Scratch

Aplikacija za interaktivno crtanje i kolaboracije na projektima

Stefan Bogosavljevic 16007, Dusica Janjevic 16116

Opis sistema

Aplikacija je predvidjena da pomaze u kolaboraciji distribuiranih timova. Omogucava svojim koristinicma da kreiraju, cuvaju i organizuju dijagrame i crteze potrebne za resavanje problema na kojima rade.

Funkcionalni zahtevi:

- Kreiranje projekta, dodavanje i uklanjanje clanova tima.
- Kreiranje vise “soba” koje predstavljaju skup tabli vezane za resavanje nekog potproblema.
- Svaka soba mora imati jednu aktivnu tablu u svakom trenutku i skup tabli koje su sacuvane na serveru aplikacije.
- Kreiranje nove prazne table u bilo kojoj sobi.
- Kreiranje dijagrama pomocu skupa unapred definisanih oblika kao i mogucnost crtanja slobodnom rukom.
- Istovremeni rad više korisnika na jednoj tabli
- Kreiranje naloga

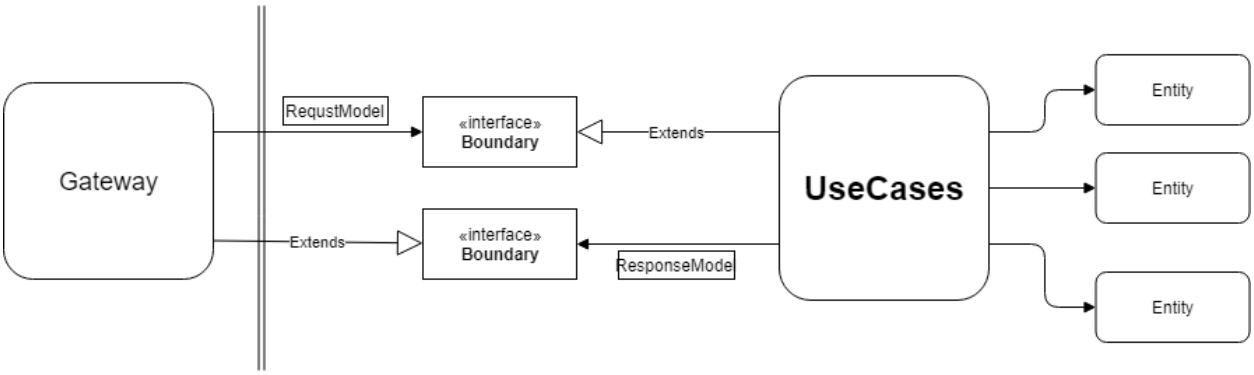
Nefunkcionalni zahtevi:

- Performanse
    - Promene na tablama moraju biti vidljive ostalim clanovima tima u sto je krace moguce vremenskom intervalu.
- Lakoca koriscenja
    - Sistem mora biti intuitivan i lak za navigaciju i upravljanje.
- Lakoca testiranja
    - Izbegavanje direktne povezanosti komponenata koriscenjem interfejsa

Prikaz arhitekture sistema

Clean Architecture

Arhitektura koriscena u sistemu je slojevita po ugledu na *Clean Architecture* dizajn koji je prikazan na slici ispod.

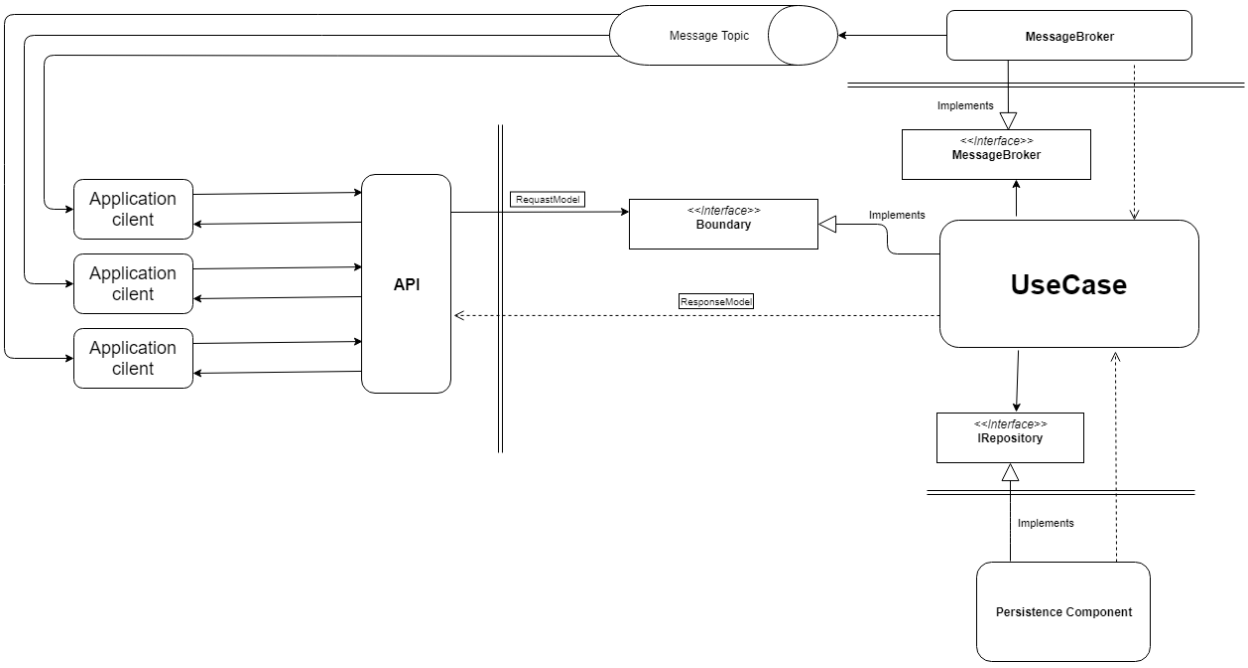


Slojevi u ovoj arhitekturi su Gateway, Boundary, UseCases i entities.

