Arhitekturni dokument – Faza 1

Projekat: DrawSync

Autori:

Aleksandar Mitić 19253

Natalija Nikolić 19286

Datum: 18.01.2026.

Bordo tim	Verzija: 1.0
Arhitekturni izveštaj - faza I	Datum 18.01.2026.
DrawSync	

Pregled izmena

Datum	Verzija	Opis	Autor
18.01.2026.	1.0		Natalija, Aleksandar

Arhitekturni dokument – Faza I

1. Kontekst i cilj softverskog projekta

1.1 Kontekst projekta

DrawSync je web aplikacija za multiplayer crtanje u realnom vremenu, inspirisana igrama tipa scribbl.io. Aplikacija omogućava većem broju korisnika da istovremeno učestvuju u igri u okviru nezavisnih soba, gde jedan korisnik crta na zajedničkom platnu, a ostali pokušavaju da pogode zadatu reč. Svi potezi crtanja i događaji u igri sinhronizuju se u realnom vremenu i trajno čuvaju u bazi podataka.

Projekat se realizuje u kontekstu potrebe za sistemima koji podržavaju istovremeni rad više klijenata nad istim "dokumentom", uz jasna pravila konkurentnog pristupa i pouzdanu perzistenciju izmena. DrawSync je pogodan primer takvog sistema jer kombinuje real-time komunikaciju, kontrolu konkurentnosti i event-driven arhitekturu.

1.2 Cilj projekta

Cilj projekta DrawSync je razvoj skalabilnog i modularnog softverskog sistema koji omogućava real-time multiplayer igru crtanja i pogađanja, uz doslednu kontrolu konkurentnog pristupa i pouzdanu perzistenciju događaja.

Konkretni ciljevi projekta su:

- omogućiti real-time sinhronizaciju poteza crtanja i događaja igre između više klijenata,
- obezbediti konzistentnost stanja sobe primenom pesimističkog zaključavanja (jedan crtač u datom trenutku),
- omogućiti perzistenciju svih relevantnih događaja radi kasnijeg pregleda i rekonstrukcije (replay),
- projektovati arhitekturu sa jasnom podelom odgovornosti (Onion/Clean), nezavisnu od konkretnih tehnologija,
- obezbediti rad više disjunktnih grupa korisnika (više soba) bez međusobnog uticaja.

2. Arhitekturno specifični zahtevi

2.1 Funkcionalni zahtevi

1) Perzistencija domen objekata
   Sistem mora omogućiti trajno čuvanje elemenata domena aplikacije, uključujući sobe, korisnike u sobama, runde igre, poteze crtanja, poruke i događaje, korišćenjem baze podataka i odgovarajućeg ORM rešenja.
2) Real-time saradnja na dokumentu
   Više klijenata mora istovremeno posmatrati isti dokument (platno) i dobijati izmene u realnom vremenu.
3) Pessimistic lock za izmenu dokumenta
   U jednoj sobi samo jedan korisnik u datom trenutku ima pravo izmene dokumenta (crtač), dok ostali korisnici imaju read-only prikaz. Pravo izmene se prenosi na sledećeg korisnika po završetku runde ili napuštanju sobe.
4) Message broker komunikacija i perzistencija događaja
   Sistem mora koristiti message broker za razmenu događaja između komponenti i imati poseban modul za perzistenciju izmena i događaja.
5) Podrška za više dokumenata
   Sistem mora omogućiti istovremeni rad većeg broja disjunktnih grupa korisnika nad različitim dokumentima (sobe).
6) Različite uloge korisnika
   Aplikacija mora podržati različite uloge korisnika (crtač, pogađač, posmatrač) sa različitim dozvoljenim akcijama i vizuelizacijama.
7) Ograničenja u zavisnosti od stanja modela
   Dozvoljene akcije nad dokumentom i igrom zavise od trenutnog stanja sistema (npr. crtanje je dozvoljeno samo tokom aktivne runde).
8) Pregled istorije i replay
   Sistem mora omogućiti pregled istorije igre i rekonstrukciju stanja dokumenta na osnovu perzistiranih događaja.

2.2 Nefunkcionalni zahtevi

1) Performanse i nisko kašnjenje
   Sistem mora obezbediti nisko kašnjenje u prikazu poteza crtanja i događaja igre u realnom vremenu.
2) Konzistentnost stanja
   U jednom trenutku ne sme postojati više od jednog korisnika sa read-write privilegijama nad istim dokumentom.
3) Skalabilnost
   Sistem mora omogućiti rast broja korisnika i soba bez međusobnog uticaja na performanse.
4) Pouzdanost
   Događaji crtanja i igre moraju biti pouzdano sačuvani čak i u slučaju privremenih problema sa infrastrukturom.
5) Održivost i testabilnost
   Arhitektura sistema mora omogućiti jednostavno testiranje domen logike i laku zamenu infrastrukturnih komponenti.
6) Bezbednost
   Sistem mora obezbediti osnovnu autentikaciju korisnika i validaciju ulaznih podataka na domenskom nivou.

