates;
    }
}
```

```
=====
FILE: src\main\java\com\cu5448\pcb\model\PCB.java
=====
```

```
package com.cu5448.pcb.model;

import java.util.UUID;

import lombok.Getter;
import lombok.ToString;

/**
 * Abstract PCB Model using Lombok @Getter generates getters for all fields @ToString generates
 * toString method
 */
@Getter
@ToString
public abstract class PCB {

    private final String id;

    private final String type;

    private boolean failed = false;

    private String failureReason = null;

    public PCB(String type) {
        this.id = UUID.randomUUID().toString();
        this.type = type;
    }

    public void setFailed(String reason) {
        this.failed = true;
        this.failureReason = reason;
    }

    public abstract double getDefectRate(String stationType);

    public abstract DefectRates getDefectRates();
}
```

```
=====
FILE: src\main\java\com\cu5448\pcb\model\SensorBoard.java
=====
```

```
package com.cu5448.pcb.model;

import lombok.EqualsAndHashCode;

/** Sensor Board PCB Implementation using Lombok */
@EqualsAndHashCode(callSuper = true)
public class SensorBoard extends PCB {

    private final DefectRates defectRates;

    public SensorBoard(DefectRates defectRates) {
        super("SensorBoard");
        this.defectRates = defectRates;
    }

    @Override
    public double getDefectRate(String stationType) {
        return defectRates.getDefectRate(stationType);
    }

    @Override
    public DefectRates getDefectRates() {
        return defectRates;
    }
}
```

```
=====
FILE: src\main\java\com\cu5448\pcb\model\TestBoard.java
=====
```

```
package com.cu5448.pcb.model;
```

```
import lombok.EqualsAndHashCode;
```

```
/** Test Board PCB Implementation using Lombok */
@EqualsAndHashCode(callSuper = true)
public class TestBoard extends PCB {

    private final DefectRates defectRates;

    public TestBoard(DefectRates defectRates) {
        super("TestBoard");
        this.defectRates = defectRates;
    }

    @Override
    public double getDefectRate(String stationType) {
        return defectRates.getDefectRate(stationType);
    }

    @Override
    public DefectRates getDefectRates() {
        return defectRates;
    }
}
```

```
=====
FILE: src\main\java\com\cu5448\pcb\repository\SimulationResultRepository.java
=====
```

```
package com.cu5448.pcb.repository;

import java.util.List;
import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

import com.cu5448.pcb.entity.SimulationResult;

/**
 * Spring Data JPA repository for SimulationResult entities. Provides CRUD operations and custom
 * queries for simulation result data.
 */
@Repository
public interface SimulationResultRepository extends JpaRepository<SimulationResult, Long> {

    /**
     * Find the most recent simulation result for a specific PCB type.
     *
     * @param pcbType the PCB type to search for
     * @return the most recent SimulationResult for the given PCB type, if any
     */
    Optional<SimulationResult> findFirstByPcbTypeOrderByCreatedAtDesc(String pcbType);

    /**
     * Get the latest simulation result for each PCB type.
     *
     * @return list of the most recent SimulationResult for each PCB type
     */
    @Query(
        "SELECT sr FROM SimulationResult sr WHERE sr.createdAt = "
        + "(SELECT MAX(sr2.createdAt) FROM SimulationResult sr2 WHERE sr2.pcbType = "
        + " sr.pcbType)"
    )
    List<SimulationResult> findLatestResultForEachPcbType();
}
```

```
=====
FILE: src\main\java\com\cu5448\pcb\service\AssemblyLine.java
=====
```

```
package com.cu5448.pcb.service;

import java.util.List;

import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Service;

import com.cu5448.pcb.factory.PCBFactory;
import com.cu5448.pcb.model.PCB;
import com.cu5448.pcb.station.Station;

import lombok.RequiredArgsConstructor;

/**
 * Assembly Line Service using Spring Dependency Injection. Station beans are injected as an ordered
```

```

    * list, eliminating the need for manual station creation and initialization.
    */
    @Service
    @RequiredArgsConstructor
    public class AssemblyLine {

        private final List<Station> stations;

        private final PCBFactory factory;

        private final ApplicationContext applicationContext;

        public void processPCB(PCB pcb, StatisticsCollector stats) {
            for (Station station : stations) {
                station.process(pcb, stats);
                if (pcb.isFailed()) {
                    break;
                }
            }
        }

        public StatisticsCollector runSimulation(String pcbType, int quantity) {
            // Get a new prototype instance of StatisticsCollector for this simulation
            StatisticsCollector stats = applicationContext.getBean(StatisticsCollector.class);

            for (int i = 0; i < quantity; i++) {
                PCB pcb = factory.createPCB(pcbType);
                stats.recordSubmission();

                processPCB(pcb, stats);

                if (!pcb.isFailed()) {
                    stats.recordCompletion();
                }
            }

            return stats;
        }

        public List<Station> getStations() {
            return List.copyOf(stations);
        }
    }

```

```

=====
FILE: src\main\java\com\cu5448\pcb\service\StatisticsCollector.java
=====

```

```

package com.cu5448.pcb.service;

import java.util.HashMap;
import java.util.Map;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Service;

import lombok.Getter;

/**
 * Observer Pattern Implementation as Spring Service using Lombok @Getter generates getter methods
 * for all fields This service observes events from stations during PCB processing. Uses prototype
 * scope to create new instances for each simulation run.
 */
@Service
@Scope("prototype")
@Getter
public class StatisticsCollector {

    private int pcbsSubmitted;

    private final Map<String, Integer> defectFailures;

    private final Map<String, Integer> stationFailures;

    private int completedPCBs;

    public StatisticsCollector() {
        this.pcbsSubmitted = 0;
        this.defectFailures = new HashMap<>();
        this.stationFailures = new HashMap<>();
        this.completedPCBs = 0;
    }

    public void recordSubmission() {
        pcbsSubmitted++;
    }
}

```

File - C:\Users\lck\Documents\dev\source\CSCA\csc-java\5448\pcb\w4-code.txt

```
public void recordDefectFailure(String station) {
    defectFailures.merge(station, 1, Integer::sum);
}

public void recordStationFailure(String station) {
    stationFailures.merge(station, 1, Integer::sum);
}

public void recordCompletion() {
    completedPCBs++;
}

public String generateReport(String pcbType) {
    StringBuilder report = new StringBuilder();

