Report di Audit Ingegneristico

JavaBrew.pdf

SWE-Audit-Agent

23 febbraio 2026

Indice

1 Contesto del Documento

PANORAMICA STRUTTURATA DEL PROGETTO JAVABREW

OBIETTIVO DEL PROGETTO

JavaBrew è una piattaforma integrata per la gestione di distributori automatici connessi alla rete. L'applicazione consente agli utenti di acquistare prodotti tramite smartphone scansionando un codice QR univoco e utilizzando un wallet digitale, mentre offre ai gestori una dashboard centralizzata per monitorare vendite, scorte, malfunzionamenti e pianificare manutenzione e rifornimento in modo efficiente.

CASI D'USO PRINCIPALI

- UC-1 - User Login: L'utente inserisce email e password per accedere all'interfaccia personalizzata in base al proprio ruolo. - UC-2 - User Registration: L'utente crea un nuovo account fornendo nome, cognome, email e password per accedere al sistema. - UC-3 - Buy Item: Il cliente autenticato e connesso a una vending machine seleziona un prodotto, il sistema verifica saldo e disponibilità, scala il saldo ed eroga il prodotto. - UC-4 - Connect to Vending Machine: Il cliente scannerizza il QR code univoco della macchina e il sistema la collega mostrando l'interfaccia dell'inventario. - UC-5 - Recharge Balance: Il cliente autenticato sceglie metodo di pagamento e importo per ricaricare il proprio wallet digitale. - UC-6 - Mark task as completed: Il tecnico (worker) clicca sulla task in corso e marca la sua conclusione nel sistema che registra l'evento nei log. - UC-7 - View Analytics: L'amministratore accede alla pagina personale e visualizza le analytics relative a utenti, macchine e articoli. - UC-8.1 - Create New Vending Machine: L'amministratore compila un modulo con i dati della nuova macchina e il sistema la salva nel database.

REQUISITI FUNZIONALI

- Autenticazione e autorizzazione: login e registrazione con validazione credenziali e ruoli differenziati (Admin, Customer, Worker). - Gestione wallet digitale: visualizzazione saldo, ricarica con diversi metodi di pagamento (contanti, online), tracciamento storico transazioni. - Acquisto prodotti: scansione QR code, connessione a vending machine, selezione articoli, verifica disponibilità e saldo, erogazione e scalamento automatico. - Gestione macchine: creazione, configurazione prezzi, monitoraggio stato operativo e malfunzionamenti mediante autodiagnosi. - Gestione manutenzione: generazione automatica di report per scorte insufficienti e guasti, assegnazione task ai tecnici, tracciamento completamento. - Analytics e reporting: visualizzazione statistiche vendite, trend articoli, monitoraggio inventario e efficienza manutenzione. - Gestione utenti: CRUD completo per admin, worker, customer con attributi specifici per ruolo.

REQUISITI NON FUNZIONALI

- Scalabilità: architettura modulare su container Docker per deployment distribuito di applicazione e database PostgreSQL. - Performance: database in-memory H2 per test rapidi, copertura codice superiore all'80% mediante JUnit 5 e JaCoCo. - Sicurezza: gestione credenziali, validazione input, gestione autorizzazione per ruoli, transazioni persistenti tramite JPA/Hibernate. - Affidabilità: gestione eccezioni, messaggi di errore, riprovazione fallimenti di sistema, logging di operazioni critiche. - Usabilità: interfaccia mobile-friendly con mockup dedicati per cliente, tecnico e amministratore, navigazione intuitiva per QR code e acquisti. - Manutenibilità: separazione Controller-Service-DAO-Model, pattern Builder, Data Mapper, Singleton DBManager, codice strutturato in package Java.

ARCHITETTURA

L'architettura segue un pattern a strati (Layered Architecture): User → Interface → Controllers → Services → Model/DAO → ORM Hibernate → PostgreSQL. Tecnologie principali: Java 24, Maven 3.9.9, PostgreSQL 16, H2 per test, Docker, Docker Compose, JPA 3.2.0, Hibernate ORM 6.5.2.Final, Jakarta Transaction API 2.0.1. Il design domain model utilizza composizione invece di estensione, pattern Builder per ConcreteVendingMachine, Data Mapper con classi mapper dedicate. Livello DAO centralizza persistenza tramite interfacce (TaskDao, UserDao, VendingMachineDao, TransactionDao, ConnectionDao, MaintenanceDao, InventoryDao, ItemDao)

e implementazioni concrete con DBManager (Singleton). Business logic separata in Controller dedicati per ruolo e Service specifici per dominio. Database schema generato automaticamente da ORM con gestione utenti (app_user, admin, worker, customer), macchine e inventario (vendingmachine, concretevendingmachine, inventory, item), transazioni (transaction, transactionitem), manutenzione (maintenance_report, task) e connessioni (connection).

