

Università degli Studi di Salerno
Corso di Ingegneria del Software

**AniTour
System Design
Versione 1.2**

AniTour

Data: 21/01/2026

Progetto: AniTou	Versione: 1.2
Documento: System Design	Data: 21/01/2026

Indice

1. Introduction
 - 1.1 Purpose of the system
 - 1.2 Design goals
 - 1.3 Definitions, acronyms and abbreviations
 - 1.4 References
 - 1.5 Overview
2. Current software architecture
 - 2.1 Analisi dello stato attuale (As-is)
 - 2.1.1 Limitazioni tecniche critiche
 - 2.2 Architettura software di riferimento (To-be)
3. Proposed software architecture
 - 3.1 Overview
 - 3.2 Subsystem decomposition
 - 3.3 Hardware/software mapping
 - 3.4 Persistent data management
 - 3.5 Access control and security
 - 3.6 Global software control
 - 3.7 Boundary conditions
4. Subsystem services
 - 4.1 User Management Service
 - 4.2 Catalog Service
 - 4.3 Booking Service
5. Glossary

1. Introduction

1.1 Purpose of the system

Lo scopo del sistema AniTour è fornire una piattaforma di e-commerce specializzata nella vendita di tour tematici dedicati alla cultura nerd (anime, videogiochi, cultura pop giapponese). Il sistema permette agli utenti (Guest e Clienti) di scoprire e acquistare pacchetti viaggio, e agli Organizzatori di creare e gestire le proprie offerte turistiche. AniTour funge da intermediario centralizzato per semplificare la prenotazione, garantendo qualità e sicurezza nelle transazioni.

1.2 Design goals

Gli obiettivi di progettazione sono derivati dai requisiti non funzionali (RNF) identificati nel RAD. La tabella seguente definisce le priorità e il collegamento con i requisiti.

ID	Design Goal	Descrizione	Priorità	Reference
DG-01	Usabilità	L'interfaccia deve essere intuitiva per utenti non tecnici e responsive su mobile, facilitando la scoperta dei tour	Alta	RNF 3.3.1 (Layout fluido e responsive)
DG-02	Prestazioni	Il sistema deve garantire tempi di risposta < 2s anche con 500 utenti concorrenti durante i picchi di prenotazione	Alta	RNF 3.3.3 (Gestione carico)
DG-03	Sicurezza	Protezione dei dati sensibili (GDPR) e gestione sicura delle transazioni simulate	Alta	RNF 3.3.8 (Legal/Privacy)
DG-04	Affidabilità	Il sistema deve garantire la consistenza dei dati delle prenotazioni (ACID) evitando overbooking	Alta	RNF 3.3.2 (Reliability)
DG-05	Manutenibilità	L'architettura deve essere modulare (ECB) per permettere l'aggiunta futura di nuovi tipi di tour senza rifattorizzare il core	Media	RNF 3.3.4 (Supportability)

1.3 Definitions, acronyms and abbreviations

- **ECB:** Entity-Control-Boundary (Pattern architettonicale utilizzato).
- **DAO:** Data Access Object (Pattern per l'accesso ai dati).
- **DBMS:** Database Management System.
- **API:** Application Programming Interface.

- **Voucher:** Documento PDF generato post-acquisto.

1.4 References

- Requirement Analysis Document – AniTou v1.2
- Problem Statement – AniTou v1.1

1.5 Overview

Questo documento descrive l'architettura software di AniTou. Il sistema è decomposto in sottosistemi logici basati sul pattern ECB, mappati su un'architettura fisica client-server a 3 livelli (3-tier). Vengono dettagliate le strategie per la gestione dei dati persistenti, il controllo degli accessi e la gestione del flusso globale.

2. Current software architecture

Poiché il progetto AniTou nasce come soluzione *greenfield*, non esiste un sistema software legacy monolitico o integrato da sostituire o rifattorizzare. Tuttavia, è possibile analizzare l'attuale ecosistema tecnologico frammentato con cui viene gestito il dominio del turismo nerd e definire l'architettura di riferimento adottata per il nuovo sviluppo.

