Database Systems Lab
Functional and Technical Requirements

Rauch, Christian
Bouros, Panagiotis

(changed: August 19, 2025)

Contents

1 Overview

1.1 Purpose

This document outlines the requirements with functional and technical specifications necessary to meet the project goals.

1.2 Project Description

You are tasked to develop an event booking platform. The platform will support browsing, searching, booking, and organizing of events, with account management features for both attendees and event organizers.

1.3 Documentation

Document the implementation, lean and simple! Provide a PDF document for functional requirements: FR-IMPL.pdf. Have a subchapter for each requirement, list identifier and description.
For the technical documentation, you are required to submit your source code and other components needed to set up and run your system. This includes sample data.

2 Functional Requirements

2.1 Visitors

As *visitor* (i.e., person without account), I can:

[FR-Vi-01] browse events in all categories,
[FR-Vi-02] search for events using keywords and filters (e.g., date range, location, price),
[FR-Vi-03] create a new account in the system.

2.2 Users

As a *user* (i.e., person with account), I can additionally:

[FR-Us-01] log in using my username and password.
[FR-Us-02] book tickets for events,
[FR-Us-03] leave reviews for attended events,
[FR-Us-04] manage my account details.
[FR-Us-05] save events to a watchlist,
[FR-Us-06] track the status of my bookings,
[FR-Us-07] communicate with organizers via a messaging system,
[FR-Us-08] subscribe to event organizers to receive notifications for new events,
[FR-Us-09] see upcoming events from subscribed organizers in an aggregated overview,
[FR-Us-10] apply to become an event organizer.

2.3 Event Organizers

As an *event organizer*, I can additionally:

[FR-Eo-01] create and manage event listings (title, description, images, ticket prices, date, location, ...),
[FR-Eo-02] track and change the status of bookings,
[FR-Eo-03] respond to attendee inquiries via direct messages,
[FR-Eo-04] create hierarchical event categories for my organization,
[FR-Eo-05] access attendance analytics and reports (with charts).

3 Technical Requirements

3.1 Environment

[**TR-En-01**]	Version control.	Set up a version control system (i.e., Git) to manage changes to the codebase.
[**TR-En-02**]	Installation.	Install MariaDB, Python, Flask, venv, matplotlib, ...
[**TR-En-03**]	Environment.	Have a dedicated Python environment created with virtualenv. Have a shared file with project dependencies (i.e., installed libraries, pip install -r dependencies).

3.2 Data model

[**TR-Dm-01**]	Data model.	Provide a data model for your entities and their relationships. You should have at least 8 entities.
[**TR-Dm-02**]	Relationships.	Demonstrate the different relationships (n-to-m, n-to-1, ...) between entities. Label at least one edge (relationship) with roles.
[**TR-Dm-03**]	Inheritance/specialization.	You should have at least two type hierachies.
[**TR-Dm-04**]	Normalization.	Ensure that your database design follows normalization rules to eliminate redundancy and dependency anomalies. Demonstrate 3NF and BCN.

3.3 Database

[**TR-Db-01**]	Creation script.	Provide a SQL script (Data Definition Language) to create the data base on a plain system.
[**TR-Db-02**]	Populate script.	Provide a SQL script to 1) insert essential and 2) sample data into the system.
[**TR-Db-03**]	Queries.	Use: DISTINCT, LIKE, IN, GROUP BY, HAVING, and ORDER BY.
[**TR-Db-04**]	Stored procedure.	Call at least two stored procedures.
[**TR-Db-05**]	Event trigger.	Have at least two event triggers.
[**TR-Db-06**]	Assertion.	Have at least one assertion.
[**TR-Db-07**]	View.	Have at least two views.
[**TR-Db-08**]	Subqueries.	Have at least two subqueries: Correlated and non-correlated. Use: EXISTS / NOT EXISTS, ANY, and ALL.
[**TR-Db-09**]	Sets.	Use: UNION, INTERSECT, and EXCEPT. Different JOINs. DISTINCT, LIKE, etc.
[**TR-Db-10**]	Joins.	Use: OUTER JOIN and INNER JOIN.
[**TR-Db-11**]	Create, update and delete.	Use: INSERT, UPDATE, and DELETE.
[**TR-Db-12**]	Binary data.	Binary data is stored and accessed. E.g., users can add files to their comments.
[**TR-Db-13**]	Constraints.	Define constraints such as primary keys, foreign keys, unique constraints, and check constraints to ensure data integrity. E.g., Ensure email addresses are unique, product prices are non-negative.
[**TR-Db-14**]	Indices.	Define at least three indexes to optimize database query performance. E.g., Indices on frequently queried fields like user email or product name.
[**TR-Db-15**]	Transactions.	Have at least two transactions that are rolled back in an error case.

3.4 Common

[**TR-Co-01**]	Type annotations.	Use type annotations for method signatures. E.g., def example1(param1: str, param2: int) → pd.DataFrame: ...
[**TR-Co-02**]	Object orientation.	Use classes and inheritance (i.e., metaclass).
[**TR-Co-03**]	Design Principles.	Components need to be modular and decoupled; adhere to the SOLID principles at least for some components.
[**TR-Co-04**]	Error handling.	Implement comprehensive error handling mechanisms, both at the backend level (e.g., transaction rollbacks) and frontend level (e.g., user-friendly error messages).

3.5 Backend

[**TR-Ba-01**]	CRUD API.	Provide a data access API to create, read, update, and delete entities.
[**TR-Ba-02**]	Caching.	Have at least one component that uses a hash table (i.e., dict) as a caching structure to safe DB accesses.
[**TR-Ba-03**]	Unit tests.	Core components need to be tested with unit tests. There are usually different scenarios to test for a single method. Follow the naming pattern [action]_[scenario]_[behavior]. E.g., createNewUser_fieldsIncomplete_raiseException.
[**TR-Ba-04**]	Logging.	Append system logs to a file with the pattern YYYY-MM-DD.log. You have three severities (info, warning, error) and two categories (system, user).
[**TR-Ba-05**]	Auditing.	Implement a simple audit function (for at least one entity) to track changes made to data, including who made the change and when. Write these logs to a table. E.g., Using a trigger, track the order state history in a separate table.
[**TR-Ba-06**]	Parallelization and asynchronism.	Parallelize computation-heavy aspects of algorithms (multi-processing or multi-threading). Delegate long-running computations to background workers. E.g., Generate charts for selling statistics using a thread-pool.
[**TR-Ba-07**]	Configuration.	Maintain a central configuration for storing credentials and settings.

3.6 Frontend

[**TR-Fr-01**]	RESTful API.	Use RESTful communication between frontend and backend.
[**TR-Fr-02**]	Scripts.	Have a shared script file common.js with generic functions (e.g., disable_element(id), load_image(id,size)). Have one or many more specific script files with domain specific functions.
[**TR-Fr-03**]	Styling.	Have at least one CSS file. Use simple styles. Target desktop devices.
[**TR-Fr-04**]	Interactivity.	Have some generic functions to load data and update the UI. E.g., react to a click event, disable buttons, load data from the backend, update the UI with the results, enable the buttons.

4 References

- Database Systems lecture
- https://mariadb.org
- https://sqlite.org
- https://python.org
- https://flask.palletsprojects.com
- https://docs.python.org/3/library/sqlite3.html
- https://w3schools.com/sql
- https://peps.python.org/pep-0249
- https://docs.python.org/3/library/venv.html
- https://pythontutorial.net/python-oop
- https://pythontutorial.net/tkinter
- https://matplotlib.org