STRATEGIA DI TESTING

Test di unità con JUnit 5 e Mockito per isolamento componenti Service dal livello persistenza, simulando DAO tramite @Mock e @InjectMocks per focalizzare logica business e esecuzione rapida (centinaia di test in pochi secondi). Test di integrazione con database H2 in-memory per validare interazione Service-DAO-Database e rilevare errori JPA non individuabili in unit test. Analisi copertura codice tramite JaCoCo integrato in Maven raggiungendo superiore all'80% con identificazione codice morto. Per ogni UC mappati test specifici: UC-1 Login (5 test), UC-2 Signup (5 test), UC-3 Buy Item (8 test), UC-4 Connect Machine (8 test), UC-5 Recharge (7 test), UC-6 Finish Task (9 test), UC-7 View Analytics (13 test), UC-8.1 Create Machine (5 test). Utilizzo strumenti IA (GitHub Copilot per automazione boilerplate, GPT-4.1 per architettura, Claude 4 Sonnet per ottimizzazione codice, Gemini per sintesi progetti) sempre sotto supervisione umana. Codice disponibile su GitHub: https://github.com/matteominin/smartVend

2 Sintesi Esecutiva

SINTESI ESECUTIVA AUDIT DOCUMENT "JAVABREW.PDF"

Inquadramento del documento

Il documento JavaBrew costituisce la specifica di ingegneria del software per un sistema di gestione di distributori automatici integrati con servizi di manutenzione, ricarica e analitiche. L'analisi copre 29 problemi distribuiti su tre categorie principali: Requisiti (17), Architettura (4) e Testing (8). La valutazione complessiva evidenzia un progetto con buone fondamenta strutturali, ma con significativi gap tra le specifiche funzionali e la loro tracciabilità verso la verifica e l'implementazione.

Pattern di problemi riscontrati

I problemi riscontrati seguono tre pattern ricorrenti. Primo: tracciabilità debole tra casi d'uso e test di validazione. Molti casi d'uso descrivono flussi alternativi e scenari di errore non adeguatamente coperti dai test elencati, creando rischi di non rilevare malfunzionamenti in produzione. Secondo: discrepanze tra la specifica testuale dei requisiti e il modello di dominio implementato. Ad esempio, la gestione delle transazioni è complessa nel database ma sottodimensionata nei casi d'uso, mentre mockup e UI mostrano funzionalità (validazione password, forgot password) non presenti nella specifica formale. Terzo: ambiguita nei flussi alternativi e nelle post-condizioni, con scarsa descrizione delle azioni di sistema, dei messaggi utente e dei cambiamenti di stato conseguenti.

Aree critiche e severity

Tre problemi di severity HIGH richiedono intervento immediato: (1) l'assenza di test specifici per i flussi di errore di connessione alla macchinetta nel caso d'uso UC-4, (2) la mancanza di scenari di validazione input nella copertura test per UC-8.1 Create New Vending Machine, (3) l'assenza di test che verifichino il fallimento del pagamento nel flusso di ricarica UC-5. Inoltre, mancano complessivamente test end-to-end su disconnessione esplicita, rollback transazionale e gestione di errori di sistema a livello controller. Questi gap espongono il sistema a rischi elevati di regressione e comportamenti non predicibili in scenari di stress o guasto.

Valutazione complessiva della qualita

Il documento evidenzia una buona struttura organizzativa e un modello di dominio articolato, ma una scarsa maturità nella specifica formale dei requisiti e nella definizione della strategia di test. La qualita complessiva è BASSA-MEDIA: il progetto richiede interventi significativi prima della fase di implementazione per allineare le tre dimensioni (requisiti, architettura, verifica).

La densita di problemi MEDIUM (18 su 29) indica che sebbene non vi siano difetti architetturali irreversibili, la specifica necessita di numerosi chiarimenti e integrazioni per garantire una costruzione affidabile del sistema.

Azioni prioritarie

1. TRACCIABILITA REQUISITI-TEST: creare una matrice esplicita di mappatura tra ogni flusso principale e alternativo dei casi d'uso e i corrispondenti test case (almeno uno per flusso); completare la copertura test su errori di connessione (UC-4), validazione input (UC-8.1), rollback transazionale (UC-5) e disconnessione (UC-3).

2. ALLINEAMENTO SPECIFICA: revisionare e arricchire i casi d'uso con descrizioni dettagliate dei flussi alternativi, precisando azioni di sistema, messaggi utente, cambiamenti di stato e rollback; allineare le funzionalita visibili nei mockup (forgot password, conferma password) con la specifica testuale.

3. COERENZA MODELLO: risolvere il disallineamento tra la logica di persistenza (entita Transaction, Inventory, relazioni complesse) e i casi d'uso, descrivendo esplicitamente come i servizi di business gestiscono tali operazioni, e chiarire la gerarchia di composizione/estensione nel modello di dominio.

3 Punti di Forza

PUNTI DI FORZA DEL DOCUMENTO

- Struttura organizzativa chiara e metodica: il documento segue un flusso logico ben definito (Requisiti → Architettura → Testing) che facilita la comprensione del progetto complessivo e della visione d'insieme del sistema JavaBrew.

- Copertura completa dei casi d'uso con template standardizzato: i casi d'uso sono descritti con una struttura coerente (precondizioni, flusso principale, flusso alternativo, test), facilitando la tracciabilità e la verifica dei requisiti funzionali per ogni attore (Customer, Worker, Admin).

- Architettura a strati ben definita: il documento presenta una separazione netta tra livelli (Controller, Service, DAO, ORM) che favorisce manutenibilità, testabilità e scalabilità del codice.