2.1 Analisi dello stato attuale (As-is)

Attualmente i processi di organizzazione e prenotazione di tour tematici avvengono attraverso canali eterogenei e non integrati:

- **Discovery:** La ricerca avviene tramite thread su forum verticali, gruppi social (Facebook, Instagram, TikTok) o siti web di singole agenzie non specializzate.
- **Gestione Dati:** Gli organizzatori gestiscono le adesioni tramite strumenti di office automation (fogli di calcolo Excel condivisi, Google Forms) o scambi di e-mail manuali.
- **Transazioni:** I pagamenti avvengono spesso off-platform (bonifici diretti, ricariche), senza un gateway integrato che garantisca l'atomicità della transazione.

2.1.1 Limitazioni tecniche critiche:

- **Assenza di integrità dei dati:** La gestione su file dispersi porta a ridondanza e inconsistenza (es. rischio di *overbooking* dovuto al mancato aggiornamento in tempo reale dei posti disponibili).
- **Sicurezza:** I dati sensibili degli utenti (PII) vengono spesso trattati su canali non protetti, violando i principi di *Privacy by Design* e le normative GDPR.
- **Scalabilità:** I processi manuali attuali non permettono di gestire picchi di traffico o un numero elevato di utenti concorrenti.

2.2 Architettura software di riferimento (To-be)

Per superare le limitazioni sopra descritte, il sistema AniTou adotta un'architettura *Client-Server a tre livelli (Three-Tier Architecture)*. Questa scelta

garantisce il disaccoppiamento tra la logica di presentazione, la logica di business e la gestione dei dati, facilitando la manutenibilità e la scalabilità (Design Goals DG.04 e DG.05).

Il sistema è strutturato nei seguenti tier logici e fisici:

1. **Presentation Tier (Client side):**

- Responsabile dell'interazione con l'utente finale (*Guest, Cliente, Organizzatore*).
- Realizzato tramite tecnologie web standard (*HTML5, CSS3, JavaScript*).
- Comunica con il server tramite richieste HTTP/HTTPS asincrone, garantendo un'esperienza utente fluida e responsive su dispositivi desktop e mobile.

2. **Application Tier (Server side):**

- Ospita la logica di business e i sottosistemi di controllo (*User Management, Catalog, Booking*).
- Implementato in linguaggio Java su Application Server *Apache Tomcat*.
- Gestisce l'autenticazione, la validazione degli input e il coordinamento delle transazioni. Espone interfacce per il frontend e funge da mediatore verso il database e i servizi esterni (*Payment Gateway, SMTP*).

3. **Data Tier (Persistence):**

- Costituito da un Database Relazionale (*MySQL*) che garantisce la persistenza e l'integrità dei dati.
- L'accesso ai dati è mediato esclusivamente dal layer di persistenza (pattern *DAO*) presente nell'*Application Tier*, assicurando che nessuna query SQL venga eseguita direttamente dal client.
- Supporta transazioni *ACID* per la gestione critica delle prenotazione e degli stock.

