The recommended path is a phased, monolith‑first development plan that starts with project scaffolding and schema, then implements a concurrency‑safe booking flow, adds API docs and caching, and finally deploys, load‑tests, and ships documentation and diagrams. This sequence ensures correctness under race conditions first, then performance and operability, and it maps cleanly to the Evently deliverables.[spring+1](https://docs.spring.io/spring-boot/how-to/deployment/index.html)

**Plan and setup**

* Choose a modular monolith in Spring Boot with clear domain modules (users, events, bookings, analytics) to minimize operational overhead while preserving future service boundaries.[martinfowler+1](https://martinfowler.com/bliki/MonolithFirst.html)
* Confirm the stack: Spring Web, Data JPA, Validation, Security, Actuator; PostgreSQL/MySQL; Flyway for migrations; springdoc‑openapi for API docs; optional Redis for read caching; Docker for packaging; k6 for load tests.[baeldung+1](https://www.baeldung.com/database-migrations-with-flyway)

**Initialize project**

* Generate the project via Spring Initializr with Maven/Gradle, Java 17/21, and the required starters to bootstrap a production‑ready skeleton quickly.[start.spring+1](https://start.spring.io/index.html)
* Verify the project runs locally and the base endpoints are reachable to de‑risk environment issues early.[spring+1](https://spring.io/quickstart)

**Schema and migrations**

* Design the core tables: users, events, bookings, with foreign keys, NOT NULLs, and indexes on event\_id, user\_id, and event time to support hot queries.[geeksforgeeks+1](https://www.geeksforgeeks.org/springboot/spring-boot-flyway-database/)
* Manage schema with Flyway migrations (V1…Vn) so every environment is reproducible and auto‑migrates on startup.[spring+1](https://docs.spring.io/spring-boot/api/rest/actuator/flyway.html)

**Domain and services**

* Implement a layered structure (controller → service → repository) with transactional service methods to keep business logic and persistence concerns clean.[baeldung+1](https://www.baeldung.com/database-migrations-with-flyway)
* Keep modules cohesive: Users (auth/roles), Events (CRUD/list), Bookings (create/cancel/list), Analytics (aggregates), reflecting clear domain seams for future evolution.[baeldung+1](https://www.baeldung.com/spring-modulith)

**Concurrency and correctness**

* Prevent overselling with optimistic locking using a versioned aggregate (Event or EventInventory) so concurrent updates fail fast with a safe retry strategy.[baeldung+1](https://www.baeldung.com/jpa-optimistic-locking)
* Wrap booking create/cancel in a single transaction: re‑read inventory, validate capacity, persist changes, and commit or retry on version conflict.[mimacom+1](https://blog.mimacom.com/testing-optimistic-locking-handling-spring-boot-jpa/)

**Idempotency and duplicates**

* Make create‑booking idempotent with an Idempotency‑Key so duplicate POSTs return the same result, handling client retries and network timeouts safely.[geeksforgeeks+1](https://www.geeksforgeeks.org/javascript/what-is-an-idempotent-rest-api/)
* Reinforce with a unique constraint for active bookings per user‑event to eliminate duplicates under concurrent submission.[restfulapi+1](https://restfulapi.net/idempotent-rest-apis/)

**API design and docs**

* Expose REST endpoints for event browse/detail, booking create/cancel/list, and admin event CRUD plus analytics, with consistent pagination and status codes.[howtodoinjava+1](https://howtodoinjava.com/spring-boot/springdoc-openapi-rest-documentation/)
* Publish interactive API docs using springdoc‑openapi/Swagger UI so reviewers can test endpoints in the browser.[github+1](https://github.com/springdoc/springdoc-openapi)

**Caching and performance**

* Cache read‑heavy endpoints (e.g., upcoming events, event details) with Spring Cache and short TTLs; invalidate caches on event updates to avoid stale data.[spring+1](https://docs.spring.io/spring-boot/reference/io/caching.html)
* Never cache write paths or availability checks; rely on DB transactions and locking as the single source of truth.[baeldung+1](https://www.baeldung.com/jpa-optimistic-locking)