- Scelta tecnologica consapevole e giustificata: le tecnologie selezionate (Java, Hibernate, PostgreSQL, Docker) sono appropriate al contesto e supportate da motivazioni tecniche esplicite.

- Mockups UI dettagliati e differenziati per ruolo: le interfacce utente sono progettate specificamente per le esigenze di Customer, Worker e Admin, con diagrammi di navigazione che illustrano i flussi operativi.

- Strategia di testing articolata: il documento descrive test unitari, di integrazione e di copertura del codice, dimostrando consapevolezza della qualità del software.

4 Scorecard per Area

Area	HIGH	MEDIUM	LOW	Punteggio	Voto
Requisiti	0	14	3	0/100	**F**
Architettura	0	1	3	81/100	**B**
Testing	3	3	2	0/100	**F**
Media Complessiva				**27/100**	**F**

5 Copertura Feature Attese

Su 7 feature attese: 6 presenti, 1 parziali, 0 assenti. Copertura media: **95%**.

Feature	Stato	Copertura	Evidenza
Unit testing framework implementation	Presente	100% (5/5)	La sezione 8 descrive l'uso di JUnit 5 e Mockito, mostra test con assertEquals/assertNotNull e verify (Listing 2), illustra un approccio sistematico ai test di unità e di integrazione (8.1, 8.1.2), e riporta la copertura tramite JaCoCo (Figura 18).
Use of UML Diagrams for system modeling	Presente	100% (8/8)	Sono presenti use case diagram per User, Customer, Worker e Admin (Figure 1–4) con attori e relazioni; diagrammi UML di classi per domain model, DAO, services e controllers (Figure 12–16); un diagramma di architettura a strati (Figura 11); mockup UI e diagramma di navigazione (Figure 5–10) che chiariscono requisiti funzionali e flussi di navigazione, usando notazione UML standard («include», «extends», «summary»).
Identification and definition of system actors	Presente	100% (8/8)	La sezione 2.2 definisce chiaramente i ruoli Customer, Admin e Worker e le loro responsabilità; i casi d'uso per ciascun attore (sezione 4, Figure 1–4) mostrano interazioni specifiche con il sistema e azioni utente dettagliate, strutturando i requisiti funzionali per i diversi ruoli (es. acquisto, ricarica, gestione CRUD, manutenzione).
Definition and Documentation of Use Cases	Presente	100% (5/5)	I casi d'uso UC-1..UC-8.1 includono ID, nome, livello, attori, pre-condizioni, post-condizioni, flusso principale, flussi alternativi e riferimento ai test; i diagrammi di Figure 1–4 illustrano anche relazioni tra use case («include», «extends») come per Buy Item e Connect to Vending Machine.
User interface and interaction design principles	Parziale	71% (5/7)	La sezione 5 mostra mockup per login, registrazione, dashboard per i ruoli e catalogo prodotti (Figure 6–10) e un diagramma di navigazione con percorsi chiari; sono descritti campi strutturati per input utente e ruoli distinti con relative funzionalità. Non viene però trattato un processo di prenotazione né sono esplicitati meccanismi di validazione lato UI, sebbene alcuni errori siano citati a livello di casi d'uso.

Feature	Stato	Copertura	Evidenza
Separation of concerns in software architecture	Presente	100% (5/5)	La struttura dei package (sezione 6.2) separa chiaramente model, controllers, services, dao e db; il diagramma a strati (Figura 11) e i diagrammi di controller e services (Figure 15–16) documentano interazioni tra Controller, Service, DAO e modello e descrivono i ruoli di ciascun componente, enfatizzando un design modulare manutenibile.
Data Access Object (DAO) pattern	Presente	100% (7/7)	La sezione 6.4 descrive interfacce DAO per entità diverse (TaskDao, UserDao, VendingMachineDao, ecc.) con operazioni CRUD e query specifiche (Figura 14) e implementazioni *Impl* che incapsulano l'accesso al DB tramite DBManager singleton; viene esplicitata l'astrazione dell'accesso ai dati e in 8.3 sono elencati test per vari DAO (es. TransactionDaoImplTest, ConcreteVendingMachineDaoImplTest).

6 Tabella Riassuntiva

Categoria	HIGH	MEDIUM	LOW	Totale
Requisiti	0	14	3	17
Architettura	0	1	3	4
Testing	3	3	2	8
Totale	**3**	**18**	**8**	**29**

7 Dettaglio delle Problematiche

7.1 Requisiti (17 problemi)

UC-8

REQ-013 — LOW — Pagina 13 Esiste un'incoerenza di identificazione per il caso d'uso "Create New Vending Machine": l'ID è indicato come "UC-8.1", mentre nel diagramma degli Admin Use Cases compare un solo caso numerato 8 per le macchine, creando ambiguità di tracciamento.

4.4.2 Create New Vending Machine UC-8.1 Create New Vending Machine

Raccomandazione: Uniformare l'identificativo del caso d'uso: se il requisito è uno solo, rinominarlo coerentemente come "UC-8 Create New Vending Machine" in tutto il documento. Se invece UC-8.1 è una specializzazione di un UC-8 generico (Manage Vending Machines CRUD), esplicitare anche UC-8 nel testo con la sua descrizione, indicando la relazione include/extend tra i due.

