CS673 Software Engineering
Team 4 - beatmap
Project Proposal and Planning

Your project Logo here if any

Team Member	Role(s)	Signature	Date
Samuel Akerman	Team Leader Configuration Leader	*Samuel Akerman*	9/9/2025
Michael Laszlo	Security Leader	*Michael Laszlo*	9/9/2025
Gregory Grimaylo	QA Leader	*Gregory Grimaylo*	9/10/2025
Zhehao Liao	Requirement Leader	*Zhehao Liao*	9/9/2025
Edoardo Cavallero	Design and Implementation Leader	*Edoardo Cavallero*	9/9/2025
Zixuan Cheng	Design and Implementation Leader	*Zixuan Cheng*	9/10/2025

Revision history

Version	**Author**	**Date**	**Change**
1.0.0	Edoardo Cavallero	9/9/25	Added risk management section, management plan timeline
1.0.0	Samuel Akerman	9/9/25	Added Overview, Related Work, Configuration Management sections

1.0.0	Michael Laszlo	9/9/25	Added Security Requirements and some Essential Security Features
1.0.0	Greg Grimaylo	9/8/25	Added Quality Assurance Plan
1.0.0	Zixuan Cheng	9/9/25	Added Management Plan(Objectives and Priorities)
1.0.0	Zhehao Liao	9/9/25	Added Functional Requirements with estimated time

1. Overview

Our project, beatmap, is a web application designed to help travelers and music fans discover live events in the cities they plan to visit. At its core, the app allows a user to enter a destination and travel dates, then returns a curated list of concerts and music events happening in that timeframe.

To make the results more personalized, users can refine their search by selecting preferred genres or artists. For an even richer experience, the app can connect to a user's streaming service (e.g., Spotify or YouTube Music) to access listening history and recommend shows that match their tastes.

We plan to implement and develop this project using Python. At this stage, we haven't yet made a determination of which libraries and framework will be employed. However, in addition to Python as the core language, we'll be using GitHub for SCM, GitHub Actions for

CI/CD, Docker for containerization. We will likely use Jira for issues tracker and Google Cloud Run for deployment.

2. Related Work

There are a few products to attempt to solve a need similar to BeatMap, those are:

Bandsintown: is a website where you can add your location and a list of your favorite artists. They will notify you when one of the artists in the list schedules a concert in the selected location.

Songkick: similarly, it notifies you when an artist of interest announces a new concert. It appears to only integrate with Spotify, however.

Spotify: Spotify's own "concerts near you" feature. Using the readily accessible listening history of a user, they use it in order to notify when new concerts are announced.

Just like our proposed project, these systems generate a list of concerts based on user input. Our key differentiator is that our project can leverage prior user behavior, but does not rely on it exclusively. We believe this increases discoverability, as our app focuses on providing event recommendations based solely on location and dates—giving users the chance to discover and enjoy performances they might not have otherwise known about.

3. Proposed High level Requirements

a. Functional Requirements
    i. Essential Features
        - Concert Search
          As a visitor, I want to search by artist, city, or venue, so that I can quickly find relevant events.
          Estimate: 14–22 person-hours
        - Filters & Sorting
          As a visitor, I want to filter by date range, location radius, genre, and price, and sort by date/popularity, so that I can narrow results efficiently.
          Estimate: 12–20 person-hours
        - Event Details Page
          As a visitor, I want to view event time, venue, lineup, pricing info, and an official ticket link, so that I can decide whether to attend.
          Estimate: 10–16 person-hours
        - Ticket Link-Out (External Purchase)
          As a visitor, I want to open the official ticketing site in a new tab, so

that I can purchase safely on the provider's website.
Estimate: 6–8 person-hours

- User Accounts (Auth)
  As a visitor, I want to sign up and log in, so that my favorites and preferences are saved.
  Estimate: 14–24 person-hours
  Note: Prefer OAuth or passwordless "magic link" to reduce complexity.
- Favorites / Wishlist
  As a registered user, I want to save events to a personal list, so that I can revisit them later.
  Estimate: 10–16 person-hours
  Note: Use localStorage fallback for logged-out users.
- External Events API Integration
  As an admin, I want to ingest and refresh events from a third-party provider (e.g., Ticketmaster/Songkick), so that the catalog stays current.
  Estimate: 16–26 person-hours
  Note: Include rate limiting, retries, and scheduled refresh.