2.3 Tehnička ograničenja

- Sistem se implementira kao web aplikacija.
- Klijentski deo se realizuje korišćenjem Angular framework-a.
- Serverski deo se realizuje korišćenjem NestJS framework-a.
- Real-time komunikacija se ostvaruje putem WebSocket-a.
- Message broker rešenje: RabbitMQ.
- Baza podataka: PostgreSQL uz TypeORM.
- Lokalni razvoj i pokretanje sistema realizuju se pomoću Docker Compose-a.

2.4 Poslovna ograničenja

Razvoj sistema DrawSync realizuje se u okviru akademskog projekta i podleže sledećim poslovnim ograničenjima:

Rok izrade projekta je ograničen i definisan nastavnim planom predmeta, pri čemu se faza arhitekture mora završiti do 21.01.2026. godine.

Implementacija mora obavezno uključivati perzistenciju podataka, real-time komunikaciju i message broker, u skladu sa zahtevima zadatka.

Projekat se predaje kroz GitHub repozitorijum koji sadrži izvorni kod, bazu podataka i prateću dokumentaciju, uključujući opis arhitekture i UML dijagrame.

3. Arhitekturni dizajn softverskog sistema

Arhitektura sistema DrawSync projektovana je kao **slojni, distribuirani i event-driven sistem**, koji kombinuje klijent–server model, real-time komunikaciju i asinhronu perzistenciju događaja. Cilj arhitekture je da obezbedi nisko kašnjenje u realnom vremenu, konzistentnost stanja igre i jasnu podelu odgovornosti između komponenti sistema.

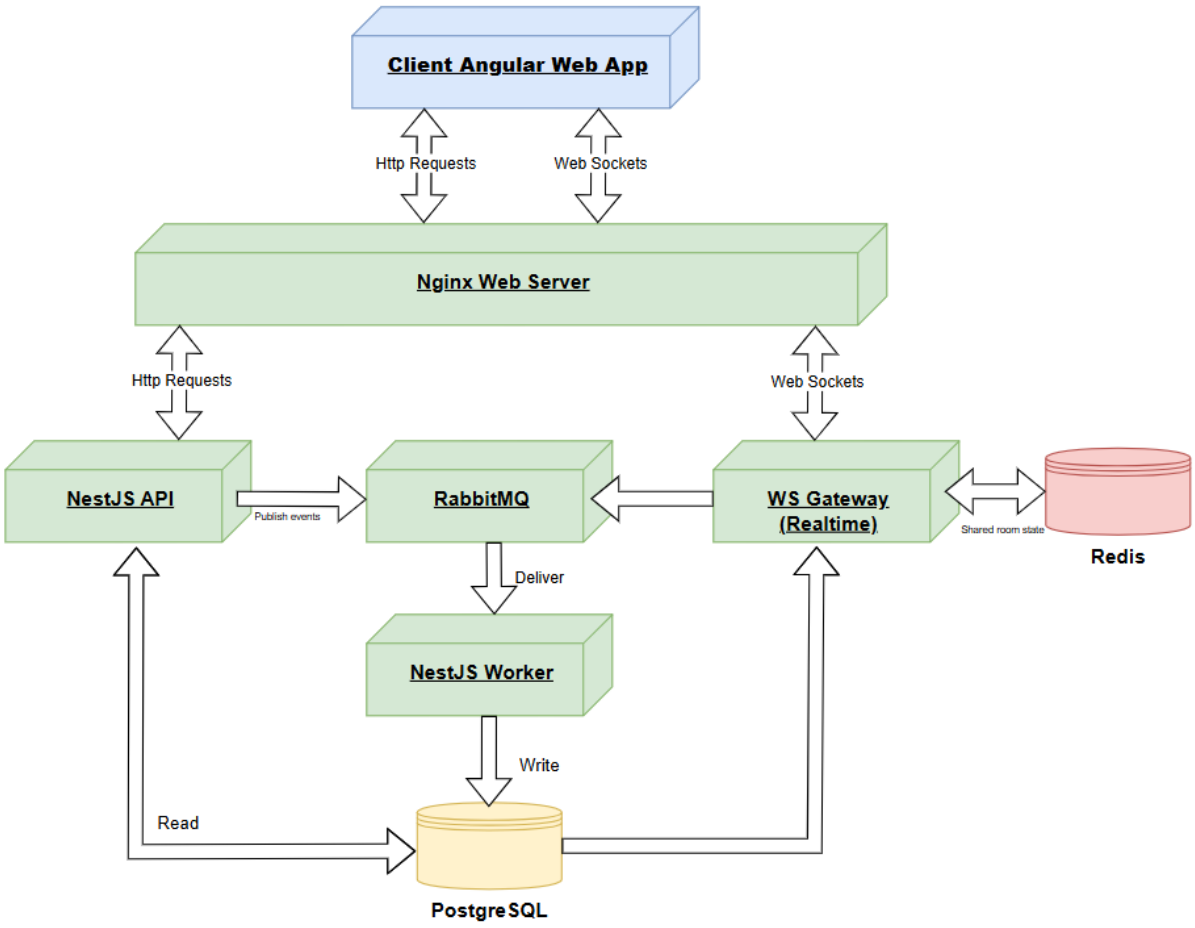
Sistem se sastoji od klijentskog dela, serverskog API sloja, real-time komunikacionog sloja, message broker-a, radnih (worker) servisa i perzistentnog skladišta podataka.