Vedi anche: TST-002

Problemi Generali

REQ-001 — MEDIUM — Pagina 7 Tutti i casi d'uso rimandano genericamente a "Vedi i test correlati" senza una vera tracciabilità o mappatura esplicita verso i test elencati nella sezione 8.3. Questo rende difficile verificare la copertura completa dei flussi principali e alternativi.

> *Test Vedi i test correlati*

Raccomandazione: Per ciascun caso d'uso, sostituire la voce generica "Vedi i test correlati" con un elenco puntuale dei nomi dei test che lo coprono (es. "UserServiceTest.loginWithValidCredentials", "UserControllerTest.login_ValidCredentials_CallsUserServiceAndReturnsUser"), utilizzando sempre la stessa nomenclatura usata nella sezione 8.3. In alternativa, aggiungere per ogni UC un campo "ID test" che riporti esattamente gli identificatori elencati in 8.3.

REQ-002 — MEDIUM — Pagina 7 Nel caso d'uso di login, il flusso alternativo non specifica chiaramente cosa accade allo stato di autenticazione e alla pagina visualizzata dopo un errore di credenziali o un errore di sistema.

> *Alternative Flow 3.1 Credenziali errate: messaggio di errore che invita a riprovare. 4.1 Errore di sistema: messaggio di errore che invita a riprovare più tardi.*

Raccomandazione: Ampliare il flusso alternativo specificando i passi operativi: ad esempio per 3.1 indicare che 1) il sistema non autentica l'utente, 2) rimane sulla pagina di login, 3) mostra un messaggio di errore esplicito, 4) non apre alcuna dashboard. Per 4.1 indicare se l'utente resta sulla stessa pagina, se il tentativo viene loggato e se vi sono limiti ai tentativi.

REQ-003 — MEDIUM — Pagina 8 Nel caso d'uso di registrazione utente manca la descrizione di come viene gestita la conferma password (presente nei mockup) e la validazione dell'unicità dell'email, generando una discrepanza tra UI e requisiti.

> *Basic Flow 1. Lo user apre la pagina per registrare un account (Fig. 7). 2. Inserisce nome, cognome, email e password. 3. Il sistema valida i campi inseriti. 4. Il sistema crea un nuovo account.*

Raccomandazione: Allineare il Basic Flow con la UI della Figura 7: specificare che l'utente inserisce anche "Conferma password" e che il sistema controlla corrispondenza tra password e conferma e l'unicità dell'email. Nel Flusso Alternativo aggiungere un punto dedicato a "email già registrata" e uno a "password e conferma non coincidono", descrivendo i messaggi di errore e cosa succede alla form.

REQ-004 — MEDIUM — Pagina 8 Nel flusso alternativo di registrazione è citato un generico "Errore di autenticazione" che è ambiguo in un contesto di sign-up, perché non è chiaro se l'errore riguarda la creazione dell'utente o un auto-login post-registrazione.

> *Alternative Flow 3.1 Campi errati o vuoti: il sistema mostra un messaggio di errore. 4.1 Errore di autenticazione: appare un messaggio che invita a ripro- vare più tardi.*

Raccomandazione: Rinominare e dettagliare il punto 4.1: ad esempio specificare se dopo la creazione dell'account viene effettuato un login automatico e che tipo di errore può fallire (problema di sessione, DB, ecc.). Esplicitare gli effetti sull'account: viene creato ma l'utente non è loggato? O la creazione viene annullata? Descrivere il comportamento atteso.

REQ-005 — MEDIUM — Pagina 9 Per il caso d'uso "Buy Item" il flusso alternativo non specifica chiaramente cosa accade alla connessione nel caso di prodotto esaurito, mentre nel caso di saldo insufficiente è esplicitata la disconnessione. Ciò crea una regola incoerente sulla durata della connessione.

> *Alternative Flow 3.1 Saldo insufficiente: errore e disconnessione. 3.2 Prodotto esaurito: errore.*

Raccomandazione: Specificare per 3.2 se il cliente resta connesso alla macchina per poter scegliere un altro prodotto o se viene anche in questo caso disconnesso. Aggiornare il Basic Flow o il diagramma di navigazione (Figura 5) per riflettere in modo coerente cosa accade allo stato di connessione in tutti gli scenari di errore.

REQ-006 — MEDIUM — Pagina 9 Nel caso d'uso "Buy Item" la post-condizione è molto sintetica e non menziona aspetti fondamentali di business già presenti nel modello e nel database, come la registrazione della transazione e l'aggiornamento dell'inventario.

> *Post-conditions Il prodotto è stato erogato e il saldo aggiornato.*

Raccomandazione: Arricchire le post-condizioni includendo: 1) l'aggiornamento della quantità dell'Item e dell'Inventory associato alla ConcreteVendingMachine, 2) la creazione di una Transaction e dei relativi TransactionItem collegati al customer e alla macchina, 3) eventuali log di audit. Questo allinea i requisiti alla logica di business mostrata nei servizi (CustomerService, TransactionService) e nello schema E-R.