    // Format according to project specification
    report.append(String.format("PCB type: %s\n", pcbType));
    report.append(String.format("PCBs run: %d\n", pcbsSubmitted));

    report.append("\nStation Failures\n");
    // Show all stations in assembly order
    String[] stationNames = {
        "Apply Solder Paste",
        "Place Components",
        "Reflow Solder",
        "Optical Inspection",
        "Hand Soldering/Assembly",
        "Cleaning",
        "Depanelization",
        "Test (ICT or Flying Probe)"
    };

    String[] stationKeys = {
        "ApplySolderPaste",
        "PlaceComponents",
        "ReflowSolder",
        "OpticalInspection",
        "HandSoldering",
        "Cleaning",
        "Depanelization",
        "Test"
    };

    for (int i = 0; i < stationNames.length; i++) {
        int failures = stationFailures.getOrDefault(stationKeys[i], 0);
        report.append(String.format("%s: %d\n", stationNames[i], failures));
    }

    report.append("\nPCB Defect Failures\n");
    // Only show defect-detecting stations
    String[] defectStationNames = {
        "Place Components",
        "Optical Inspection",
        "Hand Soldering/Assembly",
        "Test (ICT or Flying Probe)"
    };

    String[] defectStationKeys = {
        "PlaceComponents", "OpticalInspection", "HandSoldering", "Test"
    };

    for (int i = 0; i < defectStationNames.length; i++) {
        int failures = defectFailures.getOrDefault(defectStationKeys[i], 0);
        report.append(String.format("%s %d\n", defectStationNames[i], failures));
    }

    // Calculate total failures and successful PCBs
    int totalFailed = pcbsSubmitted - completedPCBs;

    report.append("\nFinal Results\n");
    report.append(String.format("Total failed PCBs: %d\n", totalFailed));
    report.append(String.format("Total PCBs produced: %d\n", completedPCBs));

    return report.toString();
}
}
```

```
=====
FILE: src\main\java\com\cu5448\pcb\station\ApplySolderPasteStation.java
=====
```

```
package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;

public class ApplySolderPasteStation extends Station {

    public ApplySolderPasteStation(double failureRate) {
```

File - C:\Users\lck\Documents\dev\source\CSCA\csc-java\5448\pcb\lw4-code.txt

```
        super("ApplySolderPaste", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        return true;
    }
}
```

=====

FILE: src\main\java\com\cu5448\pcb\station\CleaningStation.java

=====

```
package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;

public class CleaningStation extends Station {

    public CleaningStation(double failureRate) {
        super("Cleaning", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        return true;
    }
}
```

=====

FILE: src\main\java\com\cu5448\pcb\station\DepanelizationStation.java

=====

```
package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;

public class DepanelizationStation extends Station {

    public DepanelizationStation(double failureRate) {
        super("Depanelization", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        return true;
    }
}
```

=====

FILE: src\main\java\com\cu5448\pcb\station\HandSolderingStation.java

=====

```
package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;

import lombok.EqualsAndHashCode;

/** Hand Soldering Station using Lombok */
@EqualsAndHashCode(callSuper = true)
public class HandSolderingStation extends Station {

    public HandSolderingStation(double failureRate) {
        super("HandSoldering", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        double defectRate = pcb.getDefectRate("HandSoldering");
        return random.nextDouble() >= defectRate;
    }
}
```

=====

FILE: src\main\java\com\cu5448\pcb\station\OpticalInspectionStation.java

=====

```
package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;
```

```
import lombok.EqualsAndHashCode;

/** Optical Inspection Station using Lombok */
@EqualsAndHashCode(callSuper = true)
public class OpticalInspectionStation extends Station {

    public OpticalInspectionStation(double failureRate) {
        super("OpticalInspection", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        double defectRate = pcb.getDefectRate("OpticalInspection");
        return random.nextDouble() >= defectRate;
    }
}

=====
FILE: src\main\java\com\cu5448\pcb\station\PlaceComponentsStation.java
=====

package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;

import lombok.EqualsAndHashCode;

/**
 * Place Components Station using Lombok @EqualsAndHashCode(callSuper = true) includes parent class
 * fields in equals/hashCode
 */
@EqualsAndHashCode(callSuper = true)
public class PlaceComponentsStation extends Station {

    public PlaceComponentsStation(double failureRate) {
        super("PlaceComponents", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        double defectRate = pcb.getDefectRate("PlaceComponents");
        return random.nextDouble() >= defectRate;
    }
}

=====
FILE: src\main\java\com\cu5448\pcb\station\ReflowSolderStation.java
=====

package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;

public class ReflowSolderStation extends Station {

    public ReflowSolderStation(double failureRate) {
        super("ReflowSolder", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        return true;
    }
}

=====
FILE: src\main\java\com\cu5448\pcb\station\Station.java
=====

package com.cu5448.pcb.station;

import java.util.Random;

import com.cu5448.pcb.model.PCB;
import com.cu5448.pcb.service.StatisticsCollector;

import lombok.Getter;

/**
 * Abstract Station class that can be used as a Spring bean. StatisticsCollector is injected per
 * simulation run rather than at construction time.
 */
```



```
@Getter
public abstract class Station {

    protected final String name;

    protected final double stationFailureRate;

    protected final Random random = new Random();

    public Station(String name, double failureRate) {
        this.name = name;
        this.stationFailureRate = failureRate;
    }

    public void process(PCB pcb, StatisticsCollector stats) {
        if (pcb.isFailed()) {
            return;
        }

        if (checkStationFailure()) {
            stats.recordStationFailure(name);
            pcb.setFailed("Station failure at " + name);
            return;
        }

        boolean operationSuccessful = performOperation(pcb);
        if (!operationSuccessful) {
            stats.recordDefectFailure(name);
            pcb.setFailed("Defect detected at " + name);
        }
    }

    protected boolean checkStationFailure() {
        return random.nextDouble() < stationFailureRate;
    }

    protected abstract boolean performOperation(PCB pcb);
}

=====
FILE: src\main\java\com\cu5448\pcb\station\TestStation.java
=====

package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;

import lombok.EqualsAndHashCode;

/** Test Station using Lombok */
@EqualsAndHashCode(callSuper = true)
public class TestStation extends Station {

    public TestStation(double failureRate) {
        super("Test", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        double defectRate = pcb.getDefectRate("Test");
        return random.nextDouble() >= defectRate;
    }
}