3.1 Arhitekturni obrasci koji će biti korišćeni

U sistemu DrawSync primenjuju se sledeći arhitekturni i projektni obrasci:

1) Client–Server arhitektura
   Sistem je organizovan po client–server modelu, gde Angular aplikacija predstavlja klijenta, a NestJS aplikacija serversku stranu sistema. Klijent komunicira sa serverom putem HTTP (REST) i WebSocket protokola.
2) Onion / Clean Architecture
   Serverski deo sistema projektovan je prema Onion arhitekturi, gde je domen u centru, a infrastruktura (baza, message broker, transport) u spoljašnjim slojevima. Ovim pristupom se postiže nezavisnost domenske logike od konkretnih tehnologija.
3) Ports & Adapters (Hexagonal Architecture)
   Komunikacija između API sloja i domena realizuje se preko jasno definisanih portova (ClientApi, DomainApi), dok adapteri služe za mapiranje ulaza i izlaza.
4) Pub/Sub (Publish–Subscribe)
   RabbitMQ se koristi za asinhronu distribuciju događaja (potezi crtanja, završetak runde), čime se razdvaja real-time komunikacija od perzistencije.
5) Repository obrazac
   Pristup bazi podataka realizuje se kroz ORM repozitorijume u infrastrukturnom sloju.
6) Event-driven arhitektura, gde se potezi crtanja i događaji igre tretiraju kao događaji koji se distribuiraju i perzistiraju.

3.2 Pregled glavnih komponenti sistema



3.2.1 Klijentska aplikacija (Client Angular Web App)

Klijentska aplikacija je realizovana kao Angular web aplikacija i predstavlja primarnu tačku interakcije korisnika sa sistemom.
Njene odgovornosti uključuju:

- prikaz korisničkog interfejsa (canvas, chat, stanje sobe),
- slanje HTTP zahteva za standardne operacije (kreiranje i pristup sobama, istorija),
- uspostavljanje WebSocket konekcije za real-time razmenu događaja (potezi crtanja, pokušaji pogađanja).

Klijent ne sadrži poslovnu logiku igre, već isključivo reaguje na događaje koje dobija od servera.

3.2.2 Nginx Web Server

Nginx se koristi kao reverse proxy i web server, sa sledećim ulogama:

- serviranje statičkog Angular build-a,
- prosleđivanje HTTP zahteva ka NestJS API-ju,
- prosleđivanje i održavanje WebSocket konekcija ka real-time gateway-u.

Ovim se postiže jasna ulazna tačka u sistem i razdvajanje klijentskog i serverskog sloja.

3.2.3 NestJS API (HTTP sloj)

NestJS API predstavlja serverski sloj za HTTP komunikaciju.
Njegove glavne odgovornosti su:

- obrada REST zahteva (upravljanje sobama, listing, istorija/replay),
- validacija i mapiranje ulaznih podataka,
- iniciranje domen operacija i objavljivanje domen događaja u message broker.

API sloj ne vrši perzistenciju događaja direktno, već objavljuje događaje u RabbitMQ, čime se razdvaja poslovna logika od perzistencije.

3.2.4 WebSocket Gateway (Realtime sloj)

WebSocket Gateway je odgovoran za real-time komunikaciju između servera i klijenata.
Njegove uloge uključuju:

- prijem real-time događaja (potezi crtanja, pogađanja, promene stanja),
- emitovanje događaja svim učesnicima u odgovarajućoj sobi,
- održavanje aktivnih WebSocket konekcija.

Gateway koristi Redis kao zajedničko skladište stanja sobe (shared room state), gde se čuvaju informacije o:

- aktivnim korisnicima,
- trenutnom crtaču (lock owner),
- statusu runde i tajmerima.

Po potrebi, gateway može koristiti bazu podataka za inicijalnu sinhronizaciju stanja prilikom ulaska korisnika u sobu.

3.2.5 RabbitMQ (Message Broker)

RabbitMQ se koristi kao centralni message broker za asinhronu razmenu događaja.
Njegova uloga je:

- prijem događaja koje objavljuju NestJS API i WebSocket Gateway,
- pouzdana isporuka događaja radnim (worker) servisima,
- razdvajanje real-time toka od perzistencije.

Ovakav pristup omogućava otpornost sistema na privremene greške i lakše skaliranje.

3.2.6 NestJS Worker (Persistence servis)

NestJS Worker predstavlja poseban servis za obradu i perzistenciju događaja.
Njegove odgovornosti su:

- konzumiranje događaja iz RabbitMQ,
- upis poteza crtanja, poruka i događaja runde u bazu podataka,
- garantovanje da se svi relevantni događaji trajno sačuvaju.

