Software Requirement Specification

Application Name	**FitCity – Find your perfect destination before you go**
Member	1. **Kornchanok Phattanasiri 66070503403** 2. **Kamin Jittapassorn 66070503409** 3. **Nutthawut Jaroenchokwittaya 66070503421** 4. **Pansa Intawong 66070503474** 5. **Phuwasin Kriangdechachai 66070503478** 6. **Corentin Navone 68540460048** 7. **Corentin Pivert 68540460063**

Document Version:

Date	Author	Details	Version
10 Aug 25	*Everyone*	***Create Document (Choose Topic, Project Scope, Project Theme & Problem Statement, Roles, and Activity Log Set Up)***	***1.0***
17 Aug 25	*Everyone*	***Identifying system actors, developing use case diagrams, and writing user stories with corresponding acceptance criteria.***	***1.1***
25 Aug 25	*Everyone*	***Identifying functional and non-functional requirements with SRS outline***	***1.2***
1 Sep 25	*Everyone*	***Extend SRS and translate our finalized Software Requirements Specification (SRS) into architectural models and UML diagrams (Class diagrams, Sequence diagrams, Activity diagrams, and Component Diagrams)***	***1.3***
14 Aug 25	*Everyone*	***Refine and finalize design diagrams and Create Package Diagram***	***1.4***
17 Aug 25	*Everyone*	***Create SRS Document for ONLY FitCity (Move to this file)***	***2.0***
20 Aug 25	*Everyone*	***Write SRS, Prioritization and Estimate Efforts, and MVP Scope***	***2.0***

Reference Information:

UX/UI Design	***Prototype_fitcity***
Prototype	***Prototype_fitcity***
Functional Specification	*- link -*
Non-Functional Specification	*- link -*
Technical Specification	*- link -*
Test Case	*- link -*
User Manual	*- link -*

Application Overview

Application Concept

The app helps users search for places they want to visit, showing real photos, reviews, and 5-star ratings. It highlights safety and conditions of the location, such as crime-prone areas, poor lighting, cleanliness, or homeless presence. The app also suggests nearby locations with ratings to help users plan their visit.

Application Specification

- Web Application developed using
    - HTML
    - JavaScript
    - Typescript
    - CSS
    - React
    - Vite
    - Gin framework
    - GO
    - PostgreSQL
- Compatible for browsers (Chrome)
- 6 key features
    - Destination Search
    - View Destination Details
    - Save/Unsave Favorites
    - Favorites List and Deep-link to Details
    - Update Destination Data
    - User accounts

System Requirement

Functional Requirement

- **Destination Search**
    - Search Bar
        - The system must display a search bar on the homepage and other relevant pages.
        - The search bar must be accessible at all times during the usage.

    - Keyword Search
        - The system must handle case-insensitive searches.
        - The system must allow users to search by entering keywords (city, province, landmark, etc.).
        - The system must provide auto-suggestions based on partial inputs (Ex. Typing “Siam” suggests “Siam Paragon”, “ICON Siam”, “Siam Square”, etc.).

    - Search Results Display
        - The system must display search results in a following format:
            - Destination Name
            - Ratings (5-stars format)
            - Thumbnail Image
            - Short Description
        - The system must sort search results by relevance by default.
        - The system must allow users to click on a result to view the destination detail page.

    - Filter and Sorting
        - The system must allow users to filter destinations by categories such as:
            - Type of destination (city, mall, beach, etc.)
        - The system must allow users to sort search results by:
            - Popularity
            - Alphabetical Order

    - No Results Handling
        - If no results are found, the system must display a message: “No destinations found. Please try different destinations.”.

    - Search History
        - The system must store the last 5 search queries made by the user and display them in the search bar dropdown.

- **View Destination Details**
    - Accessing Destination Details Page
        - The system must display a clickable button for each destination shown in search results, recommendations, or saved destinations.
        - The system must open the Destinations Details page when users select a destination.
        - Users without an account can only view destination details while logged in users have access to more features (Favorites, Rating, Reviews).