=====
FILE: src\test\java\com\cu5448\pcb\PcbApplicationTests.java
=====

package com.cu5448.pcb;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class PcbApplicationTests {

    @Test
    void contextLoads() {}
}

=====
FILE: src\test\java\com\cu5448\pcb\config\SpringBeanConfigurationTest.java
=====
```

```
package com.cu5448.pcb.config;

import static org.junit.jupiter.api.Assertions.*;

import java.util.List;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.TestConstructor;

import com.cu5448.pcb.service.AssemblyLine;
import com.cu5448.pcb.station.*;

import lombok.RequiredArgsConstructor;

/** Test class to verify Spring bean configuration using Abstract Factory Pattern */
@SpringBootTest
@TestConstructor(autowireMode = TestConstructor.AutowireMode.ALL)
@RequiredArgsConstructor
class SpringBeanConfigurationTest {

    private final AssemblyLine assemblyLine;

    private final List<Station> stations;

    @Test
    void testAssemblyLineInjection() {
        assertNotNull(assemblyLine, "AssemblyLine should be injected");

        List<Station> assemblyStations = assemblyLine.getStations();
        assertEquals(8, assemblyStations.size(), "Assembly line should have 8 stations");
    }

    @Test
    void testStationListOrder() {
        assertNotNull(stations, "Station list should be injected");
        assertEquals(8, stations.size(), "Should have 8 stations");

        // Verify the correct order of stations
        assertEquals("ApplySolderPaste", stations.get(0).getName());
        assertEquals("PlaceComponents", stations.get(1).getName());
        assertEquals("ReflowSolder", stations.get(2).getName());
        assertEquals("OpticalInspection", stations.get(3).getName());
        assertEquals("HandSoldering", stations.get(4).getName());
        assertEquals("Cleaning", stations.get(5).getName());
        assertEquals("Depanelization", stations.get(6).getName());
        assertEquals("Test", stations.get(7).getName());
    }

    @Test
    void testStationFailureRatesAreConfigured() {
        // All stations should have the same configured failure rate
        double expectedFailureRate = 0.002; // From application.properties

        for (Station station : stations) {
            assertEquals(
                expectedFailureRate,
                station.getStationFailureRate(),
                "Station " + station.getName() + " should have configured failure rate");
        }
    }

    @Test
    void testAssemblyLineStationsAreSameAsInjectedList() {
        List<Station> assemblyStations = assemblyLine.getStations();

        // Verify same stations are used (but different list instance due to List.copyOf)
        assertEquals(stations.size(), assemblyStations.size());

        for (int i = 0; i < stations.size(); i++) {
            assertEquals(
                stations.get(i),
                assemblyStations.get(i),
                "Station " + i + " should be the same bean instance");
        }
    }
}

=====
FILE: src\test\java\com\cu5448\pcb\config\SpringConfigurationTest.java
=====

package com.cu5448.pcb.config;

import static org.junit.jupiter.api.Assertions.*;
```

File - C:\Users\lck\Documents\dev\source\CSCA\csca-java\5448\pcb\w4-code.txt

```
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.TestConstructor;

import com.cu5448.pcb.controller.SimulationController;
import com.cu5448.pcb.factory.PCBFactory;
import com.cu5448.pcb.service.AssemblyLine;

import lombok.RequiredArgsConstructor;

@SpringBootTest
@TestConstructor(autowireMode = TestConstructor.AutowireMode.ALL)
@RequiredArgsConstructor
class SpringConfigurationTest {

    private final SimulationController simulationController;

    private final AssemblyLine assemblyLine;

    private final PCBFactory pcbFactory;

    private final StationProperties stationProperties;

    private final PCBProperties pcbProperties;

    @Test
    void testSpringDependencyInjection() {
        // Verify that all Spring beans are properly injected
        assertNotNull(simulationController);
        assertNotNull(assemblyLine);
        assertNotNull(pcbFactory);
    }

    @Test
    void testConfigurationProperties() {
        // Verify that configuration properties are loaded correctly
        assertEquals(0.002, stationProperties.getFailureRate(), 0.0001);

        // Test PCB defect rates from properties (using Lombok-generated getters)
        assertEquals(0.05, pcbProperties.getTestboard().getPlaceComponentsDefectRate(), 0.0001);
        assertEquals(0.002, pcbProperties.getSensorboard().getPlaceComponentsDefectRate(), 0.0001);
        assertEquals(0.004, pcbProperties.getGatewayboard().getPlaceComponentsDefectRate(), 0.0001);
    }

    @Test
    void testPCBFactoryWithConfiguration() {
        // Test that PCB factory creates boards with configuration-driven defect rates
        var testBoard = pcbFactory.createPCB("Test Board");
        assertEquals("TestBoard", testBoard.getType());
        assertEquals(0.05, testBoard.getDefectRate("PlaceComponents"), 0.0001);

        var sensorBoard = pcbFactory.createPCB("Sensor Board");
        assertEquals("SensorBoard", sensorBoard.getType());
        assertEquals(0.002, sensorBoard.getDefectRate("PlaceComponents"), 0.0001);
    }
}
```

=====

FILE: src\test\java\com\cu5448\pcb\factory\PCBFactoryTest.java

=====

```
package com.cu5448.pcb.factory;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import com.cu5448.pcb.config.PCBProperties;
import com.cu5448.pcb.model.*;

class PCBFactoryTest {

    private PCBFactory factory;

    @BeforeEach
    void setUp() {
        factory = new PCBFactory(new PCBProperties());
    }

    @Test
    void testCreateTestBoard() {
        PCB pcb = factory.createPCB("testboard");
        assertInstanceOf(TestBoard.class, pcb);
        assertEquals("TestBoard", pcb.getType());
    }
}
```

```

@Test
void testCreateSensorBoard() {
    PCB pcb = factory.createPCB("sensorboard");
    assertInstanceOf(SensorBoard.class, pcb);
    assertEquals("SensorBoard", pcb.getType());
}

@Test
void testCreateGatewayBoard() {
    PCB pcb = factory.createPCB("gatewayboard");
    assertInstanceOf(GatewayBoard.class, pcb);
    assertEquals("GatewayBoard", pcb.getType());
}

