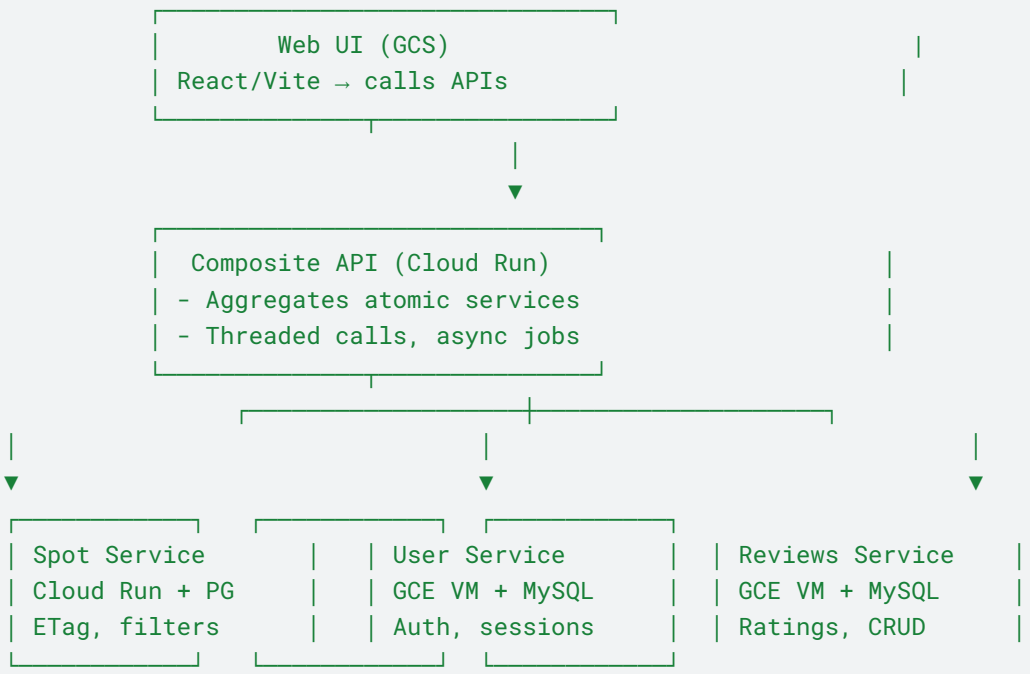
SPRINT 2 — PROJECT PLAN SKETCH

Overall Goal

Expand our NYC Study Spots cloud project into a full multi-microservice cloud system meeting all Sprint 2 expectations:

- 3 atomic microservices deployed across **Cloud Run** and **Compute Engine**

None



- 1 **composite microservice** handling cross-service aggregation, parallel execution, and logical FK validation
- 3 databases (**2 Cloud SQL instances** + **1 MySQL VM**)
- A **browser web UI** hosted in **Cloud Storage**
- Full API correctness demonstrated in Spot Management Service (ETag, query params, pagination, linked data with relative paths, 201 Created, and 202 Accepted with async polling)
- Updated GitHub Projects/Trello and live demo of the system

System Architecture

Component	Hosting	Database	Description
Spot Management Service	Cloud Run	Cloud SQL (Postgres)	Manages study spot details, amenities, hours; provides filters, pagination, ETag
User Management Service	GCE VM	Cloud SQL (MySQL)	Handles user accounts, authentication, preferences; ETag, pagination, sessions
Reviews & Ratings Service	GCE VM	VM (MySQL)	Stores reviews/ratings, links to users + spots, avg rating query
Composite Gateway Service	Cloud Run	(Encapsulates and delegates to atomic APIs (façade + aggregation))	Aggregates calls to other services, enforces FK checks, threaded calls, 202 async jobs
Web UI	Cloud Storage (SPA)	N/A	React + Vite frontend for users to browse, review, rate spots
CI/CD + Infra	GitHub Actions + Terraform (optional)	—	Builds, tests, deploys, and maintains the pipeline and board

run Spot Management on Cloud Run since it's lightweight, stateless, and scales easily.

The Spot Management service is admin-controlled: it only manages public study-spot information — names, addresses, amenities, and hours.

It's stateless — every request is independent (no login sessions or user-specific state).