    - Display Destination Information
        - Destination details page must display structured information as follows:
            - Destination Name
            - Location
            - Gallery
            - Description of the destination
            - Categories (Type of destination)
            - Attractions and Activities

    - User Interaction
        - Save Destination: Logged-in users must be able to save a destination to “Favorites”
        - Ratings and Reviews: Logged-in users must be able to write and submit ratings and reviews.

    - Ratings and Reviews
        - The system must display average ratings for the destination.
        - The system must display user created reviews with reviewer’s name/username, date, text.

- **Save/Unsave Favorites**
    - Accessing Favorites Feature
        - The system must display a Save/Unsave button on each destination detail page.
        - The Save/Unsave button must be accessible from search results, recommendation lists, and the Favorites list.
        - The system must require the user to be logged in before saving or unsaving a destination. If not logged in, the system must redirect the user to the login page.

    - Saving a Destination
        - The system must allow users to save a destination to their Favorites list by clicking the Save button.
        - The system must prevent duplicate saves; if the destination is already saved, the button must change to Saved (or Unsave).
        - The system must display a confirmation message after a successful save (e.g., “Destination saved to Favorites”).
        - The system must update the button state to reflect that the destination is already in Favorites.

    - Unsaving a Destination
        - The system must allow users to remove a destination from their Favorites list by clicking the Unsave button.
        - The system must update the button state to reflect that the destination has been removed from Favorites.
        - The system must display a confirmation message after a successful unsave (e.g., “Destination removed from Favorites”).

    - Favorites List Management
        - The system must maintain a list of saved destinations for each user, linked to their account.
        - The system must update the Favorites list immediately after a save or unsave action.
        - The system must allow users to click on a saved destination in the Favorites list to view the full destination detail page.

    - Error Handling & Logging
        - If a save or unsave action fails (e.g., network/server error), the system must revert to the last confirmed state and display an error message (e.g., “Could not update Favorites. Please try again.”).
        - The system must record each save/unsave action in the Activity Log with user ID, destination ID, action type (save/unsave), and timestamp.

- **Favorites List and Deep-link to Details**
    - Access Favorites List
        - The system must provide a Favorites button in the main navigation bar.
        - The system must display search results in a following format:
            - Destination Name
            - Ratings (5-stars format)
            - Thumbnail Image
            - Short Description
        - The system must sort Favorites List in ascending order of the time added.

    - Managing Favorites
        - The system must allow users to remove destinations from their Favorites list.
        - The system must display a confirmation prompt before removing a destination.
        - Changes to the Favorites list must be updated in real time.

    - Deep-link Access
        - The system must generate a unique deep-link for each destination detail page.
        - When a deep-link is accessed, the system must open the corresponding details page.

    - Authentication Handling
        - If the user is not logged in, the system must display basic destination details (Without save to favorites and unable to submit review).
        - When a user clicks on a feature that requires logging in, the system must redirect users to the login page and then return them to the original deep-link destination page after successful login.

    - Error Handling
        - If the deep-link is invalid or does not exist, the system must display “Destination not found” page.

- **Update Destination Data**
  - Accessing Update Feature
    - The system must display an Update Destination option in the Admin CMS dashboard.
    - The system must require the user to be authenticated and authorized as an Admin before accessing update functions.
    - If the user is not an Admin, the system must display an Access Denied message.

  - Editing Destination Information
    - The system must allow Admins to edit existing destination information, including:
    - Destination Name
    - Description
    - Category/Type (city, mall, beach, etc.)
    - Media (images, videos)
    - Safety information (e.g., crime rate, lighting, cleanliness)
    - The system must allow Admins to upload new media or remove outdated media.
    - The system must provide a preview mode before publishing changes.

  - Saving & Publishing Updates
    - The system must validate input fields before saving (e.g., no empty name, valid media formats).
    - The system must allow Admins to save a draft version of the destination before publishing.
    - The system must record all published updates with a version number for rollback if needed.
    - The system must display a confirmation message after a successful update (e.g., “Destination updated successfully”).