3. Proposed software architecture

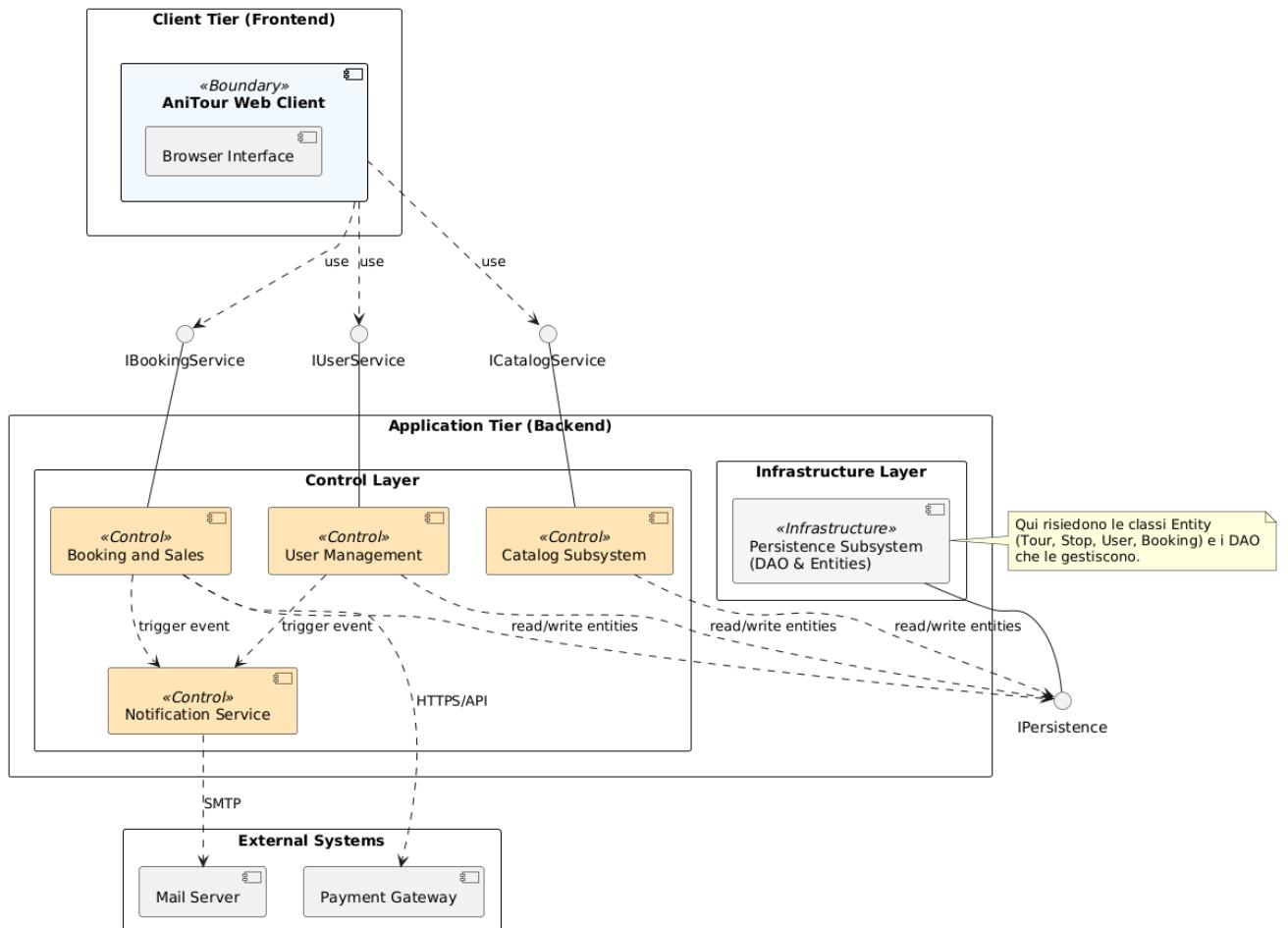
3.1. Overview

L'architettura proposta è basata sullo stile Client-Server a 3 livelli (3-Tier).

Logicamente, l'applicazione segue il pattern Entity-Control-Boundary (ECB) per garantire una chiara separazione delle responsabilità:

- **Boundary:** Gestisce l'interazione con l'utente.
- **Control:** Gestisce la logica applicativa e i casi d'uso.
- **Entity:** Rappresenta i dati di dominio.