@Test
void testCreateWithAlternativeNames() {
    assertInstanceOf(TestBoard.class, factory.createPCB("test"));
    assertInstanceOf(SensorBoard.class, factory.createPCB("sensor"));
    assertInstanceOf(GatewayBoard.class, factory.createPCB("gateway"));
}

@Test
void testCreateWithInvalidType() {
    assertThrows(IllegalArgumentException.class, () -> factory.createPCB("invalid"));
}
}

```

```

=====
FILE: src\test\java\com\cu5448\pcb\factory\StationFactoryTest.java
=====

package com.cu5448.pcb.factory;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.TestConstructor;

import com.cu5448.pcb.station.*;

import lombok.RequiredArgsConstructor;

/** Test class for StationFactory implementation */
@SpringBootTest
@TestConstructor(autowireMode = TestConstructor.AutowireMode.ALL)
@RequiredArgsConstructor
class StationFactoryTest {

    private final StationFactory stationFactory;

    @Test
    void testFactoryInjection() {
        assertNotNull(stationFactory, "StationFactory should be injected");
    }

    @Test
    void testCreateIndividualStations() {
        // Test station creation using abstract factory method
        Station applySolderPaste = stationFactory.createStation("ApplySolderPaste");
        assertNotNull(applySolderPaste);
        assertEquals("ApplySolderPaste", applySolderPaste.getName());

        Station placeComponents = stationFactory.createStation("PlaceComponents");
        assertNotNull(placeComponents);
        assertEquals("PlaceComponents", placeComponents.getName());

        Station reflowSolder = stationFactory.createStation("ReflowSolder");
        assertNotNull(reflowSolder);
        assertEquals("ReflowSolder", reflowSolder.getName());

        Station opticalInspection = stationFactory.createStation("OpticalInspection");
        assertNotNull(opticalInspection);
        assertEquals("OpticalInspection", opticalInspection.getName());

        Station handSoldering = stationFactory.createStation("HandSoldering");
        assertNotNull(handSoldering);
        assertEquals("HandSoldering", handSoldering.getName());

        Station cleaning = stationFactory.createStation("Cleaning");
        assertNotNull(cleaning);
        assertEquals("Cleaning", cleaning.getName());

        Station depanelization = stationFactory.createStation("Depanelization");
        assertNotNull(depanelization);
        assertEquals("Depanelization", depanelization.getName());
    }
}

```

```

        Station test = stationFactory.createStation("Test");
        assertNotNull(test);
        assertEquals("Test", test.getName());
    }

    @Test
    void testCreateStationByType() {
        // Test station creation by type name using abstract factory
        Station applySolder = stationFactory.createStation("ApplySolderPaste");
        assertInstanceOf(ApplySolderPasteStation.class, applySolder);
        assertEquals("ApplySolderPaste", applySolder.getName());

        Station placeComponents = stationFactory.createStation("PlaceComponents");
        assertInstanceOf(PlaceComponentsStation.class, placeComponents);
        assertEquals("PlaceComponents", placeComponents.getName());

        Station test = stationFactory.createStation("Test");
        assertInstanceOf(TestStation.class, test);
        assertEquals("Test", test.getName());
    }

    @Test
    void testCreateStationByTypeInvalid() {
        // Test invalid station type using abstract factory
        assertThrows(
            IllegalArgumentException.class,
            () -> stationFactory.createStation("InvalidStation"),
            "Should throw exception for invalid station type");

        assertThrows(
            IllegalArgumentException.class,
            () -> stationFactory.createStation(""),
            "Should throw exception for empty station type");

        assertThrows(
            IllegalArgumentException.class,
            () -> stationFactory.createStation("SomeRandomName"),
            "Should throw exception for random station type");
    }

    @Test
    void testAllStationTypesSupported() {
        // Test all expected station types are supported by abstract factory
        String[] stationTypes = {
            "ApplySolderPaste",
            "PlaceComponents",
            "ReflowSolder",
            "OpticalInspection",
            "HandSoldering",
            "Cleaning",
            "Depanelization",
            "Test"
        };

        for (String stationType : stationTypes) {
            assertDoesNotThrow(
                () -> {
                    Station station = stationFactory.createStation(stationType);
                    assertNotNull(
                        station, "Station should be created for type: " + stationType);
                },
                "Should be able to create station for type: " + stationType);
        }
    }
}

```

```

=====
FILE: src\test\java\com\cu5448\pcb\model\DefectRatesTest.java
=====

```

```

package com.cu5448.pcb.model;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

/** Test class for DefectRates model */
class DefectRatesTest {

    @Test
    void testBuilderPattern() {
        DefectRates rates =
            DefectRates.builder()
                .placeComponentsDefectRate(0.01)
                .opticalInspectionDefectRate(0.02)
                .handSolderingDefectRate(0.03)
                .testDefectRate(0.04)
    }
}

```

```

        .build();

        assertEquals(0.01, rates.getPlaceComponentsDefectRate());
        assertEquals(0.02, rates.getOpticalInspectionDefectRate());
        assertEquals(0.03, rates.getHandSolderingDefectRate());
        assertEquals(0.04, rates.getTestDefectRate());
    }

    @Test
    void testGetDefectRateWithValidStations() {
        DefectRates rates =
            DefectRates.builder()
                .placeComponentsDefectRate(0.05)
                .opticalInspectionDefectRate(0.10)
                .handSolderingDefectRate(0.05)
                .testDefectRate(0.10)
                .build();

        assertEquals(0.05, rates.getDefectRate("PlaceComponents"));
        assertEquals(0.10, rates.getDefectRate("OpticalInspection"));
        assertEquals(0.05, rates.getDefectRate("HandSoldering"));
        assertEquals(0.10, rates.getDefectRate("Test"));
    }

    @Test
    void testGetDefectRateWithInvalidStation() {
        DefectRates rates =
            DefectRates.builder()
                .placeComponentsDefectRate(0.05)
                .opticalInspectionDefectRate(0.10)
                .handSolderingDefectRate(0.05)
                .testDefectRate(0.10)
                .build();

        assertEquals(0.0, rates.getDefectRate("ApplySolderPaste"));
        assertEquals(0.0, rates.getDefectRate("ReflowSolder"));
        assertEquals(0.0, rates.getDefectRate("Cleaning"));
        assertEquals(0.0, rates.getDefectRate("Depanelization"));
        assertEquals(0.0, rates.getDefectRate("NonExistentStation"));
    }

