🧭 CollabSpace Multi-Service Collaboration Plan

> ***Goal:*** *Deliver a modular, production-grade system where each team owns a cleanly isolated service but all connect seamlessly through a shared API gateway.*

🌍 1. Final System Overview

Layer	Role	Managed By	DB
🟩 **Node API Gateway**	Entry point for all frontend traffic. Auth, proxy routing, token validation.	Node Team (Gateway Squad)	Postgres (users/auth)
🟦 **Java Service**	Core business logic — projects, tasks, teams.	Java Squad	Postgres (projects)
🟩 **Node Realtime Service**	Chat, notifications, comments, live updates via WebSockets.	Node Realtime Squad	Redis or MongoDB
🟥 **Angular Frontend**	Single UI talking only to API Gateway.	Frontend Squad	—

✅ Gateway = "traffic control tower." ✅ Java = "project factory." ✅ Realtime Node = "chat heartbeat." ✅ Angular = "control panel."

🧱 2. Repository Creation & Structure

Each squad will own a separate repo. The repos **must** follow a common naming, environment, and documentation standard.

Repo Name	Purpose	Responsible Team
`collabspace-gateway`	Authentication, API proxy, entry point	Node Gateway Team
`collabspace-backend-java`	Core domain logic (projects, teams, tasks)	Java Team
`collabspace-realtime`	WebSocket/notification service	Node Realtime Team
`collabspace-frontend`	Angular client UI	Angular Team
(optional) `collabspace-dev-env`	Shared `docker-compose` for all services	Scrum Master / DevOps

---

📁 Common Folder Pattern

Every repo follows a predictable structure:

```
/src
  ├── controllers/
  ├── services/
  ├── routes/
  ├── models/
  ├── config/
  ├── tests/
.env.template
Dockerfile
README.md
```

🔑 3. Shared Standards (All Teams)

Element	Standard	Notes
JWT Secret	Same across all backends	Stored in `.env`
Enum casing	uppercase (`TODO`, `IN_PROGRESS`, `DONE`)	Avoid mismatches
Error format	`{ "error": "message", "code": 400 }`	Unified API design
Date format	ISO UTC (`2025-10-28T15:00:00Z`)	Consistency across DBs
Branching	`main` (stable), `dev` (active sprint), feature branches per module	No direct commits to main
API Docs	Swagger / Postman collection	Shared across repos
.env.template	Must include all needed variables	Copied from gateway repo

---

⚙️ 4. Phase-by-Phase Execution Plan

🪜 Phase 1 — Repository Setup & Alignment (Day 1–2)

Task	Owner	Deliverable
Create all repos with base structure	All Teams	4 clean repos initialized
Set up `.env.template` and shared variables	Gateway Team	Template shared to all
Establish shared `docker-compose` file	Scrum Master	Base stack spins up cleanly
Write basic README + setup guide	Each Team	Dev onboarding ready

---

🟩 Phase 2 — Node API Gateway (Day 2–4)

Focus	Deliverables
Auth (login/register)	JWT issuance, password hashing
Middleware	`verifyToken`, `checkRole`
Proxy setup	Routes to Java (`/projects`) and Realtime (`/realtime`)
Error handling	Global handler, standardized responses
Docs	Swagger / Postman routes ready

Repo: `collabspace-gateway`

Endpoints:

- `/api/auth/login`
- `/api/auth/register`
- `/api/projects/*` → proxy → Java
- `/api/realtime/*` → proxy → Realtime

🟦 Phase 3 — Java Backend (Day 3–5)

Focus	Deliverables
Entities	`projects`, `tasks`, `teams`
Team logic	Cached team model (Option B)
DB	PostgreSQL with Flyway/JPA schema migration
Security	Verify headers `X-User-Id`, `X-User-Role`
Testing	Integration + controller tests

Repo: `collabspace-backend-java`

🟩 Phase 4 — Realtime Backend (Day 4–6)

Focus	Deliverables
Socket.IO server	Handle room joins, emits, notifications, task comments
Auth	JWT verification at socket connection
Pub/Sub	Redis integration for message broadcasting
Chat model	Basic persistence in Redis/Mongo
Internal event route	`/notify` (Java → Realtime) for updates
Testing	Simulated socket client integration tests

Repo: `collabspace-realtime`

🟥 Phase 5 — Angular Frontend (Day 3–7)