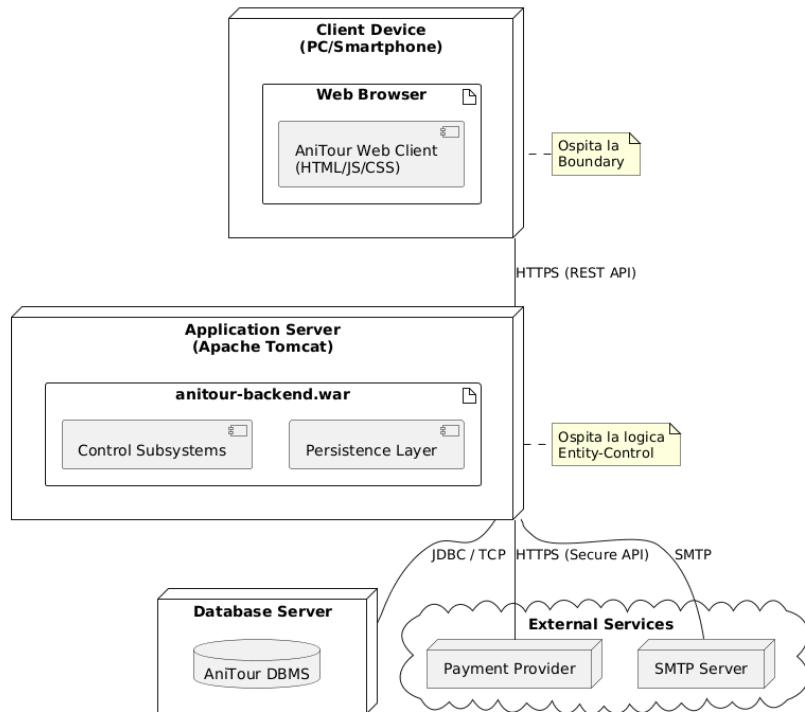
3.2. Subsystem decomposition



- 1. Presentation Subsystem (Boundary - com.anitour.boundary conceptual):**
 - Responsabilità:** Gestisce l'interfaccia utente tramite pagine JSP/HTML e risorse statiche (CSS/JS). Intercetta gli eventi utente e invoca i Controller appropriati. Contiene le classi Boundary (es. TourPage, CartPage, LoginForm)
 - Componenti:** AniTour Web Client, pagine `index.jsp`, `checkout.jsp`.
- 2. User Management Subsystem (Control):**
 - Responsabilità:** Gestisce la logica relativa agli utenti: autenticazione, registrazione, gestione profili.
 - Classi Chiave (Design):** AuthManager, UserManager..
- 3. Catalog Subsystem (Control):**
 - Responsabilità:** Gestisce la ricerca, il filtraggio e la visualizzazione dei dettagli dei tour. Permette agli organizzatori di creare/modificare tour e agli operatori di approvarli.

- **Classi chiave (Design):** CatalogManager, TourManager.
4. **Booking and Sales Subsystem (Control – com.anitour.control):**
- **Responsabilità:** Cuore operativo del sistema implementato. Gestisce il carrello (sessione), il flusso di checkout transazionale e la validazione delle regole di business.
 - **Classi chiave (Implementation):**
 - **CartManager:** Coordina la transazione di checkout e l'integrazione con il DAO.
 - **OrderManager:** Gestisce lo stato temporaneo della sessione utente (pattern Session Façade).
5. **Payment Subsystem (External Interface – com.anitour.dao):**
- **Responsabilità:** Interfaccia verso gateway di pagamento esterni. Gestisce la transazione finanziaria sicura. Gestisce la transazione finanziaria sicura
 - **Componenti:**
 - **IPaymentGateway:** Interfaccia Adapter per disaccoppiare il sistema dal provider reale
6. **Notification Subsystem (Service):**
- **Responsabilità:** Gestisce l'invio asincrono di e-mail transazionali (conferma registrazione, invio voucher PDF, notifiche stato tour).
7. **Persistence Subsystem (Infrastructure/Data – com.anitour.dao):**
- **Responsabilità:** Astrarre l'accesso al database relazionale garantendo l'integrità dei dati.
 - **Classi Chiave (Implementation):**
 - **BookingDAO:** Implementa le operazioni CRUD sulla tabella bookings.
 - **DatabaseConnection:** Gestisce il pool di connessioni JDBC (Singleton/Factory).
 - **IBookingRepository:** Interfaccia per l'inversione delle dipendenze.