Essential subtotal: ~82–132 person-hours (fits ~3–4 weeks for 1–2 web devs)

ii. Desirable Features:
- Map View (Web)
  As a visitor, I want to see events on an interactive map and search within a radius, so that I can plan by location.
  Estimate: 12–18 person-hours
- Artist & Venue Profiles
  As a visitor, I want dedicated artist/venue pages with upcoming shows, so that I can explore related events easily.
  Estimate: 8–12 person-hours
- Calendar Export (.ics)
  As a registered user, I want to export saved events to my calendar, so that I can manage my schedule.
  Estimate: 8–10 person-hours
- Email Alerts (Basic)
  As a registered user, I want to receive an email when new nearby shows from my tracked artists are listed, so that I don't miss them.
  Estimate: 10–14 person-hours

iii. Optional Features:
- Personalized recommendations (based on favorites/history)
- Price tracking with email alerts
- Social sharing (Open Graph/Twitter cards)

- Multi-language support (i18n)
- PWA + web push notifications + offline caching

b. Nonfunctional Requirements
    i. Security Requirements
        1. Authentication & Authorization
            a. All authentication must use secure protocols (OAuth2, OIDC, or JWT with short lifetimes)
            b. Passwords must be stored using salted, adaptive hashing algorithms (bcrypt, Argon2, or PBKDF2)
            c. Access must be controlled using role-based or attribute-based access control at the API layer
            d. Refresh token rotation and revocation mechanisms must be implemented
        2. Data Protection
            a. All data in transit must be encrypted with TLS 1.2+ (HTTPS enforced with HSTS)
            b. All sensitive data at rest must be encrypted using industry-standard algorithms (AES-256 or equivalent)
            c. No secrets (API keys, credentials, tokens) may be stored in code or version control; they must be managed via environment variables or a secret manager
            d. Logs must not include sensitive information such as passwords, tokens, or Personally Identifiable Information
        3. Input Validation & Integrity
            a. All API inputs must be validated using Pydantic models with explicit field constraints
            b. The system must reject malformed or invalid payloads
        4. System Hardening
            a. OpenAPI documentation must be disabled or access-controlled in production
        5. Resilience & Abuse Protection
            a. Authentication and other sensitive endpoints must have rate limiting to prevent brute force attacks
            b. The system must detect and log repeated failed login attempts and trigger account lockouts or alerts
            c. APIs must implement throttling and request size limits to mitigate DoS risks
            d. Error messages must be generic and must not disclose internal system details
        6. Monitoring & Auditing
            a. All security-relevant events (login, logout, failed attempts, role changes, data exports) must be logged

   b. Audit logs must be immutable, time-synchronized, and retained for a defined period of time
   c. Anomalous activity must generate alerts for the security team
7. Deployment & Operations
   a. CI/CD pipeline must include automated security tests (linting, dependency checks, secret scanning)

4. Management Plan

a. Objectives and Priorities

Primary Objective

- Build a reliable, user-friendly concert discovery platform that allows visitors to search, explore, and save live music events.

Highest Priority

- Implement concert search (by city, venue, date period).
- Provide secure user sign-up and login (OAuth or passwordless “magic link”).
- Enable favorites/wishlist (persisted for logged-in users; localStorage fallback for visitors).
- Ensure a stable and deployable system with no critical bugs.

Stability and Quality Goals

- Apply unit testing, integration testing, and peer code reviews.
- Design a responsive, user-friendly interface accessible across devices.
- Prioritize security and privacy with secure authentication, safe data handling, and proper API key management.

Future Iterations

- Add advanced filters and sorting (date range, genre, price, popularity).
- Develop event detail pages with official ticket link-outs.
- Introduce caching and refresh strategies to keep event data up to date.