It mostly handles read-heavy traffic (many users fetching data, few writing).

That makes it ideal for Cloud Run, where Google automatically scales containers and turns them off when idle.

Cloud Run is secure and low-maintenance: you don't manage the OS or networking manually; Google handles that for you.

BUT User Management runs on GCE because it needs more control over runtime, security, and session behavior.

Same for the others too

Breakdown — 7 Work Chunks

---

1. Cloud Database and Networking Setup

Purpose: Create and configure all databases, networking, and access control.
Responsibilities:

- Provision 2 Cloud SQL instances: Postgres (for spots) + MySQL (for users)
- Provision 1 stand-alone MySQL VM (for reviews)
- Configure private IP, VPC, connectors for Cloud Run + VMs
- Seed tables and test connectivity from each service
  Deliverables:
- SQL schemas, seed scripts (`mysql-db` repo)
- Connection docs and firewall diagrams
- Verified connectivity via `mysql` and `psql` CLI

2. Spot Management Service on Cloud Run

Purpose: Build the core spots microservice with FastAPI + Postgres.
Responsibilities:

- Containerize (`Dockerfile`) and deploy to Cloud Run
- Implement FastAPI endpoints with SQLAlchemy connected to Cloud SQL Postgres
- Add ETag support on `GET /studyspots/{id}`
- Implement query parameters and filters on all collection endpoints (e.g. /studyspots?city=&open_now=)
- Implement pagination on GET /studyspots with page and page_size
- Return 201 Created + Location header for POST requests
- Add linked data with relative paths (links.self, links.reviews)
- Implement a 202 Accepted asynchronous operation (POST /studyspots/{id}/geocode)
- - Returns 202 + Location: /jobs/{job_id}
- - Provide GET /jobs/{job_id} polling endpoint that returns {"status": "pending|running|complete", "result": {...}}
- Deliverables:
- Working Cloud Run deployment URL

- OpenAPI docs showing eTag, pagination, filters, 201 + 202 async examples
- Demo data visible in Cloud SQL (Postgres)

---

3. User Management Service on GCE VM

Purpose: Finalize auth and preferences logic; connect to Cloud SQL MySQL.
Responsibilities:

- Configure VM service (Nginx + Uvicorn + TLS)
- Implement authentication (`/auth/register|login|logout|me`)
- Add pagination and filters to `GET /users`
- Add ETag handling on `GET /users/{id}` + PUT with `If-Match`
- Validate sessions and preferences endpoints
  Deliverables:
- VM service live on HTTP port 8002
- Swagger docs showing auth routes
- README explaining deployment and systemd setup

---

4. Reviews and Ratings Service on VM

Purpose: Implement review and rating CRUD backed by the MySQL VM.
Responsibilities:

- Connect FastAPI to MySQL VM
- Complete CRUD for reviews and ratings and delete flows
- Implement average rating query and optimize indexes
- Include linked data (`links.spot`, `links.user`)
- Return 201/204 responses properly
  Deliverables:
- Working endpoints reachable from browser and composite service
- Proof of optimized query and index plan (EXPLAIN)

---

5. Composite Gateway Service

Purpose: Central API layer aggregating the 3 atomic microservices.
Responsibilities:

- Build new FastAPI service deployed on Cloud Run

- Implements the public-facing aggregation layer that encapsulates and delegates to the atomic microservices.
- Implement façade routes (e.g., /api/spots, /api/users, /api/reviews) that mirror atomic APIs and forward requests to them.
- Implement threaded fan-out for parallel calls (aggregate spot + reviews + user in one response).
- Validate logical foreign keys (ensure user and spot exist) before creating composite reviews.
- Optionally proxy Spot Service's 202 async geocode job for composite clients (the primary 202 implementation resides in Spot Management).