- **UseCase** - najbitniji sloj arhitekture jer se u njemu nalazi sva biznis logika specificna za aplikaciju. On se sastoji od vise useCase Objekta a svaki od njih predstavlja jednu atomicnu akciju koju korisnik moze da zatrazi od sistema
- Entity - sloj u kome je implementirana sva logika koja nije vezana samo za ovu aplikaciju vec se moze koristiti i u drugim aplikacijama kao gotovi moduli. Ovi objekti obicno vrse neka genericna racunavanja ili generalne zadatke ceste u ovakvim vrstama aplikacija.
- Boundary - sloj koji predstavlja granicu izmedju Gateway i UseCase sloja. Obezbedjuje da ta dva sloja nisu cvrsto Povezana.
- Gateway - zadatak ovog sloja je da prikuplja zahteve od spoljasnjeg sveta, u najcescem Slucaju je to web, i da ih prosledi dalje UseCase sloju preko boundary Sloja

Nasa implementacija arhitekture

Nasu implementaciju Clean Architecture dizajn obrasca mozete videti na slici ispod



Application Client objekti sa slike predstavljaju vise instanci naseg web Clienta. Posto je razvijeni sitem web aplikacija koja treba da opsluzi vise klijenata istovremeno Gateway Sloj se implementira kao WEB API. Ima zadatak da prihvata zahteve korisnika i prepakuje parametre zahteva u odgovarajuce RequstModele i poziva odgovarajuci useCase kroz interface koji je definisan u Boundary Sloju.

Entity objekti u nasoj implementaciji, izmedju ostalih, cine Repository i MessageBrokeri.

Zadaci ovih entiteta su :

- Message Broker se koristi da obavesti korisnike o tome da je zajednicka tabla bila promenjena i i salje odgovarajucu promenu u MessageTopic. Svaka tabla ima svoj poseban MessageTopic.
- Repositorijumi su entiteti koje vrse osnovne CRUD operacije nad nekom bazom podataka. Obezbedjuju persistenciju podataka u sistemu.

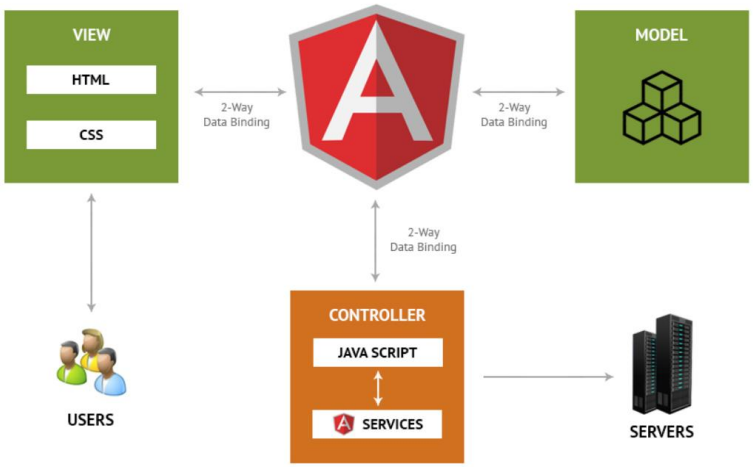
Izbor Radnih Okvira

Za izradu sistema smo izabrali sledece Radne okvire koje ce biti koriscenje u datim komponentama:

- .NET Core 3.0
  - API
- .NET standard
  - Boundary
  - UseCase
  - Repository
- << Neki MQ >>
  - MessageBroker
- Angular
  - Client Application

Arhitektura Klijentske aplikacije

Angular implementira MVC obrazac koriscenjem TypeScript-a i HTML-a.
View je definisan u HTML-u, dok su model i kontroler implementirani u TypeScript-u. Postoje razliciti nacini kako se ove komponente povezuju zajedno u Angular-u.



Nedostatak MVC-a je da ako promenimo prikaz, on se ne ažurira u modelu. Ovaj problem je rešen u Angular2 prelaskom na MVVM arhitekturu.

MVVM Arhitektura

MVVM ukljucuje 3 komponente:

- Model
- View
- View Model

Controller je zamenjen sa View Modelom u MMVM dizajnu. View Model nije ništa drugo nego TypeScript funkcija koja je opet poput kontrolera i odgovorna je za održavanje odnosa između pogleda i modela, ali razlika je u tome što se, ako ažuriramo bilo što u pogledu, ažurira u modelu, menjamo bilo šta u modelu, prikazuje se u view, što nazivamo 2-way binding.

View je HTML i na ovaj način podele funkcionalnosti view i view model mogu međusobno komunicirati. Stoga, kad god dođe do promene pogleda, view model zna za to. Tada view model ažurira model.