Desirable Features

- Cross-device wishlist synced via accounts.
- Interactive map view (with location radius and permissions handling).
- Email alerts for new shows from tracked artists or locations.

Optional Features

- Personalized recommendations (based on favorites/history).
- Price tracking with alerts.
- Social sharing cards (Open Graph/Twitter).
- Multi-language support (i18n).
- PWA enhancements (offline caching, push notifications).

Our project aims to deliver a reliable, user-friendly concert discovery platform with essential features such as search, secure accounts, and favorites completed first. Beyond these core goals, we will expand iteratively to include advanced filters, event detail pages with ticket link-outs, map-based search, and email alerts. Optional features like personalized recommendations, price tracking, social sharing, and multi-language support (i18n) will be considered if time permits. By combining strong engineering practices with tools like GitHub, GitHub Actions, Docker, and Jira, the team will ensure the system is secure, maintainable, and high quality.

b. Risk Management (need to be updated constantly)

Personnel

- **Loss of team members** (Priority 40): Low likelihood but high impact and cost if one or more members drop the class. This would require redefining project scope and redistributing the departing member's tasks among the remaining team.

- **Addition of new members** (Priority 45): Low likelihood, moderate impact. One new member has already been added, and the team adapted by splitting the demanding design and implementation leader role into two shared positions to balance workload. Future additions would follow a similar redistribution plan.

- **Lack of motivation or responsibility** (Priority 25 – high priority): Very high potential impact if motivation falters. Currently assessed as unlikely, since initial engagement was strong. If it arises, the plan is to address concerns openly and provide support so that work can be completed collaboratively.

Communication

- **Ineffective communication / unclear individual goals** (Priority 60): Currently assessed as very unlikely, given clear communication at the start. Impact would be moderate if it did occur, potentially slowing progress. Mitigation includes raising issues during meetings and clarifying roles and expectations.

Requirements

- **Unclear requirements** (Priority 36): Some risk of ambiguity in what is expected. If encountered, the team will meet with the requirement leader to clarify needs and offer support to ensure alignment.

- **Scope creep** (Priority 36): Slightly more likely than unclear requirements but still low risk, given the well-defined project scope. The team is aligned on expectations, but meetings will be used to prevent scope expansion if it arises.

Management

- **Lack of management skills** (Priority 15 – highest risk): Most team members see management as an area for improvement. A failure in this area would have severe consequences for project quality, structure, and timely completion. The team plans to share management duties, holding each other accountable and building skills as the project progresses.

Technology Competence

- **Unfamiliarity with chosen framework** (Priority 36): Unlikely since the team chose a familiar language, but gaps could still emerge. If so, group discussions and peer support will be used to overcome them.

- **Other technical challenges** (Priority 36): General technology issues are possible but manageable through the same support and collaborative learning approach.

Design & Implementation

- **Improper design** (Priority 20): Moderate likelihood but very high impact if design flaws undermine the project. Five of six members have software backgrounds, and the design/implementation leader role is shared to provide oversight, review, and mutual support to reduce this risk.

- **Messy or inconsistent code** (Priority 32): Could reduce efficiency and collaboration. The team plans to use AI tools and coding best practices to produce clear, well-structured, and mutually understandable code.

Testing

- **Not enough testing** (Priority 32): Testing gaps could weaken the final product. Mitigation is strong, as one member has professional QA engineering experience and will guide thorough, systematic testing.

Overall, the group identified risks across seven categories: personnel, communication, requirements, management, technology competence, design and implementation, and testing. The highest-priority risks are lack of management skills,

loss of team members, and lack of motivation, since they could directly threaten project completion. Other risks, such as unclear requirements, technical challenges, or code quality, are considered moderate but manageable with clear communication and teamwork. Strong initial engagement, shared leadership roles, and the presence of experienced members in design and QA provide confidence that the team can effectively mitigate risks and maintain steady progress throughout the semester.