REQ-007 — MEDIUM — Pagina 10 Nel caso d'uso "Connect to Vending Machine" i flussi alternativi non specificano quali azioni intraprende il sistema per ciascun errore (messaggio all'utente, log, possibilità di riprovare), riducendo la testabilità e chiarezza.

> *Alternative Flow • Vending machine già connessa: errore. • Vending machine fuori uso: errore.*

Raccomandazione: Per ciascun punto del flusso alternativo, aggiungere i passi operativi: ad esempio 1) il sistema non crea una nuova Connection, 2) mostra un messaggio specifico (es. "Macchina già occupata da un altro utente" o "Macchina fuori servizio"), 3) resta sulla schermata di scan QR con possibilità di riprovare o annullare. Specificare se viene creato un log o un report di manutenzione per "vending machine fuori uso".

REQ-008 — MEDIUM — Pagina 10 Il caso d'uso "Recharge Balance" non definisce in modo chiaro il comportamento per il flusso alternativo quando il pagamento non riesce: manca ciò che viene mostrato all'utente, se vi sono rollback, e se la transazione viene comunque registrata come fallita.

> *Alternative Flow 3.1 Pagamento non riuscito.*

Raccomandazione: Espandere il punto 3.1 specificando: 1) che il saldo del customer rimane invariato, 2) che tipo di messaggio vede l'utente, 3) se viene registrata una Transaction con stato fallito o se non viene generata alcuna transazione, 4) se l'utente può immediatamente riprovare con un altro metodo di pagamento.

REQ-009 — MEDIUM — Pagina 10 Il caso d'uso "Recharge Balance" non menziona esplicitamente la creazione di una transazione di ricarica, sebbene la logica di business e il DB prevedano entità Transaction e TransactionService, creando un gap nei requisiti di tracciabilità finanziaria.

Post-conditions Il customer vede il nuovo saldo aggiornato.

Raccomandazione: Aggiornare le post-condizioni per specificare che: 1) viene creata una Transaction con il nuovo saldo (initialBalance e updatedBalance) e il PaymentMethod utilizzato, 2) questa transazione è consultabile nello storico (getTransactionHistory). Assicurarsi che il Basic Flow includa un passo di creazione della transazione in linea con TransactionServiceTest.testCreateTransactionValid.

REQ-010 — MEDIUM — Pagina 11 Nel caso d'uso "Finish Task" la pre-condizione è minimale: non chiarisce se il worker deve essere assegnato alla task che tenta di chiudere, condizione importante per la sicurezza e coerenza del flusso di manutenzione.

Pre-conditions Il worker è autenticato.

Raccomandazione: Integrare le pre-condizioni con una regola di business del tipo: "La task è assegnata al worker autenticato e si trova in uno stato che consente la chiusura (es. inProgress)". Allineare questo con i test WorkerServiceTest.testChangeTaskStatusWithCompletedTask e con gli stati definiti in MaintenanceStatus.

REQ-011 — MEDIUM — Pagina 11 Nel caso d'uso "Finish Task" il flusso alternativo "Errore nel salvataggio" non spiega cosa succede allo stato della task né cosa vede l'utente, impedendo di progettare test di integrazione precisi.

Alternative Flow • Errore nel salvataggio: messaggio di errore.

Raccomandazione: Specificare 1) che la task rimane nello stato precedente (es. inProgress), 2) che il worker vede un messaggio d'errore e resta sulla schermata della task o viene reindirizzato alla lista, 3) se il sistema tenta un retry o logga l'errore per l'admin. Integrare queste condizioni nelle post-condizioni o in note aggiuntive.

REQ-012 — LOW — Pagina 12 Il caso d'uso "View Analytics" è descritto solo come caricamento dei dati, ma non chiarisce quali metodi di business vengono utilizzati (es. getMachines, getUsers) né quali viste statistiche siano garantite, rendendo il requisito troppo generico rispetto ai servizi implementati.

Post-conditions L'admin vede le analytics degli utenti e degli items.

Raccomandazione: Dettagliare le analytics previste: ad esempio specificare che l'admin può visualizzare conteggi di utenti per ruolo, vendite per macchina, vendite per item, ecc. Collegare queste metriche alle operazioni di AdminService (es. getMachines(), viewAnalytics(machineId: long)) così da avere un requisito verificabile e allineato alla logica di business.

REQ-014 — MEDIUM — Pagina 13 Nel caso d'uso "Create New Vending Machine" non è indicato come vengono inizializzati i dati correlati fondamentali (Inventory associato, status iniziale della macchina, eventuale creazione di QR code), sebbene tali entità siano presenti nel modello e nello schema E-R.

Post-conditions Il sistema contiene una nuova vending machine che appare nelle analy- tics.

Raccomandazione: Estendere le post-condizioni per indicare che: 1) viene creato l'oggetto ConcreteVendingMachine con status iniziale (es. operative), 2) viene creato l'Inventory associato (come da relazione uno-a-uno nel modello), 3) se previsto, viene generato un identificativo/QR code collegato alla macchina. Allineare la descrizione con AdminService.testCreateMachineReturnsCreatedMachine e con il dominio ConcreteVendingMachine/Inventory.