  - Auditing & Logging
    - The system must record each update in the Activity Log with Admin ID, destination ID, type of update, and timestamp.
    - The system must maintain a change history for each destination, showing previous values and updated values.
    - The system must allow Admins to view past update history for transparency.

  - Error Handling
    - If an update fails (e.g., database error, invalid input), the system must display an error message (e.g., “Update failed. Please try again.”).
    - The system must not overwrite existing data until the update is confirmed as successful.

- **User accounts**
    - Register
        - The system must allow users to register a new account using username, email and password.
        - The system must validate email format and ensure the email is unregistered.
        - The system must enforce password restrictions (Minimum length of 12 characters, at least one of uppercase letter, lowercase letter, number, and special character).
        - The system must display “Terms and Conditions” and require the user to accept before creating an account.

    - Login
        - The system must allow users to login using username or email and password.
        - The system must authenticate users by validating their credentials against the stored records.
        - The system must display error messages for failed login attempts.
        - The system must provide a Forgot Password button.

    - Edit and Update Profile
        - Logged-in users must be able to access Profile page.
        - The system must display profile information as follows:
            - Username
            - Email Address (read-only)
            - Profile Picture
            - Role (read-only)
        - The system must allow users to edit and update their Username and Profile Picture.
        - The system must restrict uploaded profile pictures to file type JPG and PNG with max size limit.
        - The system must save and change immediately after user confirmation.
        - Users must not be able to edit their Email address and role.

    - Remove Account
        - Logged-in users must be able to permanently delete their account from the Settings page.
        - The system must display a warning and confirmation message about data loss before confirming.
        - When confirmed, the system must delete all of the user’s account data.
        - The system must display a confirmation message after deletion.

- User Roles and Role Management
  - The system must support different user roles. (Regular user, Administrator
  - Role assignment must be performed by Administrator only.
  - Administrator must be able to:
    - Assign roles when creating a new user.
    - Update roles for existing users.
    - Remove user accounts.
  - The system must enforce role-based access control (RBAC):
    - Regular users can only access general features (Search, favorites, profile, ratings and reviews).
    - Administrators have full control over managing users, assigning roles, moderating contents.

Non-Functional Requirement

1. Destination Search

Search Bar

- Security
    1. Input Sanitization & Validation; All query inputs must be validated and sanitized server-side to prevent injection.
    2. Transport & Data Protection; All requests/responses must use TLS 1.2+; no mixed content. Search telemetry/logs must not store raw PII or full tokens.
    3. Abuse & Rate Limiting; Implement IP+account rate limits and bot detection on search endpoints.
    4. Error Handling; Errors do not reveal stack traces, queries, or schema details.
- Performance
    1. Return search results quickly under normal load. ≤ 1.5 s for queries with ≤3 filters
    2. Cache frequent queries and type-ahead suggestions.
- Usability
    1. Discoverability & Feedback; The search bar is visible on all relevant pages and provides immediate feedback (loading indicator, empty-state hints).
    2. Typo Tolerance & Suggestions; Support minor misspellings
    3. Responsiveness; work on mobile view and desktop view

Search Result

- Security & Privacy
    1. Only items the user is authorized to view appear in results/snippets.
    2. No Sensitive Data in UI/Logs
    3. Failures show generic messages and correlation IDs, never stack traces or query internals. E.g. 4xx/5xx error code
- Performance
    1. Image load within 1 sec with stable internet connection.
    2. Sort/filter/apply or clear facets updates UI ≤ 300ms
    3. Next page fetch ≤ 1 sec & maintains scroll position and focus.
- Usability
    1. Readability & Density; Each result shows title, information, picture.
    2. Empty, No-Match, and Error States; Provide helpful next steps.
    3. Accessibility; Fully keyboard-operable (Tab, Enter, Arrow)
    4. Responsiveness; work on mobile view and desktop view
    5. Consistent Card Layout; Visual regression tests pass across top 20 result templates.