  Deliverables:

- /composite/spots/{id}/full → aggregates Spot + Reviews + Ratings
- /api/spots, /api/users, /api/reviews → façade routes delegating to atomic APIs
- README and OpenAPI spec describing parallelization and FK enforcement

---

6. Web UI on Cloud Storage

Purpose: Build and deploy the frontend for users to interact with the system.
Responsibilities:

- Develop React + Vite frontend in `web-app`
- Integrate with Spot, Composite, and User services (via fetch API)
- Show spots with filters and pagination
- Detail page fetching composite data
- Form to add review → 202 job feedback (polling)
- Deploy to GCS static hosting (+ CORS configured)
  Deliverables:
- Live static URL on GCS (e.g., https://storage.googleapis.com/studyspots-ui/index.html)
- Demonstrate 202 async job handling by submitting a geocode request and polling job status until complete.
- Screenshots and demo GIF
- README with deploy commands

---

7. CI/CD & QA Automation

Purpose: Maintain project organization, testing, and deployment standards.
Responsibilities:

- Set up GitHub Actions (build/test/lint/deploy)
- Create PR template with definition of done checks
- Write E2E script that creates user → spot → review → fetch composite
- Manage Kanban board status and team updates
- Prepare Sprint 2 demo recording and report

Deliverables:

- Green build badges on repos
- Passing E2E demo script in CI
- Updated GitHub Project board with all cards linked

Title	Owner	Repo	Acceptance Criteria	Checklist / Sub-tasks
Cloud DB & Networking Setup	Brian	`mysql-db`	All 3 DBs live; private IPs; VPC access; schemas seeded	☐ Provision PG Cloud SQL ☐ Provision MySQL Cloud SQL ✅ Provision VM MySQL ✅ Firewall/VPC ☐ Seed data ☐ Docs ready
Spot Management Microservice	Erica	`spot-management`	Cloud Run deployed; DB connected; pagination, filters, ETag, 201 POST working	✅ Dockerfile ✅ Cloud Run deploy ☐ Implement filters/ pagination ☐ Add ETag ✅ POST 201 ☐ Linked data ☐ OpenAPI docs
User Management Microservice		`user-management`	Auth routes implemented; ETag support; VM + Cloud SQL connection verified	☐ Nginx/TLS ☐ Connect to Cloud SQL ☐ ETag + If-Match ☐ Pagination ☐ Auth flows

				☐ Docs updated
Reviews & Ratings Microservice		`reviews-ratings`	VM to VM MySQL link works; CRUD complete; avg rating optimized (< 50 ms)	☐ Connect MySQL VM ☐ CRUD reviews/ratings ☐ Avg rating endpoint ☐ Add links ☐ Optimize index ☐ OpenAPI
Composite Gateway Microservice		`composite-gateway`	Aggregates atomic APIs; parallel threads; FK validation; 202 async job	☐ Repo setup ☐ Threaded fan-out ☐ FK check POST /review ☐ Async geocode (202) ☐ Polling endpoint ☐ Docs
Web UI on Cloud Storage	Bharadwaj	`web-app`	GCS deployed SPA interacting with Composite + Spots APIs (ETag/pagination)	☐ Fetch integration ☐ Pagination UI ☐ Review form ☐ Geocode polling ☐ Deploy GCS ☐ CORS config ☐ Demo GIF
CI/CD + QA Automation	Avi	All	CI builds green; PR template ready; E2E demo script runs successfully, write up sprint 2 deliverables	☐ GitHub Actions per repo ☐ PR template ☐ E2E script ☐ Update Kanban ☐ Sprint report ☐ Demo video

Parallelization Plan

Phase	Work Items (Titles from above)	Who Starts	Dependency	Notes
Phase 1: Infra & DB Foundations	**Cloud DB & Networking Setup**		None	Must complete Cloud SQL + MySQL VM setup before others can connect. Includes VPC connectors and seed data.
Phase 2: Atomic Microservices	**Spot Management**, **User Management**, **Reviews & Ratings**		Depends on Phase 1	Once databases are online, these 3 atomic services can be built and deployed in parallel.
Phase 3: Composite Service	**Composite Gateway Microservice**		Depends on atomic APIs running	Can start stub routes early but needs fully running atomic APIs to aggregate data and test threaded calls.
Phase 4: Web UI	**Web UI on Cloud Storage**		Depends on Composite and Spot APIs	Can start frontend design immediately; API integration happens after Composite endpoints are deployed.
Phase 5: CI/CD & QA	**CI/CD + QA Automation**		Continuous across all phases	Starts on day one (lint, PR templates, CI setup) and expands once each service is live to add end-to-end testing.