Worker je jedina komponenta sistema koja vrši write operacije nad bazom podataka za događaje igre, čime se obezbeđuje konzistentnost i jednostavnija kontrola konkurentnosti.

3.2.7 PostgreSQL (Baza podataka)

PostgreSQL predstavlja centralno perzistentno skladište sistema.
U bazi se čuvaju:

- podaci o sobama i korisnicima,
- rezultati rundi i skorovi,
- događaji igre (stroke events, chat, round events) u vidu event log-a.

Baza omogućava rekonstrukciju stanja igre i replay istorije.

3.3 Tokovi komunikacije (sažetak)

- HTTP tok: Angular → Nginx → NestJS API → PostgreSQL (read)
- Real-time tok: Angular → Nginx → WS Gateway → klijenti u sobi
- Event tok: API / WS Gateway → RabbitMQ → Worker → PostgreSQL (write)
- State tok: WS Gateway ↔ Redis (shared room state)

3.4 Detaljna arhitektura backenda (NestJS)

3.4.1 Uloga backenda u sistemu

Backend u DrawSync sistemu je podeljen na dva serverska podsistema:

1) NestJS API (HTTP sloj) – obrađuje REST zahteve i obezbeđuje "upravljanje igrom" kroz standardne operacije (npr. kreiranje sobe, listing soba, dohvat istorije/replay).
2) WebSocket Gateway (Realtime sloj) – obrađuje real-time događaje i sinhronizuje stanje sobe i canvas-a između više klijenata u realnom vremenu.

Uz njih postoji i treća serverska komponenta:

3) NestJS Worker – asinhrono obrađuje događaje iz RabbitMQ i vrši perzistenciju u PostgreSQL.

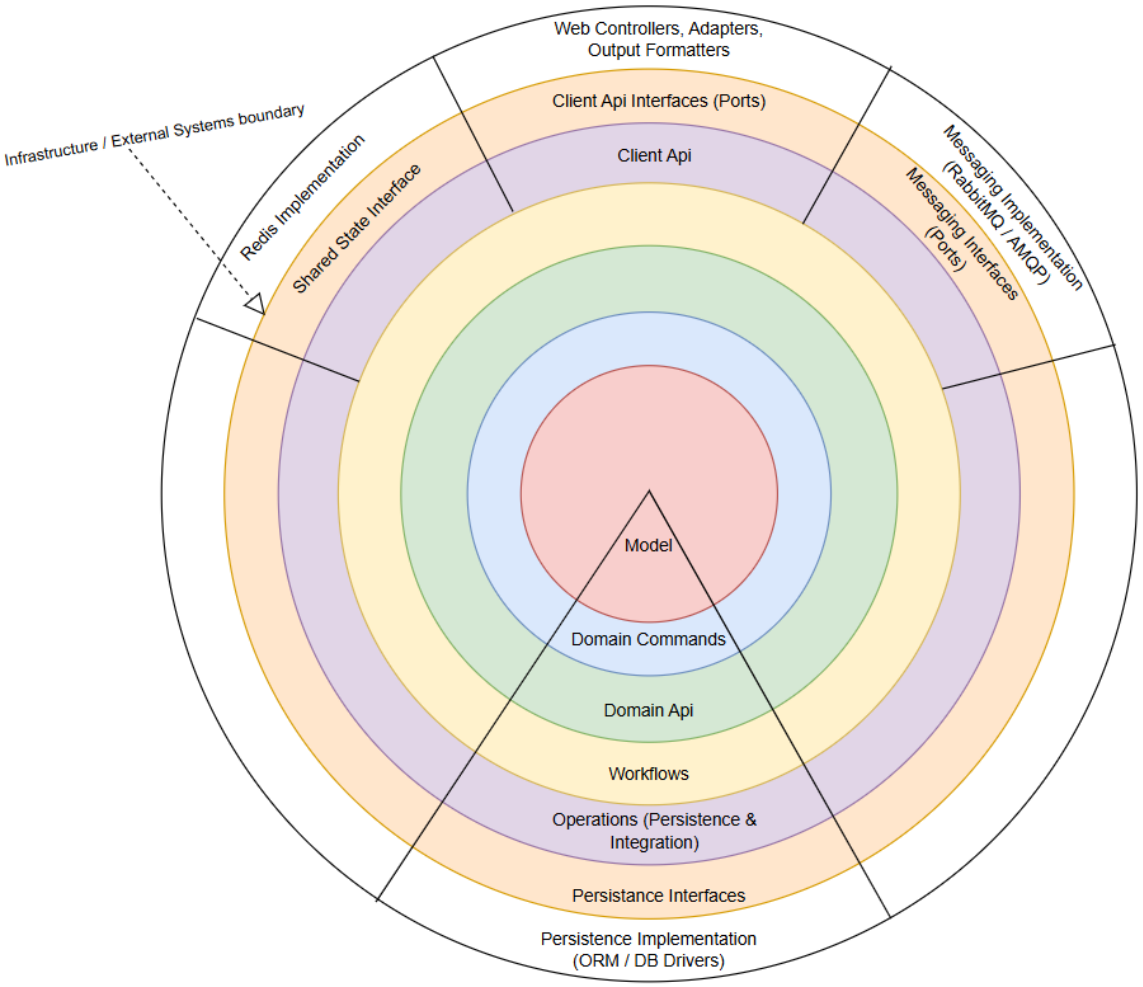
Ova podela je uvedena da bi se:

- odvojio real-time tok od perzistencije,
- obezbedilo nisko kašnjenje (WS) i pouzdanost (broker + worker),
- omogućila jasna odgovornost: API/WS ne "pišu" događaje direktno u bazu, već to radi worker.

3.4.2 Arhitekturni pristup unutar backenda (Onion + Ports & Adapters)

Unutar NestJS aplikacija koristi se Onion/Clean arhitektura i Ports & Adapters pristup. Cilj je da domen logika ostane nezavisna od:

- HTTP i WebSocket transporta,
- baze i ORM-a,
- RabbitMQ i AMQP protokola,
- Redis mehanizama.



Dijagram prikazuje Onion/Clean arhitekturu sistema, u kojoj se domen nalazi u centru i ne zavisi od infrastrukturnih detalja, dok su perzistencija, messaging i deljeno stanje realizovani u spoljašnjim slojevima kroz jasno definisane portove i adaptere.

Slojevi i odgovornosti

Backend je organizovan kroz sledeće slojeve (od spolja ka unutra):

A) Transport sloj (API/WS)

- Controllers (HTTP): primaju REST zahteve, pozivaju adapter i vraćaju odgovor.

- Gateways (WS): primaju real-time događaje i pozivaju adapter, zatim broadcastuju rezultate.

Ovaj sloj ne sadrži poslovnu logiku.

B) Adapter sloj

- Adapter mapira ulaz (HTTP/WS) u interne DTO objekte i poziva ClientApi.
- Adapter ne radi validaciju poslovnih pravila i ne poziva domen direktno mimo ClientApi sloja.

C) ClientApi sloj (port prema domenu)

- Predstavlja granicu između transport sloja i domenske logike.
- Svaka metoda ClientApi poziva:
    - Workflow (ako postoji orkestracija) ili
    - direktno DomainApi (ako je jednostavan slučaj).

D) Workflow sloj (opcion)

- Predstavlja poslovnu orkestraciju (redosled više domen koraka).
- Ne sadrži validaciju i ne poziva repozitorijume.
- Postoji samo kada use-case zahteva koordinaciju više komandi ili domen API poziva.

E) DomainApi sloj

- Ulazna tačka u domen.
- Svaka metoda DomainApi poziva tačno jednu Domain Command i vraća Result DTO.

F) Domain Command sloj (ključni sloj)

- U ovom sloju se nalazi sva poslovna i scenarijska logika:
    - validacija scenarija,
    - primena pravila igre,
    - kontrola stanja (npr. “može li se crtati”, “ko je crtač”, “da li je runda aktivna”),
    - emitovanje domen događaja,
    - poziv Operation sloja kada je potrebna perzistencija ili integracija.

G) Operation sloj

- Tehnički sloj koji enkapsulira perzistenciju i nisko-nivo operacije:
  - upis/čitanje kroz repozitorijume,
  - eventualno emitovanje događaja nakon uspešnog upisa.
- Domen komanda poziva operations, ali komanda ne komunicira direktno sa bazom.

H) Infrastructure sloj

- Implementacije repozitorijuma (ORM), messaging publish/consume (RabbitMQ), redis store, itd.
- Ovaj sloj može da se menja bez promene domena.

3.5 Integracija sa RabbitMQ i perzistencija događaja

- Backend koristi event-driven princip:
- NestJS API i WS Gateway objavljuju događaje u RabbitMQ (npr. stroke_applied, guess_submitted, round_ended).
- Worker konzumira te događaje i perzistira ih u PostgreSQL kao event log.

Ovim se postiže:

- pouzdana perzistencija,
- odvajanje real-time toka od storage toka,
- mogućnost kasnije rekonstrukcije (replay).