2. View Destination Details

Destination Detail

- Security & Privacy
    1. Only public fields shown to anonymous users; user‑specific saved states require login.
    2. PII Minimization; No raw emails/phone numbers of reviewers or hosts in UI or logs
    3. Third-Party SDK Bounds; Map/analytics SDK keys restricted by domain and quota; no query strings containing tokens.
- Performance
    1. Image preload.
    2. Details API ≤ 500 ms, reviews ≤ 700 ms, user saved states ≤ 400 ms.
- Usability
    1. Information Hierarchy: Title → rating → price range/availability→ actions (save, share, book) → key facts (location, best season)
    2. Scannability: Specs in labeled bullets (climate, activities, amenities); reviews show highlights + filters.
    3. Media UX: Gallery supports swipe/keyboard; zoom with pinch/double-tap; video playback without autoplay audio.

3. Save/Unsave Favorites

Save Favorites

- Security & Privacy
    1. Only authenticated users can save; operation scoped to caller's tenant.
    2. Logs contain opaque item IDs and hashed user IDs; no titles/PII.
- Performance
    1. API Latency < 500 ms
    2. Burst Handling; Sustain ≥ 10 writes/sec/user for 5s without breaching
- Reliability & Data Integrity
    1. Repeating the same save (same user+item) does not create duplicates; return 200/204 consistently.
    2. Uniqueness: Enforced via UNIQUE(user_id, item_type, item_id) at storage.
    3. Consistency: Read-after-write for caller ≤ 1 s; cross-device propagation ≤ 5 s.
    4. Rate Limit; 10 writes/sec/user
- Usability
    1. Icon button update status within 100 ms

Unsave Favorite

- Security & Privacy
    1. Only authenticated users can save; operation scoped to caller's tenant.
    2. Logs contain opaque item IDs and hashed user IDs; no titles/PII.
- Performance
    1. API Latency < 500 ms
    2. Burst Handling; Sustain ≥ 10 writes/sec/user for 5s without breaching
- Reliability & Data Integrity
    1. Repeating unsave on an already-removed item is safe (200/204 or 404 by contract); no errors or side effects.
    2. Cascade Safety: Removing favorites must not delete the underlying destination or shared resources; only the relation row is removed. If the place is removed, the favorites record must be removed automatically.
- Usability
    1. Icon un-fills within 100 ms
    2. Multi-select unsave with clear, non-destructive confirmation for "Remove all".

4. Favorites List and Deep-link to Details

Search Result

- Security & Privacy
    1. Only the owner can view/manage their list; no cross-tenant leakage.
    2. No Sensitive Data in UI/Logs
    3. Failures show generic messages and correlation IDs, never stack traces or query internals. E.g. 4xx/5xx error code
    4. Rate Limiting: Reads ≥ 20 req/min/user
- Performance
    1. Image load within 1 sec with stable internet connection.
    2. Sort/filter/apply or clear facets updates UI ≤ 300ms
    3. Next page fetch ≤ 1 sec & maintains scroll position and focus.
    4. Read-after-write: Saves/unsaves reflect on the list for the caller ≤ 1s; cross-device ≤ 5s.
- Usability
    1. Readability & Density; Each result shows title, information, picture.
    2. Empty, No-Match, and Error States; Provide helpful next steps.
    3. If thumbnails fail, show placeholders; if an item is deleted/unavailable, show disabled card with reason and action to remove.
    4. Multi-select remove, sort (recently added, A-Z, rating), filters (type, tags).

Deep-link to Details

- Link Integrity & Addressability
    1. Stable URLs: Canonical, version-independent path.
    2. Opaque IDs: Links never expose internal sequential IDs alone; use slugs or ULIDs.
- Security & Access Control
    1. Auth-Gated Content; If details require login/role, redirect to auth and return to the original deep-link post-login.
    2. Tenancy Enforcement: Multi-tenant items respect org visibility; 403/404 as configured.
- Behavior for Unavailable Items
    1. Deleted/Unlisted: Show 410 (gone) or 404 with user-friendly message and link back to Favorites; offer to remove stale favorites.
- Performance & UX
    1. From Favorites list, hover/touchstart prefetch of next possible details page when bandwidth idle.
    2. Arrival From External Apps; First contentful paint ≤ 1.6s

5. Update Destination Data

Add Page

- Security & Privacy
    1. Only roles Admin can add.
    2. Validate and sanitize all fields.
    3. No plaintext API keys or personal contacts in payload/logs.
- Performance
    1. Form Load < 1.2 s
    2. Create API < 500 ms (excluding media upload).
    3. Media Upload direct to blob; handshake
- Reliability & Data Integrity
    1. Re-submitting the same draft with identical client idempotency key creates 1 record.
    2. Server-side rejects invalid geos
    3. Metadata + image references committed atomically; partial failures roll back.
- Usability
    1. Field-level errors within 100 ms after blur; no page reloads.