    @Test
    void testPCBIntegration() {
        // Test that PCB implementations can use DefectRates
        DefectRates testRates =
            DefectRates.builder()
                .placeComponentsDefectRate(0.05)
                .opticalInspectionDefectRate(0.10)
                .handSolderingDefectRate(0.05)
                .testDefectRate(0.10)
                .build();

        TestBoard testBoard = new TestBoard(testRates);
        DefectRates defectRates = testBoard.getDefectRates();

        assertNotNull(defectRates);
        assertEquals(0.05, testBoard.getDefectRate("PlaceComponents"));
        assertEquals(0.05, defectRates.getDefectRate("PlaceComponents"));

        DefectRates sensorRates =
            DefectRates.builder()
                .placeComponentsDefectRate(0.002)
                .opticalInspectionDefectRate(0.002)
                .handSolderingDefectRate(0.004)
                .testDefectRate(0.004)
                .build();

        SensorBoard sensorBoard = new SensorBoard(sensorRates);
        DefectRates actualSensorRates = sensorBoard.getDefectRates();

        assertNotNull(actualSensorRates);
        assertEquals(0.002, sensorBoard.getDefectRate("PlaceComponents"));
        assertEquals(0.002, actualSensorRates.getDefectRate("PlaceComponents"));
    }
}

```

```

=====
FILE: src\test\java\com\cu5448\pcb\service\StatisticsCollectorTest.java
=====

```

```

package com.cu5448.pcb.service;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class StatisticsCollectorTest {

```

```

private StatisticsCollector stats;

@BeforeEach
void setUp() {
    stats = new StatisticsCollector();
}

@Test
void testInitialState() {
    assertEquals(0, stats.getPcbsSubmitted());
    assertEquals(0, stats.getCompletedPCBs());
    assertTrue(stats.getDefectFailures().isEmpty());
    assertTrue(stats.getStationFailures().isEmpty());
}

@Test
void testRecordSubmission() {
    stats.recordSubmission();
    stats.recordSubmission();
    assertEquals(2, stats.getPcbsSubmitted());
}

@Test
void testRecordCompletion() {
    stats.recordCompletion();
    stats.recordCompletion();
    assertEquals(2, stats.getCompletedPCBs());
}

@Test
void testRecordDefectFailure() {
    stats.recordDefectFailure("PlaceComponents");
    stats.recordDefectFailure("PlaceComponents");
    stats.recordDefectFailure("Test");

    assertEquals(2, stats.getDefectFailures().get("PlaceComponents"));
    assertEquals(1, stats.getDefectFailures().get("Test"));
}

@Test
void testRecordStationFailure() {
    stats.recordStationFailure("ApplySolderPaste");
    stats.recordStationFailure("Cleaning");

    assertEquals(1, stats.getStationFailures().get("ApplySolderPaste"));
    assertEquals(1, stats.getStationFailures().get("Cleaning"));
}

@Test
void testGenerateReport() {
    stats.recordSubmission();
    stats.recordSubmission();
    stats.recordCompletion();
    stats.recordDefectFailure("Test");
    stats.recordStationFailure("Cleaning");

    String report = stats.generateReport("Test Board");

    assertTrue(report.contains("PCB type: Test Board"));
    assertTrue(report.contains("PCBs run: 2"));
    assertTrue(report.contains("Total failed PCBs: 1"));
    assertTrue(report.contains("Total PCBs produced: 1"));
    assertTrue(report.contains("Test (ICT or Flying Probe) 1"));
    assertTrue(report.contains("Cleaning: 1"));
}
}

```

```
=====
FILE: package.json (.json file)
=====
```

```
{
  "name": "pcb-client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/dom": "^10.4.1",
    "@testing-library/jest-dom": "^6.7.0",
    "@testing-library/react": "^16.3.0",
    "@testing-library/user-event": "^13.5.0",
    "@types/jest": "^27.5.2",
    "@types/node": "^16.18.126",
    "@types/react": "^19.1.10",
    "@types/react-dom": "^19.1.7",
    "react": "^19.1.1",
    "react-dom": "^19.1.1",
    "react-scripts": "5.0.1",
    "typescript": "^4.9.5",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject",
    "dev": "react-scripts start"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

```
=====
FILE: public\index.html (.html file)
=====
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
```