3.3. Hardware/software mapping



La tabella seguente descrive lo stack tecnologico installato sui nodi fisici dell'architettura

Nodo	Componente Software	Tecnologie
Client Node	Web Browser	Chrome 120+, Firefox, Edge (Supporto HTML5/CSS3)
App Server Node	Web Container	Apache Tomcat 10.1 (Jakarta EE)
	Runtime Environment	Java OpenJDK 17
	Framework and Libraries	Servlet API 6.0, JSTL, JUnit 5 & Mockito (Testing)
Database Node	DBMS	MySQL Server 8.0

3.4. Persistent data management

La persistenza è gestita tramite un **Database Relazionale**, scelto per garantire integrità referenziale e consistenza nelle transazioni finanziarie (ACID).

- **Mapping:** Le classi *Entity* (User, Tour, Booking, ecc. descritte nel RAD) vengono mappate su tabelle relazionali. Il sito deve essere pienamente operativo su browser moderni (Chrome, Firefox, Edge, ecc...).
- **Ruolo dei DAO:** I metodi di accesso ai dati (CRUD) risiedono nelle classi DAO all'interno del *Persistence Subsystem*. Le classi *Control* invocano i DAO per rendere persistenti le *Entity*. Non vi è logica SQL all'interno delle Entity o delle Boundary.
- **Sessione:** Il carrello (Cart) è mantenuto volatile nella sessione HTTP o in uno store temporaneo (es. Redis) finché non viene convertito in ordine (Booking).

Classe Java (com.anitour.model)	Tabella DB (schema.sql)	Mapping Attributi Principali
User	users	id -> id (PK) email -> email (Unique) password -> password_hash
Tour	tours	id -> id (PK) price -> price seatsAvailable -> seats_available
Booking	bookings	id -> id (PK) status -> status (ENUM) total -> amount
Cart	N/A (<i>In-Memory</i>)	I dati del carrello risiedono nella HttpSession. Vengono salvati in bookings solo al checkout.

3.5. Access control and security

L'accesso è controllato tramite una *Access Control List (ACL)* basata sui ruoli definiti nel RAD:

Funzionalità / Risorsa	Guest (Non Loggato)	Cliente	Organizzatore	Operatore
Visualizza Catalogo	✓	✓	✓	✓
Ricerca Tour	✓	✓	✓	✓
Aggiungi al Carrello	✓	✓	✗	✗
Checkout (Pagamento)	✗ (Richiede Login)	✓	✗	✗
Crea/Modifica Tour	✗	✗	✓	✗
Approva/Rifiuta Tour	✗	✗	✗	✓
Gestione Utenti (Ban)	✗	✗	✗	✓

3.6. Global software control

Il flusso di controllo è *Event-Driven*, tipico delle web-app moderne.

1. L'utente interagisce con la *Boundary* (click su un pulsante).
2. La *Boundary* invia una richiesta HTTP al server.
3. Il *Control* appropriato intercetta la richiesta, elabora la logica di business, manipola le *Entity* e usa i *DAO* per la persistenza.
4. Il *Control* restituisce i dati alla *Boundary* per la visualizzazione della risposta.

Eccezione: Il sottosistema di notifica opera in modo asincrono per non bloccare l'interfaccia utente durante l'invio delle e-mail.