Update Page

- Security & Privacy
    1. Only roles Admin can update.
    2. Only editable fields accepted; immutable IDs guarded.
- Performance
    1. Page Load < 1.2 s
    2. PATCH/PUT: ≤ 350 ms for metadata-only; ≤ 700 ms if media references change.
- Reliability & Data Integrity
    1. If-Match/version. On conflict return 409 with diff preview.
    2. 100% conflicting edits blocked without data loss.
- Usability
    1. Highlight unsaved edits; “Save” disabled until changed; confirm on navigate-away.

Remove Page

- Security & Privacy
    1. Only roles Admin can remove.
    2. Every record is soft-delete.
- Performance
    1. Delete API < 400 ms
- Reliability & Data Integrity
    1. Delete only the destination record; do not remove shared entities (images, reviews).
    2. Related indexes/docs receive tombstone updates within ≤ 60 s.
    3. Restore soft-deleted item in ≤ 10 s; retains original ID and relations.
- Usability

1. Confirmation UX: Clear impact summary (e.g., “Hidden from search, N reviews remain”).
2. Undo: Immediate undo for 10 s after soft delete.
3. Accessibility: Dialog is focus-trapped, ESC/Enter operable.

Approve Page

- Security & Privacy
    1. Users that are not action makers and roles Admin can be approver.
    2. Hidden fields in pending items remain invisible to non-approvers.
- Performance
    1. Approve/Reject Action: ≤ 300 ms
    2. Moderation queue list 600 ms for 50 items with filters.
- Reliability & Workflow Integrity
    1. Valid transitions only: draft → submitted → approved|rejected, delete → submitted → approved|rejected, update → submitted → approved|rejected
       No skipping states.
    2. Approval binds to content hash/version; if submitter edits after review started, require re-approval.
    3. Other actions can not be made if that content is waiting for review.
    4. On approve action apply within 30 sec
    5. On rejection, notify the submitter with reasons.
- Usability
    1. Diff view vs. last approved version; flagged fields highlighted; quick actions (approve, reject, request changes).

6. User accounts

Register (Sign-up)

- Security
    1. Password & Secret Handling: Server‑side hashing.
    2. No plaintext secrets in logs; hash config verified in code scan.
    3. Bot & Abuse Controls; CAPTCHA /account rate limits.
    4. PII Minimization: Collect only necessary fields
    5. Sign-up attempts, completion rate, verification failures, spam blocks.
- Performance
    1. Page Load < 1 sec
    2. Create API: P95 ≤ 350ms (excluding email/SMS send).
- Usability
    1. Field errors within 100ms after blur; password strength meter.

Login (Sign-in)

- Security
    1. Transport: TLS 1.2+; HSTS; secure, HttpOnly, SameSite=strict cookies.
    2. Brute-force Defense: Per-IP+account rate limit; progressive backoff; lockout notifications.
- Performance
    1. Page Load < 1 sec
    2. Auth API ≤ 350ms
- Usability
    1. Clear Errors: Generic messages (“Invalid credentials”); correlation ID provided.

View Account (Profile Read)

- Security
    1. Only owner (and Admin where policy allows) can view
    2. Never display secrets (hashes, tokens); partial masking for identifiers.
- Performance
    1. Profile Read API: ≤ 200ms

Update Account (Profile Edit)

- Security & Privacy
    1. Sensitive changes (email, phone, password, etc.) require re-auth.
    2. Only editable fields accepted; immutable IDs guarded
- Performance
    1. PATCH/PUT: ≤ 300ms (metadata)
    2. Picture upload ≤ 3 sec (picture)
- Usability & Accessibility
    1. Inline Validation & Save Feedback: Success toast ≤ 100ms.