REQ-015 — LOW — Pagina 4 La sezione sui possibili svantaggi introduce funzionalità future importanti (offline register, anonymous users) che non sono presenti nei casi d'uso né nella progettazione corrente, rischiando di essere interpretate come requisiti già in scope.

> *Questo problema potrebbe essere affrontato nelle successive iterazioni del software, implementando un meccani- smo di rilevamento dei distributori non connessi (ad esempio tramite polling) e introducendo un offline register per tracciare localmente le transazioni avvenute durante il periodo di disconnes- sione. ... Nelle successive iterazio- ni del software, questo problema potrebbe essere risolto permettendo a utenti non registrati (anonymous users) di effettuare transazioni solamente in contante.*

Raccomandazione: Esplicitare che queste funzionalità (offline register, anonymous users) non fanno parte del perimetro della versione attuale. Aggiungere una sezione "Requisiti fuori scope" o "Future work" che le elenchi chiaramente come evoluzioni previste, per evitare che vengano confuse con obiettivi della release descritta nei casi d'uso.

REQ-016 — MEDIUM — Pagina 15 Nel diagramma di navigazione la funzione "Forgot password?" visualizzata nel mockup di login non ha un corrispondente caso d'uso né una descrizione dei requisiti di reset credenziali.

> *Figura 6: Schermata di login.*

Raccomandazione: Aggiungere un caso d'uso dedicato (es. "UC-X Reset Password") che descriva pre-condizioni, flusso principale (richiesta reset, invio email/SMS, cambio password), flussi alternativi (email non esistente, link scaduto) e post-condizioni. In alternativa, se la funzionalità non è implementata, rimuovere o marcare come non attiva la voce "Forgot password?" dal mockup.

REQ-017 — MEDIUM — Pagina 14 Il diagramma di navigazione mostra schermate come "Transaction History" e "Task Detail" che non hanno casi d'uso testuali espliciti (solo sono implicite nei controller/servizi), creando buchi di specifica funzionale.

> *Figura 5: Flusso di navigazione dell'applicazione.*

Raccomandazione: Per le schermate principali che rappresentano funzionalità autonome (es. Visualizzare lo storico transazioni, Visualizzare dettaglio task), introdurre casi d'uso separati (es. "View Transaction History", "View Task Details") con i relativi campi standard (pre/post-condizioni, flussi, test). Collegare questi UC ai test esistenti come CustomerServiceTest.getTransactionHistory o WorkerControllerTest.getTaskDetails se presenti.

7.2 Architettura (4 problemi)

ARCH-001 — LOW — Pagina 21 Il Domain Model mostra enumerazioni e tipi (MachineType.coffee, snack, pizzaOneForcito; ItemType.expressBeverage) che non sono considerati nei requisiti o nei casi d'uso, creando funzionalità potenziali non specificate a livello di analisi.

> *MachineType enumeration - coffee, snack, pizzaOneForcito - ItemType enumeration - snack, bottle, expressBeverage*

Raccomandazione: O integrare nei requisiti funzionali (sezione 2 e 4) la distinzione di comportamento per i diversi MachineType/ItemType (es. regole speciali per macchine pizzaOneForcito o per expressBeverage), oppure, se al momento non sono rilevanti, documentare chiaramente che si tratta di estensioni future e che l'attuale release tratta tutti i tipi in modo uniforme. In questo modo si evita ambiguità tra ciò che il sistema fa davvero e ciò che il modello suggerisce.

ARCH-002 — MEDIUM — Pagina 27 La descrizione della gestione transazioni nel database evidenzia una logica dettagliata (transaction e transactionitem) che non è completamente riflessa nei casi d'uso "Buy Item" e "Recharge Balance", creando una disallineamento tra logica di persistenza e requisiti utente.

> *Le transazioni degli utenti sono gestite attraverso due tabelle: • transaction: Intestazione della transazione (es. customer_ id, paymentmethod, initialbalance, updatedbalance). • transactionitem: Dettaglio della transazione, realizzando una relazione molti-a-molti tra transazioni e articoli. Ogni record collega un transaction_ id a un item_ id e specifica l'importo (amount).*

Raccomandazione: Aggiornare i casi d'uso relativi ad acquisto e ricarica per includere chiaramente la creazione di transaction e transactionitem come parte del flusso principale (ad esempio passi espliciti "il sistema registra la transazione con il metodo di pagamento X" e "il sistema registra ogni articolo acquistato come TransactionItem"). Questo renderà i requisiti allineati con la struttura del database e con i test TransactionServiceTest.

ARCH-003 — LOW — Pagina 22 Il documento dichiara espressamente l'uso della composizione al posto dell'estensione nel modello UML, ma nel diagramma sono presenti entità che estendono 'app_user' nel database (tabelle admin, worker, customer) e una gerarchia di sottotipi user-role nel domain model; la coerenza tra il principio dichiarato e l'implementazione effettiva non è chiarita.