File - C:\Users\ck\Documents\dev\source\CSCA\csc-java\5448\pcb\pcb-client\w3-client.txt

```
This HTML file is a template.
If you open it directly in the browser, you will see an empty page.

You can add webfonts, meta tags, or analytics to this file.
The build step will place the bundled scripts into the <body> tag.

To begin the development, run `npm start` or `yarn start`.
To create a production bundle, use `npm run build` or `yarn build`.
-->
</body>
</html>

=====
FILE: public\manifest.json (.json file)
=====

{
  "short_name": "React App",
  "name": "Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#####f"
}

=====
FILE: src\App.css (.css file)
=====

.App {
  text-align: center;
  max-width: 1200px;
  margin: 0 auto;
  padding: 20px;
}

.App-header {
  background-color: #282c34;
  padding: 20px;
  color: white;
  margin-bottom: 30px;
  border-radius: 8px;
}

.App-header h1 {
  margin: 0;
  font-size: 2rem;
}

main {
  display: flex;
  flex-direction: column;
  gap: 30px;
}

/* Simulation Controls */
.simulation-controls {
  background-color: #f5f5f5;
  padding: 20px;
  border-radius: 8px;
  text-align: left;
}

.simulation-controls h2 {
  margin-top: 0;
  color: #333;
}
```

```
.control-group {
  margin-bottom: 15px;
}

.control-group label {
  display: block;
  margin-bottom: 5px;
  font-weight: bold;
}

.control-group select,
.control-group input {
  padding: 8px;
  border: 1px solid #ddd;
  border-radius: 4px;
  font-size: 16px;
  margin-left: 10px;
}

.button-group {
  display: flex;
  gap: 10px;
  margin-bottom: 15px;
  flex-wrap: wrap;
}

.btn-primary,
.btn-secondary,
.btn-danger {
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  font-size: 16px;
  transition: background-color 0.3s;
}

.btn-primary {
  background-color: #007bff;
  color: white;
}

.btn-primary:hover:not(:disabled) {
  background-color: #0056b3;
}

.btn-secondary {
  background-color: #6c757d;
  color: white;
}

.btn-secondary:hover:not(:disabled) {
  background-color: #545b62;
}

.btn-danger {
  background-color: #dc3545;
  color: white;
}

.btn-danger:hover:not(:disabled) {
  background-color: #c82333;
}

button:disabled {
  opacity: 0.6;
  cursor: not-allowed;
}

/* Messages */
.message {
  padding: 15px;
  border-radius: 4px;
  margin: 10px 0;
  font-weight: bold;
}

.message.success {
  background-color: #d4edda;
  color: #155724;
  border: 1px solid #c3e6cb;
}

.message.error {
  background-color: #f8d7da;
  color: #721c24;
  border: 1px solid #f5c6cb;
}
```

```
.loading {
  padding: 15px;
  background-color: #fff3cd;
  color: #856404;
  border: 1px solid #ffeaa7;
  border-radius: 4px;
  margin: 10px 0;
  font-weight: bold;
}

/* Results */
.results-container {
  text-align: left;
}

.results {
  background-color: #fff;
  border: 1px solid #ddd;
  border-radius: 8px;
  padding: 20px;
}

.simulation-result {
  margin-bottom: 30px;
  padding: 20px;
  border: 1px solid #eee;
  border-radius: 8px;
  background-color: #fafafa;
}

.simulation-result h3 {
  margin-top: 0;
  color: #333;
  border-bottom: 2px solid #007bff;
  padding-bottom: 10px;
}

.simulation-result h4 {
  color: #555;
  margin-top: 20px;
}

.summary {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 10px;
  margin: 15px 0;
  padding: 15px;
  background-color: #e9f7ff;
  border-radius: 4px;
}

.summary p {
  margin: 0;
}

.failures,
.defects {
  margin: 20px 0;
}

table {
  width: 100%;
  border-collapse: collapse;
  margin: 10px 0;
}

table th,
table td {
  border: 1px solid #ddd;
  padding: 12px;
  text-align: left;
}

table th {
  background-color: #f8f9fa;
  font-weight: bold;
}

table tr:nth-child(even) {
  background-color: #f9f9f9;
}

.formatted-report {
  margin-top: 20px;
}

.formatted-report pre {
  background-color: #f8f9fa;
}
```

```
padding: 15px;
border-radius: 4px;
overflow-x: auto;
white-space: pre-wrap;
font-family: 'Courier New', monospace;
font-size: 14px;
line-height: 1.4;
}
```

```
/* Responsive design */
@media (max-width: 768px) {
  .App {
    padding: 10px;
  }

  .button-group {
    flex-direction: column;
  }

  .btn-primary,
  .btn-secondary,
  .btn-danger {
    width: 100%;
  }

  .summary {
    grid-template-columns: 1fr;
  }
}
```

```
=====
FILE: src\App.test.tsx (.tsx file)
=====
```

```
import React from 'react';
import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

```
=====
FILE: src\App.tsx (.tsx file)
=====
```

```
import React, { useState } from 'react';
import './App.css';
import SimulationControls from './components/SimulationControls';
import SimulationResult from './components/SimulationResult';
import { apiService } from './services/api';
import { SimulationReport, PCBType } from './types';

function App() {
  const [results, setResults] = useState<SimulationReport[]>([]);
  const [isLoading, setIsLoading] = useState(false);
  const [message, setMessage] = useState<string>('');
  const [error, setError] = useState<string>('');

  const showMessage = (msg: string) => {
    setMessage(msg);
    setError('');
    setTimeout(() => setMessage(''), 3000);
  };

  const showError = (err: string) => {
    setError(err);
    setMessage('');
    setTimeout(() => setError(''), 5000);
  };

  const handleRunSimulation = async (pcbType: PCBType, quantity: number) => {
    setIsLoading(true);
    try {
      const response = await apiService.runSimulation(pcbType, quantity);
      showMessage(response);
    } catch (error) {
      showError(`Failed to start simulation: ${error}`);
    } finally {
      setIsLoading(false);
    }
  };
}
```

```
const handleRunAllSimulations = async (quantity: number) => {
  setIsLoading(true);
  try {
    const response = await apiService.runAllSimulations(quantity);
    showMessage(response);
  } catch (error) {
    showError(`Failed to start simulations: ${error}`);
  } finally {
    setIsLoading(false);
  }
};

const handleGetResults = async (pcbType?: PCBType) => {
  setIsLoading(true);
  try {
    if (pcbType) {
      const result = await apiService.getSimulationResults(pcbType);
      setResults([result]);
      showMessage(`Results retrieved for ${pcbType}`);
    } else {
      const allResults = await apiService.getAllSimulationResults();
      setResults(allResults);
      showMessage(`Retrieved ${allResults.length} simulation results`);
    }
  } catch (error) {
    showError(`Failed to get results: ${error}`);
  } finally {
    setIsLoading(false);
  }
};

const handleClearResults = async () => {
  setIsLoading(true);
  try {
    const response = await apiService.clearAllResults();
    setResults([]);
    showMessage(response);
  } catch (error) {
    showError(`Failed to clear results: ${error}`);
  } finally {
    setIsLoading(false);
  }
};

return (
  <div className="App">
    <header className="App-header">
      <h1>PCB Assembly Line Simulation Client</h1>
    </header>

    <main>
      <SimulationControls
        onRunSimulation={handleRunSimulation}
        onRunAllSimulations={handleRunAllSimulations}
        onGetResults={handleGetResults}
        onClearResults={handleClearResults}
        isLoading={isLoading}
      />

      {message && <div className="message success">{message}</div>}
      {error && <div className="message error">{error}</div>}
      {isLoading && <div className="loading">Processing...</div>}

      <div className="results-container">
        {results.length > 0 && (
          <div className="results">
            <h2>Simulation Results</h2>
            {results.map((result, index) => (
              <SimulationResult key={index} report={result} />
            ))}
          </div>
        )}
      </div>
    </main>
  </div>
);
}
```

export default App;

```
=====
FILE: src\components\SimulationControls.tsx (.tsx file)
=====
```

```
import React, { useState } from 'react';
import { PCBType } from '../types';
```