3.6 Redis i shared room state

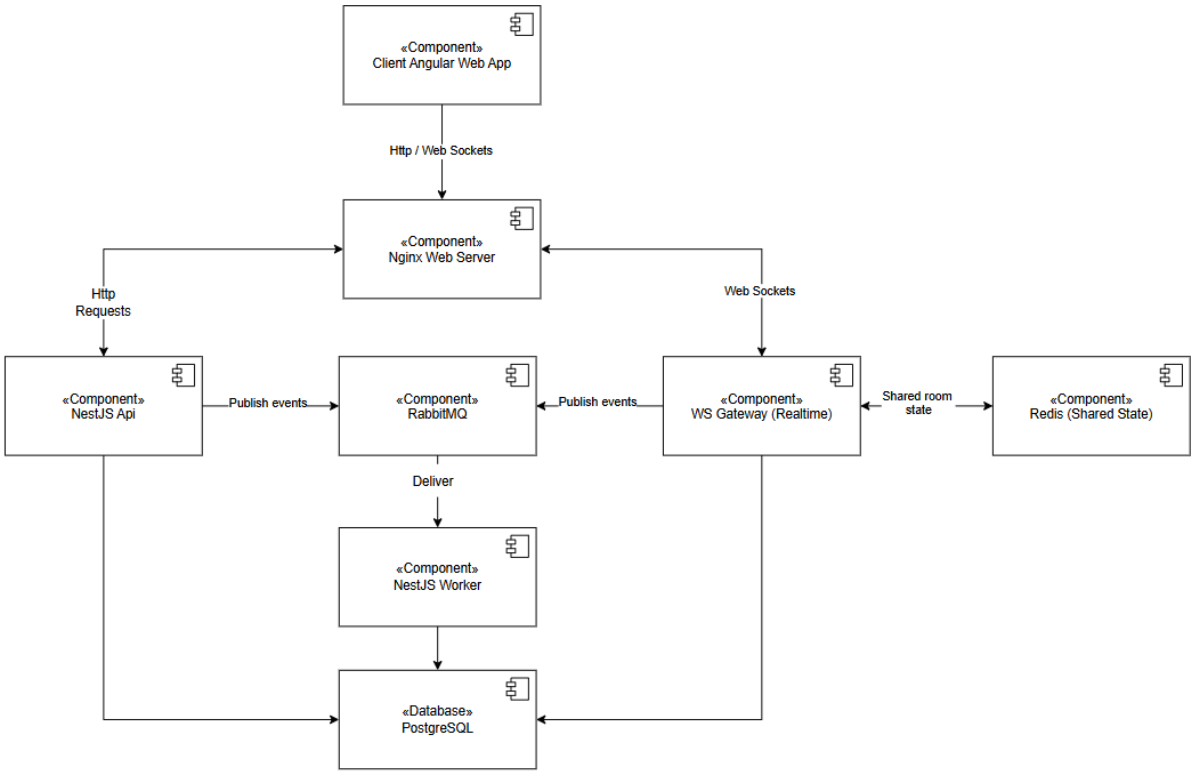
Redis se koristi kao skladište stanja koje je relevantno za real-time:

- lista aktivnih korisnika u sobi,
- trenutni crtač (lock owner),
- status runde i tajmer,
- eventualno “rate limiting” ili presence.

WS Gateway koristi Redis kako bi:

- brzo odgovorio na real-time zahteve,
- omogućio rad i u slučaju više instanci gateway-a (buduća skalabilnost)

3.7 Dijagram komponenti – UML



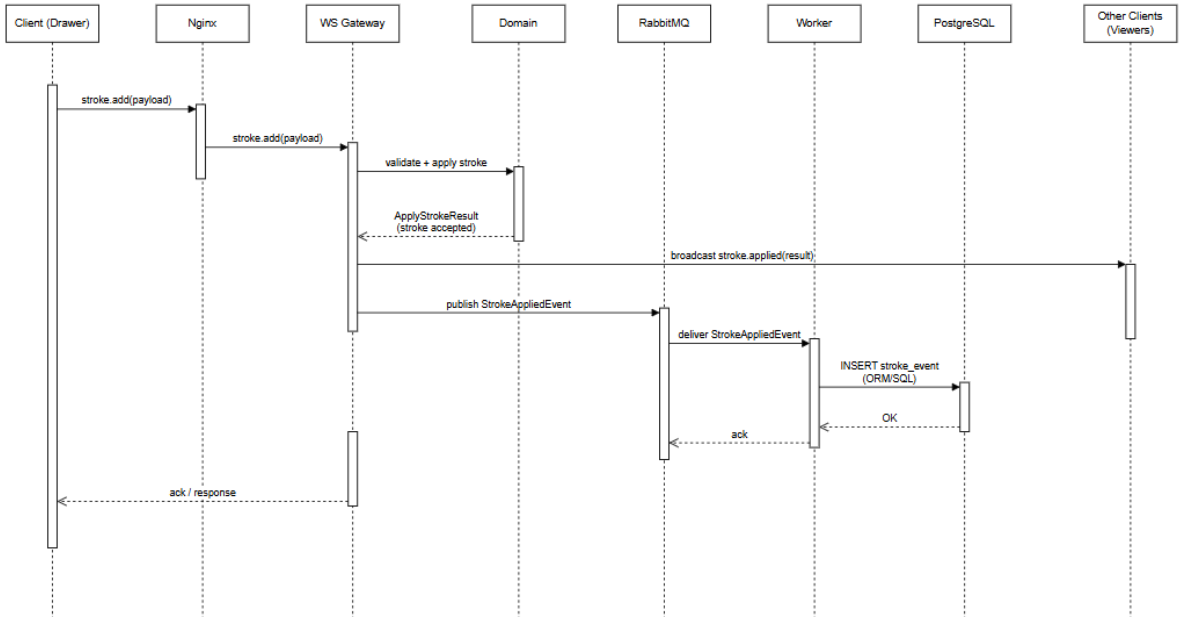
Dijagram prikazuje glavne runtime komponente sistema i njihove komunikacione veze. HTTP i WebSocket komunikacija su jasno razdvojene, dok je perzistencija izdvojena u poseban worker proces koji putem RabbitMQ obrađuje domen događaje i upisuje ih u bazu podataka.