Risk Management Sheet Link:
https://docs.google.com/spreadsheets/d/15xjpjp-jU5_uRAdZ2A_Y7XQ5df6s4UnU-VE0Q9hf0Ao/edit?usp=sharing

c. Timeline (this section should be filled in iteration 0 and updated at the end of each later iteration)

Iteration	Functional Requirements(Essential/ Desirable/Optional)	Tasks (Cross requirements tasks)	Estimated/real total person hours
1	**Functional Requirements (Essential)** **Concert Search** (15-25 hrs) **Filters and sorting** (10-20 hrs) **Event Details Page** (10-15 hrs) **Ticket Link-Out** (6-10 hrs)	**Basic UI/UX setup** (design system, navigation, layout) (8-12 hrs) **Database schema and API layer setup** (8-12 hrs) **External events API integration** (8-10 hrs) **CI/CD pipeline, linting, dependency/security checks** (6-8 hrs)	**Estimated total:** 71-112 hrs
2	**Functional Requirements (Essential)** **User Accounts** (Auth, Oauth) 14-24 hrs **Favorites/Wishlist** (with localStorage fallback) 10-16 hrs **External Events API integration** (16-26 hrs) **Desirable Features**	**Security implementation** (OAuth2/OIDC, token rotation, passwordless) (12-16 hrs) **Input validation** (Pydantic models) (6-8 hrs) **Error handling patterns and logging framework**	**Estimated total:** 78-118 hrs

	Artist and Venue Profiles (8-12 hrs)	(6-8 hrs) **Role-based access control foundation** (6-8 hrs)	
3	**<u>Desirable features</u>** **Map View** (interactive search by radius) (12-18 hrs) **Calendar Export** (.ics) 8-10 hrs) **Email Alerts** (10-14 hrs) **<u>Optional Features</u>** **Social sharing** (OG/Twitter cards) (6-8 hrs) **Basic multi-language support** (i18n) (8-10 hrs)	**Security Hardening** (rate limiting, lockouts, HSTS, request throttling) (10-14 hrs) **Monitoring and auditing** (security event logging, anomaly alerts) (10-14 hrs) **Deployment operations** (prod config, disabling OpenAPI docs secret mgmt) (6-8 hrs) **QA testing**, **polish, bug fixes** (8-12 hrs)	**Estimated total:** 70-94

5. Configuration Management Plan

a. Tools

- SCM: Git/GitHub
- CI/CD: Github Actions.
- Containers: Docker
- Deployment/Cloud hosting: Google Cloud Run
- IDE: VS Code
- Issue tracker: Jira
- AI for research and dev: ChatGPT, Gemini, Copilot.
- AI Agent: we are currently exploring different options, including the LangChain which uses OpenAI GPT models.
- Frontend: not defined yet. The backend will be exposed via REST API using FastAPI. The frontend tools are being determined.
- SAST: Github CodeQL

b. Code Commit Guideline and Git Branching Strategy

We will use a trunk based branching strategy. We will have a branch for each feature

and will encourage frequent, small commits. We also seek to continuously merge feature branches into main, with the premise that the main branch should always build successfully and be able to be deployed.
When a developer considers their work ready to be merged into main, they will issue a pull request that needs to be reviewed and approved by at least another developer (ideally 2 or more)
The preferred method for integrating commits from one branch to another will be merging as opposed to rebasing, since it is a simpler approach and poses few, if no risks.

c. CI/CD Plan

For all branches, as soon as a new push with a bundle of commits is detected, we'll use GitHub Actions to run linting, dependency checks, secret scanning, security checks and unit tests.
Upon a successful build of the main branch, we'll automatically deploy our application using a docker image to Google Cloud Run.

6. Quality Assurance Plan

a. Metrics

The result of these metrics should be reported in the progress report/ iteration summary sheet.)