Remove Account Deletion

- Security
    1. User self delete requires re-auth.
    2. Admin delete other users require approval from other admin.
    3. purge PII and tokens
- Performance
    1. Background purge jobs finish ≤ 24h with progress status.
- Reliability & Integrity
    1. Remove/anonymize dependent objects according to policy (favorites, reviews). No orphaned references.

Give Role (Assign / Revoke Roles & Permissions)

- Security
    1. Least Privilege: Roles are predefined; permissions are not assignable ad-hoc in UI.
    2. Only Admin can assign/revoke.
    3. Require approval from other admin.
- Performance
    1. Grant/Revoke API: ≤ 250ms
    2. Changes propagate to policy cache ≤ 60s.
- Reliability & Integrity
    1. Active sessions pick up reduced privileges within ≤ 60s or on next token refresh; immediate revoke for critical roles (session kill).
    2. Auditability: Append-only log with actor, target, role, justification. Retention ≥ 24 months.
- Usability & Accessibility
    1. Review UX: Show current roles, effective permissions, and blast radius before confirming.

SRS

1. Introduction

1.1 Purpose

FitCity is designed to help travelers search, view, and save information about destinations before visiting. Admins can update destination data and export statistics to track popularity. This SRS defines all functional and non-functional requirements, use cases, outcomes, and activity logs to ensure clarity for developers, testers, and stakeholders.

1.2 Scope

FitCity is a standalone web and mobile application providing:

- Search for destinations.
- View detailed destination profiles (price, culture, activities, climate).
- Save and view favorite destinations.
- Admin update of destination data.
- Export destination popularity statistics.
- Cross-device and cross-browser compatibility.

1.3 Definitions

- **Traveler:** End-user searching and viewing destinations.
- **Admin:** Authorized user managing destination information.
- **Destination Database:** Stores all destination details.
- **Favorites:** Destinations saved by the user for later reference.
- **PII:** Personally Identifiable Information.
- **Activity Log:** Records user/admin actions with timestamps.
- **Outcome:** Result visible to the user after performing an action.

2. Overall Description

2.1 Product Perspective

FitCity is a standalone system with front-end (web/mobile app), back-end server, and destination database. It provides fast, accurate, and user-friendly access to travel information.

2.2 User Characteristics

- **Travelers:** Non-technical, require easy navigation and quick search results.
- **Admins:** Technical staff authorized to edit and manage destination data.
- **Database:** Provides accurate destination information for system queries.

3. Functional Requirements & Use Cases

Use Case 1: View Destination Details

Primary Actor: Traveler
Precondition: Search result or favorite destination is selected.
Main Flow:

1. Traveler taps on destination.
2. The system displays a detailed profile: price, culture, activities, climate.

Use Case 2: Remove Favorite Destination

Primary Actor: Traveler
Precondition: Destination is in favorites.
Main Flow:

1. Traveler taps “Remove from favorites.”
2. System updates User Database.
3. Favorite button changes to “not saved.”

Use Case 3: Export Destination Statistics

Primary Actor: Admin
Precondition: Admin is authenticated.
Main Flow:

1. Admin clicks “Export statistics.”
2. System generates file with destination popularity data.
3. File is downloaded.

Use Case 4: Navigate from Favorites to Destination Details

Primary Actor: Traveler
Preconditions: Traveler has at least one saved favorite destination.
Main Flow:

1. Traveler opens the “Favorite Destinations” page.
2. Traveler taps on a saved destination box.
3. The system loads the destination’s detailed information page.

Use Case 5: Validate Admin Access for Updates

Primary Actor: System
Preconditions: User attempts to access the “Update Destination Information” menu.
Main Flow:

1. The user selects “Update Destination” option.
2. The system checks if the user is logged in as Admin.
3. If valid, the system grants access to edit functions.
4. If not, the system shows an “Access Denied” message.