```

interface SimulationControlsProps {
  onRunSimulation: (pcbType: PCBType, quantity: number) => void;
  onRunAllSimulations: (quantity: number) => void;
  onGetResults: (pcbType?: PCBType) => void;
  onClearResults: () => void;
  isLoading: boolean;
}

const SimulationControls: React.FC<SimulationControlsProps> = ({
  onRunSimulation,
  onRunAllSimulations,
  onGetResults,
  onClearResults,
  isLoading,
}) => {
  const [selectedPcbType, setSelectedPcbType] = useState<PCBType>('test');
  const [quantity, setQuantity] = useState<number>(1000);

  const pcbTypes: { value: PCBType; label: string }[] = [
    { value: 'test', label: 'Test Board' },
    { value: 'sensor', label: 'Sensor Board' },
    { value: 'gateway', label: 'Gateway Board' },
  ];

  return (
    <div className="simulation-controls">
      <h2>PCB Simulation Controls</h2>

      <div className="control-group">
        <label>
          PCB Type:
          <select
            value={selectedPcbType}
            onChange={(e) => setSelectedPcbType(e.target.value as PCBType)}
            disabled={isLoading}
          >
            {pcbTypes.map((type) => (
              <option key={type.value} value={type.value}>
                {type.label}
              </option>
            ))}
          </select>
        </label>
      </div>

      <div className="control-group">
        <label>
          Quantity:
          <input
            type="number"
            value={quantity}
            onChange={(e) => setQuantity(parseInt(e.target.value))}
            min="1"
            max="10000"
            disabled={isLoading}
          />
        </label>
      </div>

      <div className="button-group">
        <button
          onClick={() => onRunSimulation(selectedPcbType, quantity)}
          disabled={isLoading}
          className="btn-primary"
        >
          Run Single Simulation
        </button>

        <button
          onClick={() => onRunAllSimulations(quantity)}
          disabled={isLoading}
          className="btn-primary"
        >
          Run All Simulations
        </button>
      </div>

      <div className="button-group">
        <button
          onClick={() => onGetResults(selectedPcbType)}
          disabled={isLoading}
          className="btn-secondary"
        >
          Get Single Results
        </button>

        <button
          onClick={() => onGetResults()}
          disabled={isLoading}
        >

```

```

      className="btn-secondary"
    >
      Get All Results
    </button>
  </div>

  <div className="button-group">
    <button
      onClick={onClearResults}
      disabled={isLoading}
      className="btn-danger"
    >
      Clear All Results
    </button>
  </div>
</div>
);
};

export default SimulationControls;

=====
FILE: src\components\SimulationResult.tsx (.tsx file)
=====

import React from 'react';
import { SimulationReport } from '../types';

interface SimulationResultProps {
  report: SimulationReport;
}

const SimulationResult: React.FC<SimulationResultProps> = ({ report }) => {
  return (
    <div className="simulation-result">
      <h3>{report.pcbType} Results</h3>

      <div className="summary">
        <p><strong>PCBs Run:</strong> {report.pcbRun}</p>
        <p><strong>Total Produced:</strong> {report.totalPcbsProduced}</p>
        <p><strong>Total Failed:</strong> {report.totalFailedPcbs}</p>
        <p><strong>Success Rate:</strong> {(report.totalPcbsProduced / report.pcbRun) * 100).toFixed(2)}%</p>
      </div>

      <div className="failures">
        <h4>Station Failures</h4>
        <table>
          <thead>
            <tr>
              <th>Station</th>
              <th>Failures</th>
            </tr>
          </thead>
          <tbody>
            {Object.entries(report.stationFailures).map(([station, failures]) => (
              <tr key={station}>
                <td>{station}</td>
                <td>{failures}</td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>

      <div className="defects">
        <h4>Defect Failures</h4>
        <table>
          <thead>
            <tr>
              <th>Defect Type</th>
              <th>Failures</th>
            </tr>
          </thead>
          <tbody>
            {Object.entries(report.defectFailures).map(([defect, failures]) => (
              <tr key={defect}>
                <td>{defect}</td>
                <td>{failures}</td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>

      <div className="formatted-report">
        <h4>Detailed Report</h4>
        <pre>{report.formattedReport}</pre>
      </div>
    </div>
  );
};

```

```

    </div>
  );
};

export default SimulationResult;

=====
FILE: src\index.css (.css file)
=====

body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}

=====
FILE: src\index.tsx (.tsx file)
=====

import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();

=====
FILE: src\react-app-env.d.ts (.ts file)
=====

/// <reference types="react-scripts" />

=====
FILE: src\reportWebVitals.ts (.ts file)
=====

import { ReportHandler } from 'web-vitals';

const reportWebVitals = (onPerfEntry?: ReportHandler) => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};

export default reportWebVitals;

=====
FILE: src\services\api.ts (.ts file)
=====

import { SimulationReport, PCBType } from '../types';

```



```

const BASE_URL = 'http://localhost:8080/api/simulation';

class ApiService {
  private async fetchWithErrorHandling<T>(url: string, options?: RequestInit): Promise<T> {
    try {
      const response = await fetch(url, options);

      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }

      return await response.json();
    } catch (error) {
      console.error('API call failed:', error);
      throw error;
    }
  }

  async runSimulation(pcbType: PCBType, quantity: number = 1000): Promise<string> {
    const response = await fetch(`${BASE_URL}/run/${pcbType}?quantity=${quantity}`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
    });

    if (!response.ok) {
      throw new Error(`Failed to start simulation: ${response.status}`);
    }

    return await response.text();
  }

  async runAllSimulations(quantity: number = 1000): Promise<string> {
    const response = await fetch(`${BASE_URL}/run/all?quantity=${quantity}`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
    });

    if (!response.ok) {
      throw new Error(`Failed to start simulations: ${response.status}`);
    }

    return await response.text();
  }

  async getSimulationResults(pcbType: PCBType): Promise<SimulationReport> {
    return this.fetchWithErrorHandling<SimulationReport>(`${BASE_URL}/results/${pcbType}`);
  }

  async getAllSimulationResults(): Promise<SimulationReport[]> {
    return this.fetchWithErrorHandling<SimulationReport[]>(`${BASE_URL}/results/all`);
  }

  async getPcbTypes(): Promise<string[]> {
    return this.fetchWithErrorHandling<string[]>(`${BASE_URL}/types`);
  }

  async clearAllResults(): Promise<string> {
    const response = await fetch(`${BASE_URL}/results`, {
      method: 'DELETE',
    });

    if (!response.ok) {
      throw new Error(`Failed to clear results: ${response.status}`);
    }

    return await response.text();
  }
}

export const apiService = new ApiService();