Focus	Deliverables
Auth pages	Login/signup with JWT storage
Dashboard	Fetch `/api/projects`
Chat UI	Connect to Realtime WebSocket
Notification panel	Listen for push updates
Shared UI standards	Angular Material / Tailwind setup

Repo: `collabspace-frontend`

🤿 Phase 6 — Integration & Docker Testing (Day 6–7)

Task	Responsible	Deliverable
Merge Dockerfiles into `docker-compose.yml`	Scrum Master + Node Dev	Unified container setup
Network test	All	Verify Gateway ↔ Java ↔ Realtime communication
End-to-end test	Angular + Node	Confirm full login → project → chat flow
Documentation	Scrum Master	Architecture.md finalized

Run Command:

```
docker-compose up --build
```

🔄 5. Communication Protocol

Channel	Purpose	Frequency
Daily Standup	Quick status check + blockers	Every morning
Notion Board	Shared schema, API contract, tasks	Continuous
GitHub Issues	Bugs, features, PR tracking	Daily
Integration Call	Verify cross-service APIs	Mid-week & end-week

---

🧩 6. Integration Contract (For All APIs)

Request Headers (Gateway → Internal Services):

```
X-User-Id: <uuid>
X-User-Role: <role>
Authorization: Bearer <jwt>
```

Response Format:

```
{
  "success": true,
  "data": {...},
  "timestamp": "2025-10-30T10:00:00Z"
}
```

Error Format:

```
{
  "error": "Invalid token",
  "code": 401
}
```

🔒 7. Security Enforcement

Layer	Security Role
Gateway	Validates tokens, rate limits, CORS
Java	Accepts only internal traffic from Gateway
Realtime Node	Validates token at socket connect
Docker Network	Private internal service communication
DBs	No external ports exposed

🧠 8. Final Rules for Success

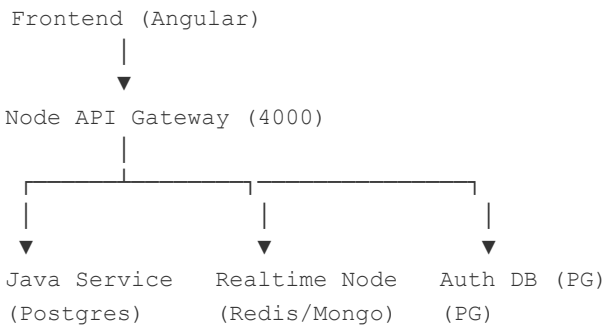
✅ One shared JWT secret (rotate per sprint). ✅ Each repo must be **runnable independently**. ✅ Keep `swagger.yaml` or Postman collection updated weekly. ✅ Follow the **naming conventions** in the schema doc. ✅ Never directly call another service's DB — always use HTTP or event triggers. ✅ All environment variables documented in `.env.template`.

📄 9. Deliverables by End of Sprint

Team	Key Deliverable
🟩 Node Gateway	Running service on port 4000 with Auth + Proxy

Team	Key Deliverable
🟦 Java Service	CRUD + team cache + DB migrations
🟩 Realtime Service	WebSocket chat + notifications
🟥 Angular Frontend	Integrated UI for projects & chat
🧱 Scrum Master	Shared `docker-compose.yml`, full integration test

🏁 10. End-Goal Architecture Summary



→ All authenticated, isolated, scalable, and communicating through the Gateway.

Perfect 👏 — let's lock this down with a **clean, visual sprint workflow board** for your whole CollabSpace squad. This one's built to look great in Notion, Jira, or any Scrum board — readable, structured, and day-by-day aligned.

🧩 Team Overview

Squad	Repo	Focus Area
🟩 Node Gateway Team	`collabspace-gateway`	Auth, API routing, security
🟦 Java Backend Team	`collabspace-backend-java`	Projects, tasks, team management
🟩 Realtime Node Team	`collabspace-realtime`	Chat, notifications, WebSockets
🟥 Angular Frontend Team	`collabspace-frontend`	UI, dashboard, chat interface
🧱 Scrum Master / DevOps	`collabspace-dev-env`	Integration, Docker, coordination

---

🗓️ Sprint Calendar: 7-Day Breakdown

Day	Gateway Team (Node)	Java Team (Spring Boot)	Realtime Team (Node)	Angular Team	Scrum Master / DevOps
Day 1 – Setup	🔧 Initialize repo 📦 Install Express,	🟤 Init Spring Boot project	⚙️ Initialize project 📦 Setup Socket.IO	🎨 Initialize Angular app	🧱 Create shared