3.8 Bihevioralni dijagram – UML Sequence Diagram (2 scenarija)

1) ApplyStroke (real-time + persistence)

AcquireLock / TransferLock (pessimistic locking)

1) Sequence: ApplyStroke (crtač crta)



Sequence ApplyStroke: Dijagram prikazuje tok real-time poteza sa niskom latencijom (broadcast) i pouzdanom perzistencijom (publish → consume → DB).

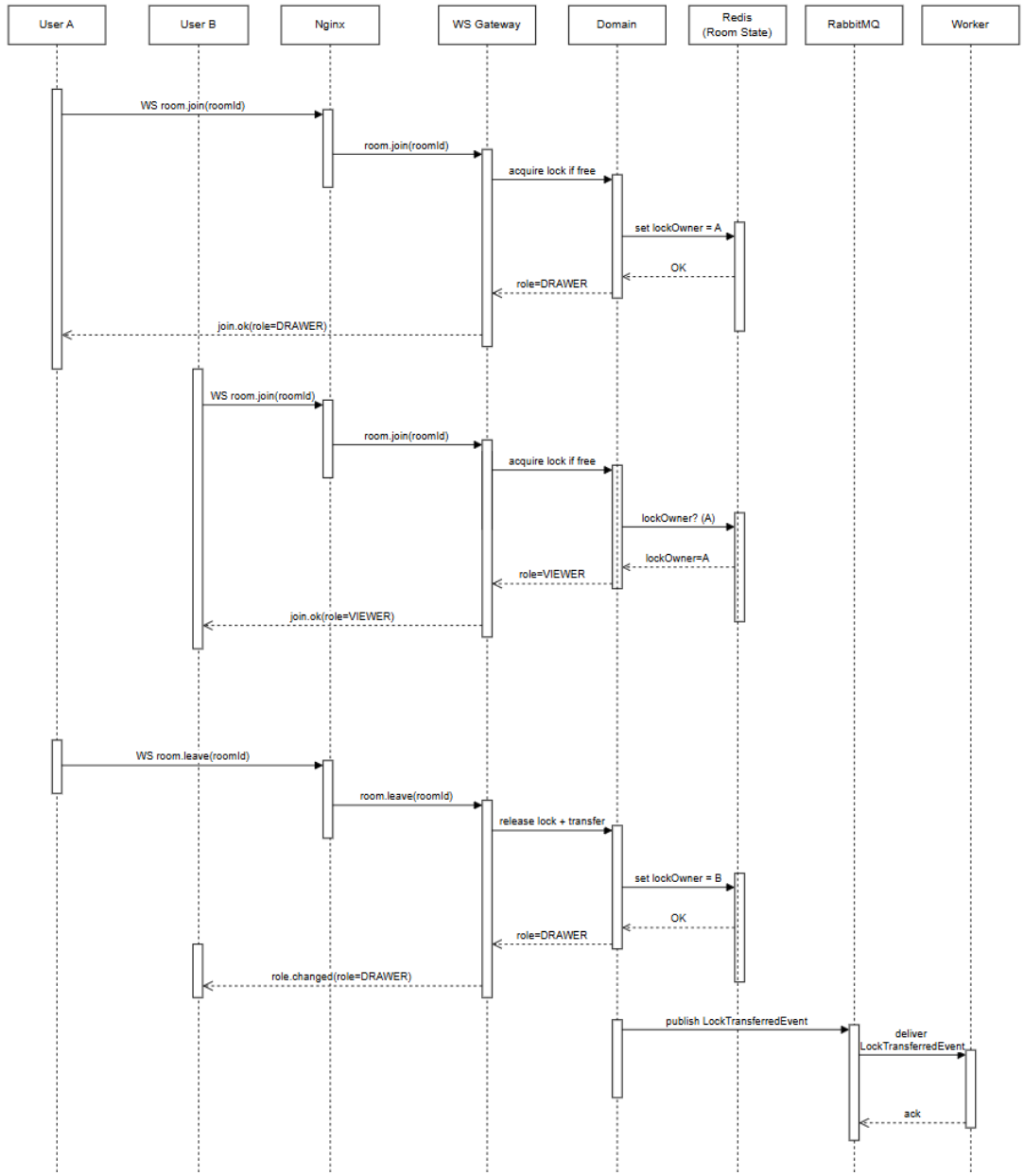
Učesnici (lifelines):

- Client (Drawer)
- Nginx
- WS Gateway
- Domain
- RabbitMQ
- Worker
- PostgreSQL
- Other Clients (Viewers)

Tok:

1. Drawer šalje stroke.add(...) preko WS
2. WS Gateway validira (lock owner + state)
3. WS Gateway broadcastuje potez ostalima (low latency)
4. WS Gateway publishuje event u RabbitMQ
5. Worker konzumira i upisuje u Postgres

1) Sequence: AcquireLock + TransferLock (pessimistic locking)



Sequence Lock: Dijagram prikazuje implementaciju pesimističkog zaključavanja i determinističan transfer write privilegije na sledećeg korisnika.