> *Composizione al posto di estensione Nel nostro modello UML abbiamo scelto di legare le entità tramite composizione, anziché trami- te estensione, per sottolineare rapporti «parte–tutto» chiari e preservare l'incapsulamento delle responsabilità. Ad esempio, una ConcreteVending-Machine possiede un oggetto Inventory per rappresen- tare il suo magazzino interno: non sarebbe corretto modellare l'inventario come sottoclasse della macchina, poiché esso non è una specializzazione della macchina stessa, bensì un suo componente intrinseco, vincolato al suo ciclo di vita.*

Raccomandazione: Esplicitare nel testo la distinzione tra i casi in cui si è comunque scelto di usare l'estensione (ad esempio per la gerarchia degli utenti o per la struttura 'app_user' + tabelle specifiche) e quelli in cui si applica rigorosamente la composizione, spiegando i criteri adottati. In aggiunta, aggiornare il diagramma UML evidenziando chiaramente dove si deroga al principio "composition over inheritance" e motivare queste eccezioni.

ARCH-004 — LOW — Pagina 25 La sezione di business logic dichiara una separazione netta tra Controller e Service, ma in 'CustomerService' sono presenti metodi che mescolano responsabilità di dominio e di presentazione (es. gestione esplicita di Connection e inventario, oltre al bilancio), mentre il caso d'uso Buy Item prevede anche la gestione della disconnessione; la responsabilità di tale disconnessione non è chiaramente attribuita a service o controller, creando un potenziale scostamento dal principio enunciato.

> *L'implementazione della business logic del sistema è stata progettata secondo un approccio modulare, adottando i seguenti principi architetturali e di progettazione: • Separazione netta tra Controller e Service: la logica applicativa è interamente delegata ai Service, permettendo ai Controller di occuparsi esclusivamente della gestione delle richieste e delle risposte verso il client.*

Raccomandazione: Rendere esplicita, nel testo e nei diagrammi, la responsabilità della gestione della disconnessione dal punto di vista architetturale (es. metodo 'disconnect' del 'CustomerService' richiamato dal controller al termine di UC-3). Aggiornare eventualmente les firme dei metodi nei controller per riflettere che la logica di business (inclusa la chiusura della Connection) risiede nel service, lasciando al controller solo il mapping delle richieste/risposte.

7.3 Testing (8 problemi)

UC-4

TST-001 — HIGH — Pagina 10 Il caso d'uso UC-4 Connect to Vending Machine prevede flussi alternativi per macchinetta già connessa e fuori uso, ma la lista dei test non include scenari che verifichino questi errori, creando un gap sui controlli di stato della connessione.

> *Alternative Flow • Vending machine già connessa: errore. • Vending machine fuori uso: errore. Test Vedi i test correlati*

Raccomandazione: Aggiungere test espliciti per gli scenari di errore nella connessione alla vending machine, ad esempio 'CustomerServiceTest.testConnectMachineAlreadyConnected' e 'CustomerServiceTest.testConnectMachineOutOfService', e relativi test nel controller, verificando che venga restituito l'errore previsto e che non venga creata una nuova connessione in questi casi.

UC-8

TST-002 — HIGH — Pagina 13 Il caso d'uso UC-8.1 Create New Vending Machine specifica flussi alternativi per campi mancanti/errati ed errori di salvataggio, ma la tabella dei test per questo caso d'uso copre solo la creazione con dati validi e errori di null/DAO, non la validazione dei dati di input né errori transazionali di persistenza.

> *Alternative Flow 3.1 Campi mancanti o errati: errore. 4.1 Errore nel salvataggio: messaggio e richiesta di riprovare. Test Vedi i test correlati*

Raccomandazione: Integrare la suite con test per la validazione dei dati e la gestione degli errori di persistenza, ad esempio 'AdminServiceTest.testCreateMachineWithInvalidFields_throwsValidationExcep 'AdminControllerTest.createMachine_MissingFields_ReturnsError', e un test di integrazione 'ConcreteVendingMachineDaoImplTest.createMachine_persistenceError_rolledBack()' che simuli un errore di salvataggio e verifichi il corretto rollback.

Vedi anche: REQ-013

UC-3

TST-003 — MEDIUM — Pagina 9 Il caso d'uso UC-3 Buy Item prevede la disconnessione automatica del customer al termine del flusso principale e in caso di saldo insufficiente, ma nei test elencati per Buy Item non compare alcun test che verifichi esplicitamente la disconnessione o la corretta chiusura della Connection.

> *Post-conditions Il prodotto è stato erogato e il saldo aggiornato. Basic Flow 1. Il customer si trova nell'interfaccia del catalogo prodotti (Fig. 10). 2. Seleziona un prodotto. 3. Il sistema controlla saldo e disponibilità. 4. Il sistema scala il saldo ed eroga il prodotto. 5. Il customer viene disconnesso. Alternative Flow 3.1 Saldo insufficiente: errore e disconnessione. 3.2 Prodotto esaurito: errore. Test Vedi i test correlati*

Raccomandazione: Aggiungere test che verifichino esplicitamente la gestione del ciclo di vita della connessione, ad esempio 'CustomerServiceTest.testBuyItemSuccess_disconnectsCustomer'

e ‘CustomerServiceTest.testBuyItemInsufficientBalance_disconnectsCustomer‘, controllando che eventuali oggetti Connection vengano chiusi/rimossi e che il controller richiami il metodo ‘disconnect‘ previsto in ‘CustomerService‘.