3.7. Boundary conditions

- **Inizializzazione:** All'avvio del server (StartServer), il sistema verifica la connessione al db e carica le configurazioni globali.
- **Terminazione:** Lo ShutDownServer chiude le connessioni al db e completa le transazioni pendenti prima di spegnersi.
- **Failure:**
 - *Db Down:* Il sistema mostra una pagina “Manutenzione in corso” e logga l'errore critico.
 - *Pagamento fallito:* Il sistema gestisce l'eccezione restituita dal Payment gateway, non crea l'ordine e informa l'utente dell'errore permettendo il retry.

4. Subsystem services

Di seguito sono definiti i servizi esposti dai sottosistemi principali, con le relative firme dei metodi (API Interne).

4.1. Booking Service (`BookingControl`)

Servizio responsabile della gestione transazionale degli ordini.

- `processCheckout(Cart cart, PaymentDTO payment)`: Esegue la transazione di acquisto.
 - *Input:* Carrello corrente, Dati carta.
 - *Output:* Oggetto Booking confermato.
 - *Exception:* `SoldOutException`, `PaymentFailedException`.

4.2. Repository Service (`IBookingRepository`)

Servizio di accesso ai dati per le prenotazioni.

- `save(Booking booking)`: Persiste una nuova prenotazione e restituisce l'ID generato.
- `checkAvailability(int tourId, int quantity)`: Verifica la disponibilità di posti in tempo reale (concorrenza).

4.3. Payment Service (`IPaymentGateway`)

Servizio esterno per l'autorizzazione finanziaria.

- `processPayment(double amount, PaymentDTO details)`: Contatta la banca per autorizzare l'addebito. Restituisce true se approvato.

5. Glossary

- **Boundary:** Componente che interfaccia il sistema con un attore esterno (UI).
- **Control:** Componente che orchestra la logica di un caso d'uso.
- **Entity:** Oggetto che rappresenta dati persistenti significativi per il dominio.
- **Tour:** Pacchetto viaggio tematico (Entity principale).
- **Booking:** Oggetto che traccia sia il carrello temporaneo che l'ordine confermato.
- **Guest:** Utente non autenticato.
- **Organizzatore:** Utente fornitore che crea i tour.

Revision History

Data	Versione	Descrizione	Autore
25/11/2025	1.0	Creazione SDD	Vincenzo Chiocca
04/12/2025	1.1	<p>Revisione Design Goals e Architettura Attuale:</p> <ul style="list-style-type: none"> - Ristrutturata la sezione 1.2 (Design Goals) in una tabella prioritaria collegata ai requisiti non funzionali presenti nel RAD. - Espansa la sezione 2 con l'analisi dello scenario As-is e delle limitazioni tecniche attuali. - Introdotta la descrizione dell'architettura To-be a 3 livelli. <p>TODO:</p> <ul style="list-style-type: none"> - Dettagliare la scomposizione dei sottosistemi (Sezione 3.2). - Specificare lo stack tecnologico in Hardware/Software mapping (Sezione 3.3). - Aggiungere la mappatura Classi-Tabelle DB in Persistent Data Management (Sezione 3.4). - Creare la Matrice degli Accessi (Sezione 3.5). - Definire le tabelle dei Servizi dei Sottosistemi (Sezione 4). 	Vincenzo Chiocca
21/01/2026	1.2	<p>Aggiornato il documento di System Design (SDD) risolvendo tutti i TODO in sospeso:</p> <ul style="list-style-type: none"> - Dettagliata la scomposizione dei sottosistemi (Sezione 3.2). - Specificato lo stack tecnologico (Java 17, Tomcat, MySQL) in HW/SW Mapping (Sezione 3.3). - Aggiunta tabella di mapping Classi-Database (Sezione 3.4). - Definita la Matrice di Controllo Accessi per i ruoli (Sezione 3.5). - Documentate le interfacce dei servizi (API Interne) per Booking, Repository e Payment (Sezione 4). 	

Partecipanti:

Nome	Matricola
Vincenzo Chiocca	0512119182
Salvatore Merola	0512120979