4. Non-Functional Requirements

- Database query response < 3 seconds.
- UI must be intuitive and easy to navigate.
- The system works on multiple browsers and devices without modification.
- The system complies with data privacy laws and protects PII/SPII.
- High availability and reliability for end-users.

5. Assumptions and Dependencies

- Users must have internet access.
- Admins must be authenticated to access management features.
- Users must have registered accounts to save favorites.
- Database must be available and synchronized with the backend.

Mobile and web devices must meet minimum hardware/software requirements.

6. MoSCoW Prioritization (Feature-level)

Feature / Capability	Feature-level	Notes / Rationale
Traveler search for destinations (keywords, basic filters)	Must Have	Core entry point; enables Use Case 1.
View destination details (price, culture, activities, climate)	Must Have	Directly supports Use Case 1 & 4.
Save/unsave favorites	Must Have	Enables Use Case 2 & 4; requires user accounts.
Favorites list & deep-link to details	Must Have	Use Case 4.
User accounts (register/login)	Must Have	Needed to store favorites & PII compliance.
Admin authentication/authorization	Must Have	Use Case 5; protects admin actions.
Admin update destination data	Must Have	Content management.
Activity logging (user/admin actions with timestamps)	Must Have	Traceability, audit, analytics To support NFR compliance
API + DB foundation	Must Have	Supports queries NFR.
Cross-browser/device support (responsive web)	Must Have	NFR
Performance baseline: p95 < 3s for DB-backed views	Must Have	NFR
Data privacy & PII protection	Must Have	NFR, **legal risk if missing.**
Email/password reset flows	Must Have	Basic UX expectation.
High availability baseline	Should Have	Start with ≥99%
Export destination popularity stats	Should Have	Use Case 3; simple file export.
Search refinements	Should Have	Improves discovery, still shippable without.
Admin bulk import	Could Have	Speeds data ops
Analytics dashboard	Could Have	aster insights.
Advanced search (geo, similarity, “nearby”)	Could Have	Nice-to-have discovery boost.

Mobile-native app	Won’t for Now	Start with responsive web
Real-time multi-region HA	Won’t for Now	Post–product

System Modules

1. Discovery & Search
   Keyword search
   Basic filters
   Result list & sorting

2. Destination Content
   Destination profile view
   Media gallery
   Content versioning

3. Favorites & User Profile
   Save/unsave favorites
   Favorites list
   Account settings

4. Identity & Access
   Traveler accounts (register/login)
   Admin roles/permissions
   Password reset
   Session management & token refresh

5. Admin CMS
   CRUD destinations
   Bulk import/export
   Validation & preview before publish
   Auditing

6. Analytics & Reporting
   Popularity counters (views, saves)
   CSV export of stats
   Admin dashboard

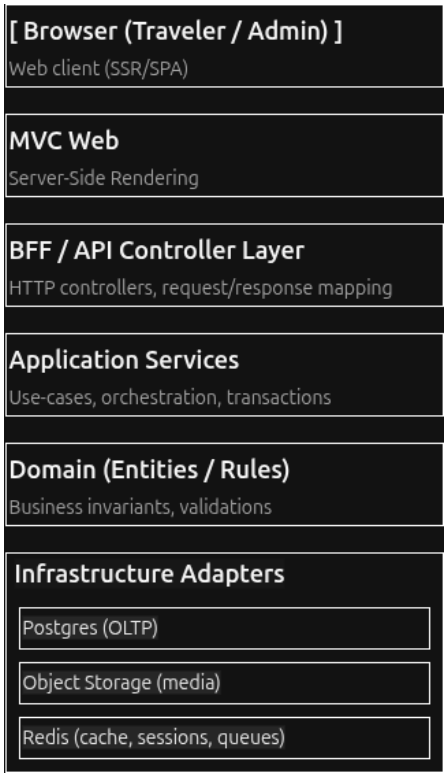
7. Observability & Activity Logs
   Structured action logs (user/admin) with timestamps
   Security/audit logs
   Usage metrics
   Alerting

8. Platform & Data Layer
   API gateway / backend services
   Destination & user DB schema, indexes
   Caching
   Backups & recovery runbooks
   Privacy & compliance controls

9. UX & Cross-Platform
   Responsive layout
   Accessibility baseline
   Cross-browser support matrix & tests

7. System Architecture

We choose MVC architecture since it is more simple and serves as the main organizing principle.