UC-5

TST-004 — HIGH — Pagina 10 Per il caso d’uso UC-5 Recharge Balance viene modellato un flusso alternativo per pagamento non riuscito, ma i test associati alla ricarica si concentrano su update del saldo e creazione transazione; non c’è un test che simuli un fallimento del pagamento a valle della scelta del metodo e importo, lasciando scoperta la logica di annullamento/rollback.

> *Basic Flow 1. Il customer si trova nella propria pagina personale (Fig. 9a). 2. Sceglie metodo di pagamento e importo. 3. Il sistema verifica e processa la transazione. Alternative Flow 3.1 Pagamento non riuscito. Test Vedi i test correlati*

Raccomandazione: Introdurre un test come ‘CustomerControllerTest.testRechargeBalance_PaymentFailed che usi un mock di ‘TransactionService‘ (o del provider di pagamento) per simulare un fallimento e verifichi che il saldo non venga aggiornato e che venga restituito un errore appropriato, oltre a un test di servizio ‘CustomerServiceTest.testUpdateBalancePaymentFailed_doesNotChangeBalance‘.

UC-7

TST-005 — MEDIUM — Pagina 12 Per UC-7 View Analytics è previsto un flusso alternativo di errore di caricamento, ma i test elencati per questo caso d’uso coprono solo il recupero di liste e il caso DAO null; non verificano come il controller traduca un errore di servizio in un messaggio di errore per l’utente (flusso alternativo 3.1).

> *Alternative Flow 3.1 Errore di caricamento: messaggio di errore. Test Vedi i test correlati*

Raccomandazione: Aggiungere test per il controller che coprano il flusso di errore, ad esempio ‘AdminControllerTest.viewAnalytics_ServiceThrows_ReturnsErrorMessage‘, verificando che un’eccezione sollevata da ‘AdminService.viewAnalytics‘ produca la risposta corretta (HTTP status o messaggio di errore) e non un crash silente.

UC-6

TST-006 — MEDIUM — Pagina 11 Nel caso d’uso UC-6 Mark task as completed il flusso alternativo prevede un errore nel salvataggio con relativo messaggio, ma nei test manca un’integrazione end-to-end che verifichi il comportamento dell’intero stack (controller-service-DAO) in caso di fallimento del salvataggio sul database.

> *Alternative Flow • Errore nel salvataggio: messaggio di errore. Test Vedi i test correlati*

Raccomandazione: Estendere i test di integrazione introducendo un caso come ‘TaskDaoImplTest.integration_updateTask_persistenceError_throws‘ e un test controller ‘WorkerControllerTest.taskCompleted_persistenceError_returnsErrorMessage‘, forzando un errore del DAO (es. vincolo violato) e verificando che il servizio propaghi correttamente l’errore e che il controller restituisca il messaggio previsto dal flusso alternativo.

Problemi Generali

TST-007 — LOW — Pagina 8 Per UC-2 User Registration il flusso alternativo 4.1 prevede un errore di autenticazione dopo la creazione dell’account, ma nei test elencati per il caso d’uso di sign up non esiste un test che copra questa specifica condizione di errore post-creazione.

> *Alternative Flow 3.1 Campi errati o vuoti: il sistema mostra un messaggio di errore. 4.1 Errore di autenticazione: appare un messaggio che invita a ripro- vare più tardi. Test Vedi i test correlati*

Raccomandazione: Aggiungere un test specifico, ad esempio 'UserServiceTest.signUpAuthenticationError' o 'UserControllerTest.signUp_AuthError_ReturnsErrorMessage', che simuli un errore nella fase di autenticazione automatica successiva alla creazione dell'utente e verifichi la restituzione del messaggio conforme al flusso 4.1.

TST-008 — LOW — Pagina 7 Per il caso d'uso UC-1 User Login è previsto un flusso alternativo 4.1 di errore di sistema con messaggio generico, ma nei test elencati per il login non è presente un caso che simuli un errore interno (es. DAO che lancia eccezione) e verifichi la corretta gestione di questo scenario a livello di controller.

> *Alternative Flow 3.1 Credenziali errate: messaggio di errore che invita a riprovare. 4.1 Errore di sistema: messaggio di errore che invita a riprovare più tardi. Test Vedi i test correlati*

Raccomandazione: Integrare il set di test con un caso come 'UserControllerTest.login_ServiceThrows_Retu in cui 'UserService.login' viene configurato per lanciare un'eccezione; il test deve verificare che il controller non esponga l'eccezione grezza ma restituisca un messaggio/HTTP status coerente con il flusso alternativo 4.1.

8 Raccomandazioni Prioritarie

Le seguenti azioni sono considerate prioritarie:

1. **TST-001** (pag. 10): Aggiungere test espliciti per gli scenari di errore nella connessione alla vending machine, ad esempio 'CustomerServiceTest.
2. **TST-002** (pag. 13): Integrare la suite con test per la validazione dei dati e la gestione degli errori di persistenza, ad esempio 'AdminServiceTest.
3. **TST-004** (pag. 10): Introdurre un test come 'CustomerControllerTest.testRechargeBalance_PaymentFail che usi un mock di 'TransactionService' (o del provider di pagament...