Day	Gateway Team (Node)	Java Team (Spring Boot)	Realtime Team (Node)	Angular Team	Scrum Master / DevOps
Monday	TS, Prisma 🧱 Setup `tsconfig.json`, `eslint`, `.env.template`	📂 Set up Postgres connection & schema	+ Redis base	🔧 Setup routing & services	`docker-compose.yml` 📄 Draft architecture doc
Day 2 – Auth & Schema	🔐 Implement `/auth/register` + `/auth/login` 🧩 JWT middleware	🧱 Create entities (Project, Task, Team) 🧩 Configure JPA mappings	⚡ Setup socket connection & auth handshake	🔑 Build login/signup UI 💾 Handle JWT in memory	🤝 Verify all teams share env variables
Day 3 – Core Routing	🚦 Implement Proxy Middleware 🧩 `/api/projects` routes → Java	🧱 Implement CRUD for Projects & Tasks 🧩 Expose `/api/projects`	💬 Setup event structure (`project:update`, `chat:new`)	📊 Implement Projects Dashboard UI	🧰 Integrate all services under Docker
Day 4 – Integration Prep	✅ Test proxying with Java container 📑 Setup Swagger docs	🧩 Implement Teams (cached model) 🧠 Add event hooks for Realtime	🔔 Handle event broadcast from Java	🧭 Connect dashboard with gateway APIs	🧪 Validate container network (Ping Gateway ↔ Java)
Day 5 – Realtime Connect	🕯️ Expose `/api/realtime` endpoint	🧩 Add notification endpoints 📮 POST to Realtime service	🚀 Handle live updates via Redis 🧠 Add socket rooms	💬 Integrate chat & notifications	📓 Collect integration test logs
Day 6 – Testing & QA	🧪 Jest tests for Auth & Proxy	🧪 Spring Boot integration tests	🧪 Simulate concurrent sockets	🧪 Full flow test (login → project → chat)	🧱 Fix Docker bugs, finalize docs
Day 7 – Wrap & Demo	📦 Polish, lint, finalize README	🎥 Prepare endpoints for demo	🎥 Test multi-user socket load	🎥 UI polish & responsive design	🎤 Conduct final integration demo

---

🧩 Integration Milestones

Milestone	Target Day	Owner	Success Criteria
M1: Gateway ↔ Java Proxy Works	Day 3	Node + Java	`/api/projects` returns data via proxy

Milestone	Target Day	Owner	Success Criteria
M2: Auth Verified via Gateway	Day 4	Node + Angular	Login & JWT working end-to-end
M3: Java ↔ Realtime Event Trigger	Day 5	Java + Realtime	Realtime receives update when task changes
M4: Full Frontend → Gateway → Java → Realtime Loop	Day 6	All Teams	User adds task → Realtime broadcasts it
M5: Docker Integration Pass	Day 6	DevOps	`docker-compose up` spins up full system
M6: Sprint Demo	Day 7	Scrum Master	Functional end-to-end demo delivered

---

🧱 Shared Artifacts (All Teams Must Update)

Artifact	Location	Maintained By
`swagger.yaml`	`/docs` in Gateway repo	Node + Java
`.env.template`	Root of each repo	Scrum Master
`docker-compose.yml`	`collabspace-dev-env`	DevOps
`API_CONTRACT.md`	Shared Notion Doc	Node + Java
`TEAM_ROLES.md`	Notion	Scrum Master

🔒 Final Quality Checks

✅ All repos run independently via `npm run dev` or `mvn spring-boot:run`. ✅ Gateway rejects unauthenticated requests. ✅ All JWTs validated across services. ✅ Frontend never directly calls Java or Realtime. ✅ Realtime updates broadcast correctly. ✅ Postman collection or Swagger UI runs end-to-end.

🚀 Expected Sprint Outcome

> *By **Day 7**, your system runs fully containerized:*
>
> - *Angular → Gateway → Java → DB*
> - *Angular → Gateway → Realtime → Socket event*
> - *Gateway handles all security*
> - *Java owns all business logic*
> - *Realtime handles all live communication*
> - *Every service deploys separately but integrates seamlessly* 🎯