=====
FILE: src\setupTests.ts (.ts file)
=====

// jest-dom adds custom jest matchers for asserting on DOM nodes.
// allows you to do things like:
// expect(element).toHaveTextContent(/react/i)
// learn more: https://github.com/testing-library/jest-dom
import '@testing-library/jest-dom';

=====

```

File - C:\Users\lck\Documents\dev\source\CSCA\csca-java\5448\pcb\pcb-client\w3-client.txt

FILE: src\types.ts (.ts file)

=====

```
export interface SimulationReport {
  pcbType: string;
  pcbsRun: number;
  stationFailures: Record<string, number>;
  defectFailures: Record<string, number>;
  totalFailedPcbs: number;
  totalPcbsProduced: number;
  formattedReport: string;
}

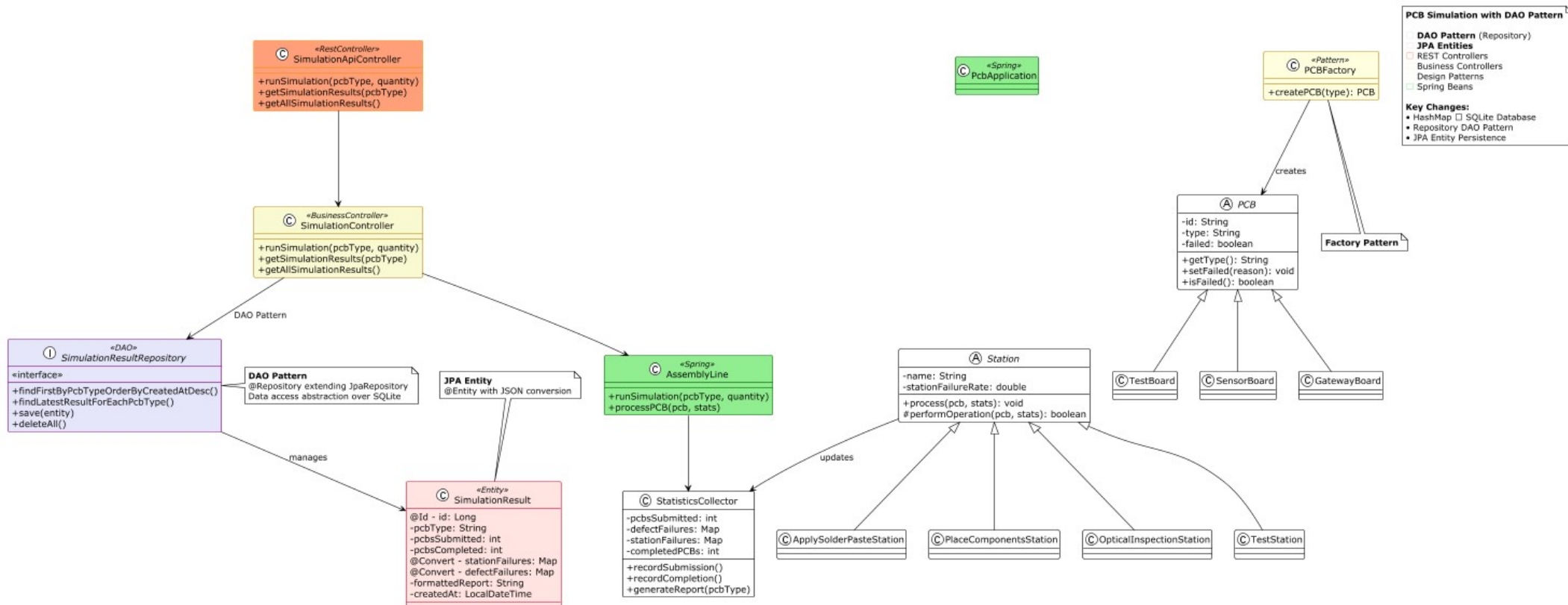
export type PCBType = 'test' | 'sensor' | 'gateway';
```

=====

FILE: tsconfig.json (.json file)

=====

```
{
  "compilerOptions": {
    "target": "es5",
    "lib": [
      "dom",
      "dom.iterable",
      "esnext"
    ],
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "noFallthroughCasesInSwitch": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx"
  },
  "include": [
    "src"
  ]
}
```



PCB Assembly Line Simulation Client

PCB Simulation Controls

PCB Type: Test Board

Quantity:

Run Single Simulation

Run All Simulations

Get Single Results

Get All Results

Clear All Results

Simulation Results

test Results

PCBs Run: 1000	Total Produced: 727	Total Failed: 273	Success Rate: 72.70%
----------------	---------------------	-------------------	----------------------

Station Failures

Station	Failures
Depanelization	1
Test	1
PlaceComponents	1
ApplySolderPaste	3
HandSoldering	2
Cleaning	2
ReflowSolder	2
OpticalInspection	2

Defect Failures

Defect Type	Failures
Test	72
PlaceComponents	51
HandSoldering	46
OpticalInspection	90

ElementsConsoleRecorderSourcesNetwork

Preserve logDisable cacheNo throttling

Filter

AllFetch/XHRDocCSSJSFontImgMediaManifestSocketWasmOther

20 ms40 ms60 ms80 ms100 ms

NameXHeadersPreviewResponseInitiatorTiming

test1 {
 "pcbType": "test",
 "pcbsRun": 1000,
 "stationFailures": {
 "Depanelization": 1,
 "Test": 1,
 "PlaceComponents": 1,
 "ApplySolderPaste": 3,
 "HandSoldering": 2,
 "Cleaning": 2,
 "ReflowSolder": 2,
 "OpticalInspection": 2
 },
 "defectFailures": {
 "Test": 72,
 "PlaceComponents": 51,
 "HandSoldering": 46,
 "OpticalInspection": 90
 },
 "totalFailedPcbs": 273,
 "totalPcbsProduced": 727,
 "formattedReport": "PCB type: test\nPCBs run: 1000\n"
}

1 requests

ConsoleWhat's new X AI assistanceIssues

What's new in DevTools 139

See all new features

new

See past highlights from Chrome 138