Map modules/features to components.

System Module	**Components**
Discovery & Search	SearchController, SearchService
Destination Content	DestinationController, DestinationService, MediaService, VersioningService
Favorites & User Profile	FavoritesController, ProfileController, FavoritesService
Identity & Access	AuthController, AuthService, RBACService, SessionService
Admin CMS	AdminDestinationsController, AdminImportExportService, ValidationService
Analytics & Reporting	CountersService, ExportService, Scheduler/Worker
Observability & Activity Logs	AuditLogService, UsageMetrics
Platform & Data Layer	API Gateway/Edge config, CachingPolicy, BackupRunbook

Justification

1. Ship fast
    - Goal/Priority: Deliver all Must-Haves quickly
    - Why this design: One deployable app with strict internal module boundaries result in minimal infra + fastest iteration. Clear seams (Search, Content, Favorites, Auth, Admin, Analytics) mean we can later extract/replace services if/when needed.
2. Hit NFRs early
    - Performance: SSR for fast TTFB, BFF tailors payloads to each view, Redis for read caches & queues.
    - Privacy/PII: Centralized data plane simplifies field-level encryption, key rotation, and DB least-privilege. enforce access + ActivityLog for audit.
3. Lowest operational complexity at MVP scale
    - Goal/Priority: Keep ops simple while features grow.
    - Why this design: Fewer moving parts than microservices
      no service mesh, fewer deployments, simpler tracing
4. Clean separation of concerns
    - Goal/Priority: Evolve product (Should/Could items) without regressions.
    - Why this design:
        - MVC Web = presentation only.
        - BFF/API = request/response mapping.
        - App Services = use-case orchestration (search, save, export).
        - Domain = rules/entities.
        - Infra Adapters = Postgres/Redis/Object Storage.
    - UI can be changed, storage can be replaced, or add queues without touching business rules.
5. Optimized for target users
    - Travelers: Fast search/results and details rendered via SSR, lean payloads via BFF.
    - Admins: Versioned content edits with audit logs, safe previews, exports run async so UI stays responsive.
6. Direct traceability, audit, and compliance
    - Goal/Priority: “Must Have” Activity logging + data governance.
    - Why this design: All write paths funnel through App Services to AuditLog, making user/admin actions provable and reportable.

How this maps to MoSCoW

- Must-Have features inside the monolith:
    - Search, Destination Content, Favorites/Profile, Identity & Access (RBAC), Admin CMS, Observability/Activity Logs, Platform/Data (Postgres, Redis, S3-style storage), UX/Responsive Web.
- Should-Have (search refinements, Analytics/Export) slot into SearchService without cross-cutting changes.

- Could-Have (bulk import, dashboard, advanced geo/similarity search) add adapters or a separate worker without touching original core flows.
- Won't for Now (mobile app) are isolated by BFF, adding them later doesn't disturb everything below.

<table>
<tr>
<td>Entities (Nouns)
User
Traveler
Admin
Destination
DestinationVersion
Favorite
MediaAsset
ActivityLog
PopularityCounter
ExportJob
Role
Permission</td>
<td>Operations (Verbs)
register/login/reset password
search destinations
view destination
save/unsave favorite
create/update/publish destination
upload/remove media
record activity
increment view/save counters
export popularity CSV
assign role / authorize</td>
</tr>
<tr>
<td colspan="2">Traceability to Requirements
User/Traveler/Admin → Accounts, Auth, Admin authorization.
Destination/DestinationVersion/MediaAsset → View details, Admin CRUD & versioning.
Favorite → Save/unsave + favorites list.
PopularityCounter → Popularity stats & export.
ActivityLog → Activity logging for audit/analytics.
ExportJob → CSV export workflow.</td>
</tr>
</table>