**Security and access**

* Add Spring Security with roles USER/ADMIN, guarding admin event and analytics endpoints while keeping public browse endpoints accessible as designed.[howtodoinjava+1](https://howtodoinjava.com/spring-boot/springdoc-openapi-rest-documentation/)
* Validate inputs systematically and standardize error semantics (400/422 for validation, 401/403 for auth, 404 for missing, 409 on capacity conflicts) for predictability.[baeldung+1](https://www.baeldung.com/spring-rest-openapi-documentation)

**Testing strategy**

* Unit test services and validators, and repository tests for queries and constraints to catch logic and schema regressions early.[baeldung+1](https://www.baeldung.com/spring-rest-openapi-documentation)
* Integration tests for booking contention and optimistic lock handling to prove no overselling occurs under concurrent requests.[mimacom+1](https://blog.mimacom.com/testing-optimistic-locking-handling-spring-boot-jpa/)

**Load testing**

* Use k6 to simulate spikes and last‑seat races, capturing throughput, latency, and error rates while verifying that booking counts never exceed capacity.[grafana+1](https://grafana.com/docs/k6/latest/set-up/install-k6/)
* Include at least one scripted scenario that retries on transient failures to demonstrate idempotency working correctly.[k6+1](https://k6.io/)

**Deployment and packaging**

* Containerize with Docker for a consistent runtime image, then deploy the backend alone to a managed platform; expose a stable base URL and health endpoints.[spring+1](https://spring.io/guides/gs/spring-boot-docker/)
* Externalize configuration via environment variables or profiles to keep secrets and DB URLs out of code and to support multiple environments cleanly.[baeldung+1](https://www.baeldung.com/dockerizing-spring-boot-application)

**Observability and health**

* Enable Actuator health/readiness so platforms can probe liveness, and include minimal logging for request traces during demo and diagnosis.[dev+1](https://dev.to/nikhilxd/deploying-a-spring-boot-application-a-comprehensive-guide-3iai)
* Confirm that Swagger UI and health endpoints are reachable in the deployed environment for evaluators.[springdoc+1](https://springdoc.org/)

**Data seeding**

* Seed a few events and users via Flyway or a controlled initializer so the live demo has realistic data without manual setup.[geeksforgeeks+1](https://www.geeksforgeeks.org/springboot/spring-boot-flyway-database/)
* Keep seeds idempotent or versioned within migrations to avoid duplicates across deploys.[spring+1](https://docs.spring.io/spring-boot/api/rest/actuator/flyway.html)

**Diagrams and documentation**

* Produce a high‑level architecture diagram showing API, DB, cache, and locking/idempotency controls, and an ER diagram for users/events/bookings.[baeldung+1](https://www.baeldung.com/spring-modulith)
* Write a short README covering design trade‑offs (locking vs retries, caching scope), scalability choices (indexes, TTLs), and link to Swagger UI and load‑test notes.[springdoc+1](https://springdoc.org/)

**Final verification and submission**

* Run end‑to‑end flows on the deployed URL via Swagger UI to record the demo video, highlighting the contention test and analytics endpoints.[spring+1](https://docs.spring.io/spring-boot/how-to/deployment/index.html)
* Share the live base URL, GitHub repo with migrations and k6 scripts, diagrams, and the 5–7 minute walkthrough to fulfill Evently’s deliverables

**In progress / next steps**

* Security: add role‑based access (USER/ADMIN) and JWT/session setup; wire a UserDetailsService that maps the domain User to UserDetails for auth.[marcobehler+1](https://www.marcobehler.com/guides/spring-security)
* API docs: add springdoc‑openapi and expose Swagger UI on the deployed URL for evaluators.[springdoc](https://springdoc.org/)
* Caching: apply Spring Cache with short TTLs for event reads (never cache availability checks).[spring](https://docs.spring.io/spring-boot/reference/io/caching.html)
* Idempotency: require Idempotency‑Key on create‑booking and handle safe retries.[restfulapi](https://restfulapi.net/idempotent-rest-apis/)
* Analytics: implement aggregate endpoints (totals, utilization, popular events) with efficient queries.[spring](https://docs.spring.io/spring-boot/reference/using/structuring-your-code.html)
* Load testing: script last‑seat contention and spike scenarios with k6; confirm no overselling under concurrency.[k6](https://k6.io/)
* Deployment: containerize and deploy the backend service; expose health endpoints and the live base URL