Učesnici (lifelines):

- User A

- User B
- Nginx
- WS Gateway
- Domain
- Redis
- RabbitMQ
- Worker
- PostgreSQL

Tok (pojednostavljen):

1. User A ulazi u sobu → dobija lock
2. User B ulazi → dobija read-only
3. User A napušta sobu → lock se prebacuje na User B
4. WS gateway update-uje Redis i broadcastuje promenu

4. Tehnološki okviri i obrasci implementacije

Komponenta	Tehnologija	Obrazac (Pattern) / Napomena
Frontend	**Angular**	**MVVM (component-based)**: Template = View, Component class = ViewModel, Services = Model/State. Real-time tok preko WebSocket klijenta; RxJS se koristi za tokove događaja i stanje u UI.
Backend – HTTP	**NestJS (Controllers)**	**MVC-ish + DI/Modularna arhitektura**: Controller je transport sloj (HTTP), mapira request u DTO i poziva Adapter → ClientApi → (Workflow) → DomainApi. Nema poslovne logike u Controlleru.
Backend – Realtime	**NestJS (WebSocket Gateway)**	**Gateway pattern**: WS Gateway prima događaje i prosleđuje ih u istu onion putanju (Adapter/ClientApi/Domain). Broadcast radi preko eventova iz domen rezultata.
Arhitektura backenda	**Onion/Clean + Ports & Adapters**	**Onion Architecture**: Domain (Model/Commands) ne zavisi od infrastrukture. **Ports & Adapters**: ClientApi i Infrastructure interfejsi (npr. EventPublisher, Repository portovi) omogućavaju zamenu transporta i infrastrukture bez promene domena.
Domain logika	(TypeScript, NestJS moduli)	**Command pattern (use-case commands)**: svaka poslovna akcija je Domain Command (validacija + pravila + koordinacija). **Workflow pattern** (opciono): orkestracija više domen koraka. **Result DTO**: domen vraća rezultat nezavisan od HTTP/WS formata.
Perzistencija (ORM)	**TypeORM**	**Data Mapper** (ORM mapira entitete u tabele). **Repository pattern** u infrastrukturnom sloju: repozitorijumi se koriste kroz Operation sloj, ne direktno iz domen komandi.
Messaging	**RabbitMQ**	**Pub/Sub + Asynchronous Messaging**: API/WS objavljuju domen događaje (npr. stroke_applied), Worker konzumira i perzistira u bazu. Broker služi za pouzdan transport događaja i razdvajanje real-time toka od perzistencije.
Worker (Persistence service)	**NestJS Worker**	**Consumer/Worker pattern**: proces koji čita poruke iz RabbitMQ i izvršava perzistencione operacije (upis event log-a, rezultata rundi, poruka).
Baza podataka	**PostgreSQL**	**Relacionalni model + event log**: čuvanje soba, rundi, skorova i događaja (stroke_events). JSON/JSONB se koristi za strukture poteza (npr. lista tačaka poteza) kada je praktično.
Cache/Shared state	**Redis**	**Cache / Shared State store**: čuvanje "shared room state" (lock owner, prisutni korisnici, stanje runde) radi bržeg realtime rada i lakše skalabilnosti WS sloja.

Za implementaciju sistema DrawSync koriste se savremeni web okviri i biblioteke koji prirodno podržavaju zahtevanu arhitekturu i obrasce.

1) Frontend je realizovan korišćenjem Angular framework-a, koji primenjuje komponentnu arhitekturu i može se opisati kao MVVM-pristup, gde su HTML template-i View, Angular komponente ViewModel, a servisi i RxJS tokovi predstavljaju Model/stanje aplikacije.
2) Backend je realizovan korišćenjem NestJS framework-a, koji u transportnom sloju koristi MVC-like pristup (Controllers i WebSocket Gateways), dok je poslovna logika organizovana prema Onion/Clean Architecture i Ports & Adapters principima, čime se obezbeđuje jasna podela odgovornosti i nezavisnost domena od infrastrukture.
3) Za rad sa bazom podataka koristi se TypeORM, koji implementira Data Mapper i Repository obrasce za mapiranje domen entiteta u relacione tabele.
4) Za asinhronu razmenu događaja i razdvajanje real-time toka od perzistencije koristi se RabbitMQ, koji implementira Publish–Subscribe (Pub/Sub) obrazac.
5) Kao perzistentno skladište koristi se PostgreSQL, dok se Redis opciono koristi kao in-memory skladište za deljeno stanje soba u real-time sloju.