Metric Name	Description
Test coverage	% of code that is covered by automated tests
Test pass rate	Number of test cases passed vs. executed
Response time	How fast the app responds to API requests
Error rate	% of failed API requests
Cost	Number of person hours used

b. Coding Standard

Since we will be using Python, we should follow PEP8, the official Python style guide, for:

- Indentation
- Naming conventions (variable_function_names, ClassNames)

- Documentation (use docstrings)

We should also organize code into different folders/modules, for example a folder for all the API related code, one for all the tests, etc.

c. Code Review Process

All team members will be responsible for reviewing each other's code. Pull requests will be created when a team member wants to merge their code to main, and at least 1, but preferably 2 or more team members will review the pull request. We can use a pull request review checklist to help review pull requests effectively, this one as an example: Pull Request Review Checklist
Reviewers should:

- Prioritize blocking issues that require a fix vs. nice to haves
- Explain why something should be changed instead of just mentioning a change
- Acknowledge & praise

d. Testing

We will use frameworks that test our Python backend, like *pytest*. We can also use *pytest-mock* for mocking dependencies our app would need, like API calls.
For integration testing, we will use FastAPI to ensure APIs are working properly; FastAPI conveniently enables Swagger UI, through which we'll be able to easily test API calls and analyze their results.
Each developer should unit test their code as they are building it, however integration testing can be done by the QA Leader since they are focused on QA, or by for example the implementation leader since they will have a better understanding of the architecture of the application.
Integration testing can be done on every push/pull request using GitHub Actions. Unit tests will also be done at this time.
Manual testing should be done to cover edge cases, after major changes/merges, and before release to verify that the product as a whole is working (user walkthrough).
Our tests should aim to verify:

- Functionality
- Reliability
- Security (ex. Secrets not exposed)
- Performance/Response times
- User experience
- Test coverage

e. Defect Management

Jira will be used as the defect management tool. All layers of our project should have defects, such as: Integration defects, security defects, validation defects, deployment

defects, and input validation defects. All members of the team should work on defect management:

- Identifying defects + reporting them (anyone can find defects)
- Root cause analysis
- Defect assignment can be done by the Team Leader
- Defect resolution
- Verification can be done by the QA Leader
- Documentation

7. AI usage Log

You are allowed and even encouraged to use AI tools to help you generate the project idea, plan it and build it, but you need to clearly describe 1) What tools were used? 2) for what specific tasks and 3) Is it helpful? 4) how did you evaluate or modify AI-generated content? Additionally, you should submit the exported AI chat history as an appendix or share that with the instructor and facilitators.

Tools	Who	Tasks	helpful	Evaluation/modification	links
Chat-GPT	Michael	Security Leader Non-Functional Requirements Generation	Yes.	I read through the provided output and thought about how what was generated may apply to this project and included those I thought were fairly doable and left off the ones that went into a level of complication we likely did not need or would get to with this project. As part of modification I changed my question from being about generic Python application requirements and focused more on what a security leader working with FastAPI specifically would think about which gave better results.	https://chatgpt.com/share/68c1b651-3288-800f-9ee3-99c77e46da81
Chat-GPT	Sam	Brainstorm project ideas	Yes	I explained what the purpose of the course was and asked to suggest different project ideas. I	https://chatgpt.com/share/68bf63bd-d18c-8013-a63d-fe40dc984

				encouraged it to make it a little heavier on the backend/API side as opposed to frontend heavy. Some of the suggested ideas were interesting and I took them to the team.	6dc
Chat-GPT	Greg	Quality Assurance Plan Suggestions	Yes	I judged the output from Chat-GPT based on how useful the proposed plan would be while testing our application and how feasible implementing that plan would be. For example, some metrics Chat-GPT provided I felt we could not efficiently evaluate, like defects per thousand lines of code.	https://chatgpt.com/share/68bf908b-7b74-8013-8e36-5056795d1141
Chat GPT	Ed	Format risk management section	Yes	Re-prompted to include more of the original details and refine the structure of the text and made minor edits for final draft	https://chatgpt.com/share/68c1e888-78a0-800d-85d9-7586d5383c9d

8. References

(Any references/citations that you have used)

9. Glossary

(Any acronym used in the document should be explained here)
SCM: Source Control Management
CI/CD: Continuous Integration/Continuous Deployment
IDE: Integrated Development Environment
SAST: Static Application Security Testing
API: Access Point Interface