Autorzy: Adrian Chrobot, Mateusz Wejman, Piotr Aksamit

Zadanie: Stworzenie forum do komunikacja. Tworzenie postów i komentarzy i reagowanie na nie. Zakładanie grup.

Technologie: NodeJS, ReactJS, PostgreSQL

Architektura i technologie:

System jest zbudowany z 3 warstw technicznych:

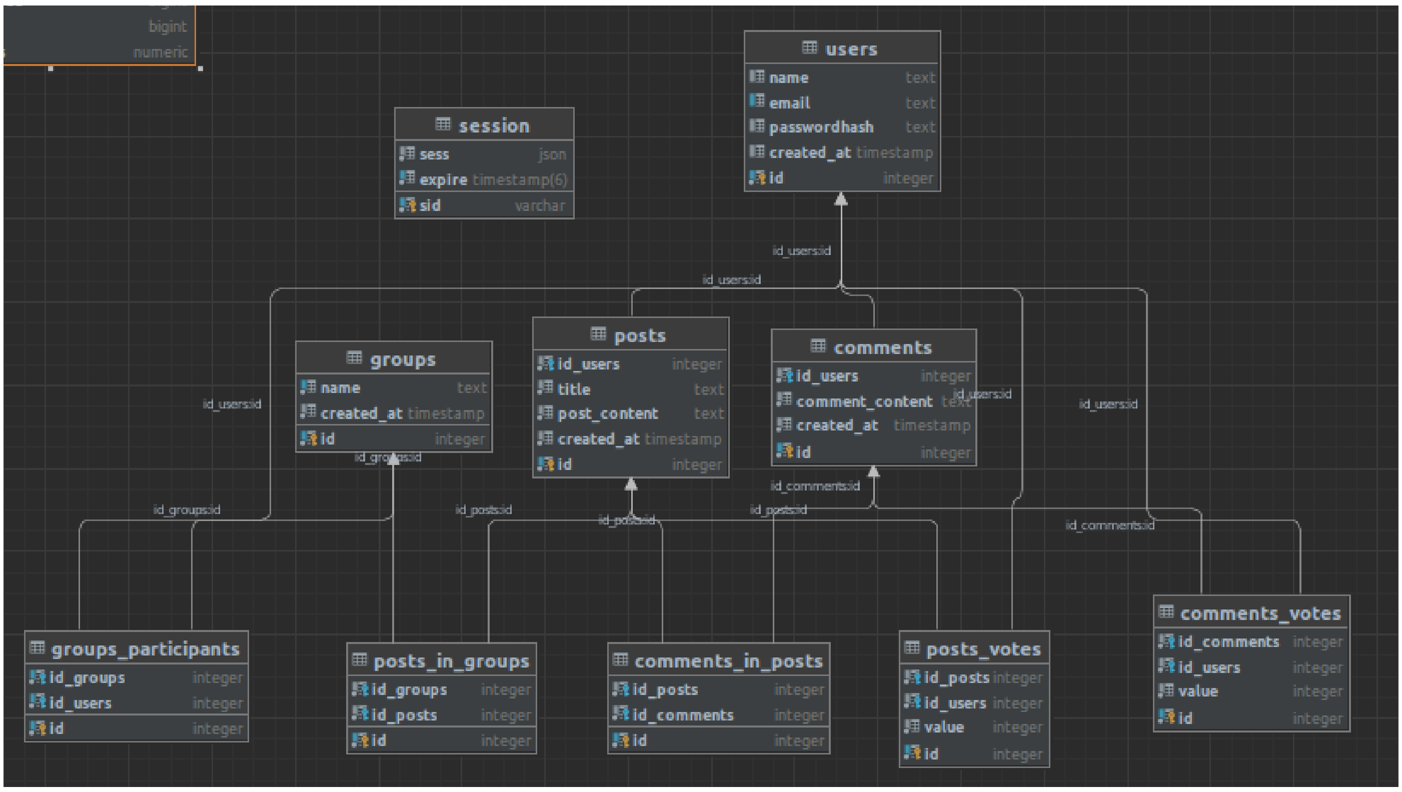
1. Warstwa persystencji – zawiera dane i stanowi ją instancją bazy danych PostgreSQL
2. Warstwa usługi/logiki biznesowej – Server w NodeJS
3. Warstwa prezentacji – ReactJS

Uruchamianie:

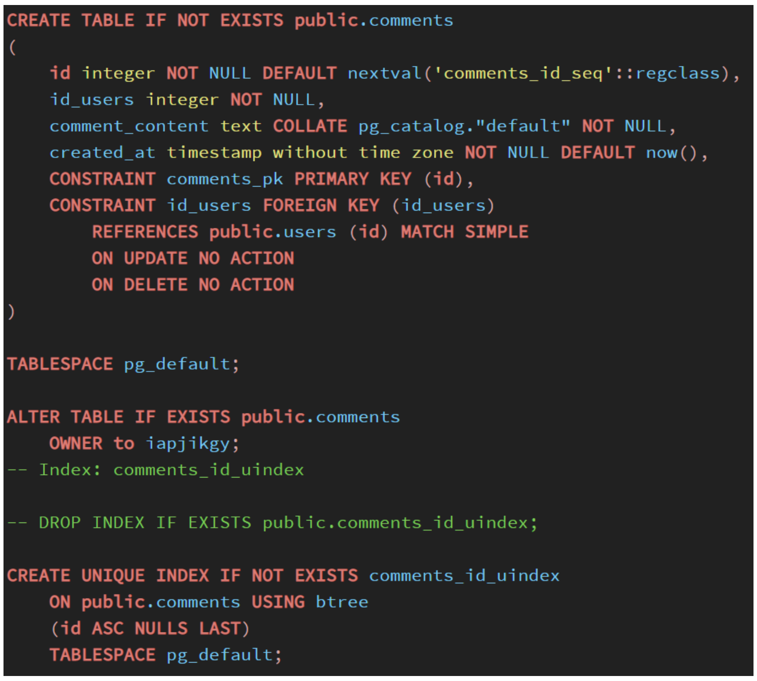
1. W obu katalogach reddit i reddit-front należy uruchomić: yarn install
2. W katalogu reddit należy uruchomić komendę: docker-compose up
3. W katalogu reddit-front należy uruchomić komendę: yarn start

Warstawa persystencji:

Schemat bazy

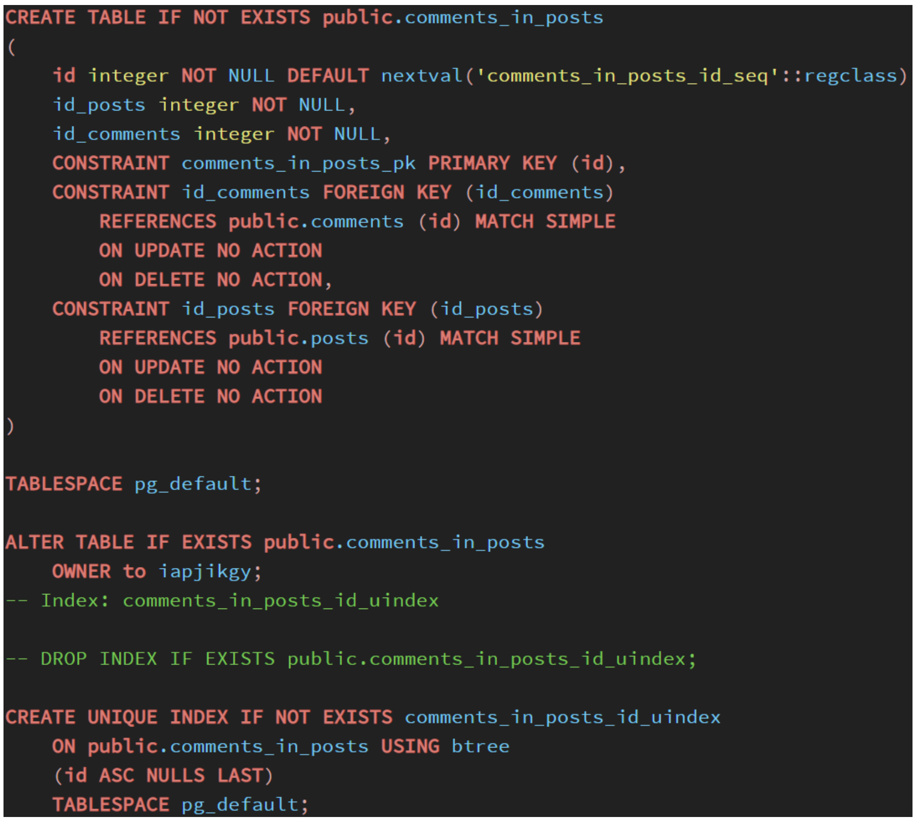


a. Tabela comments



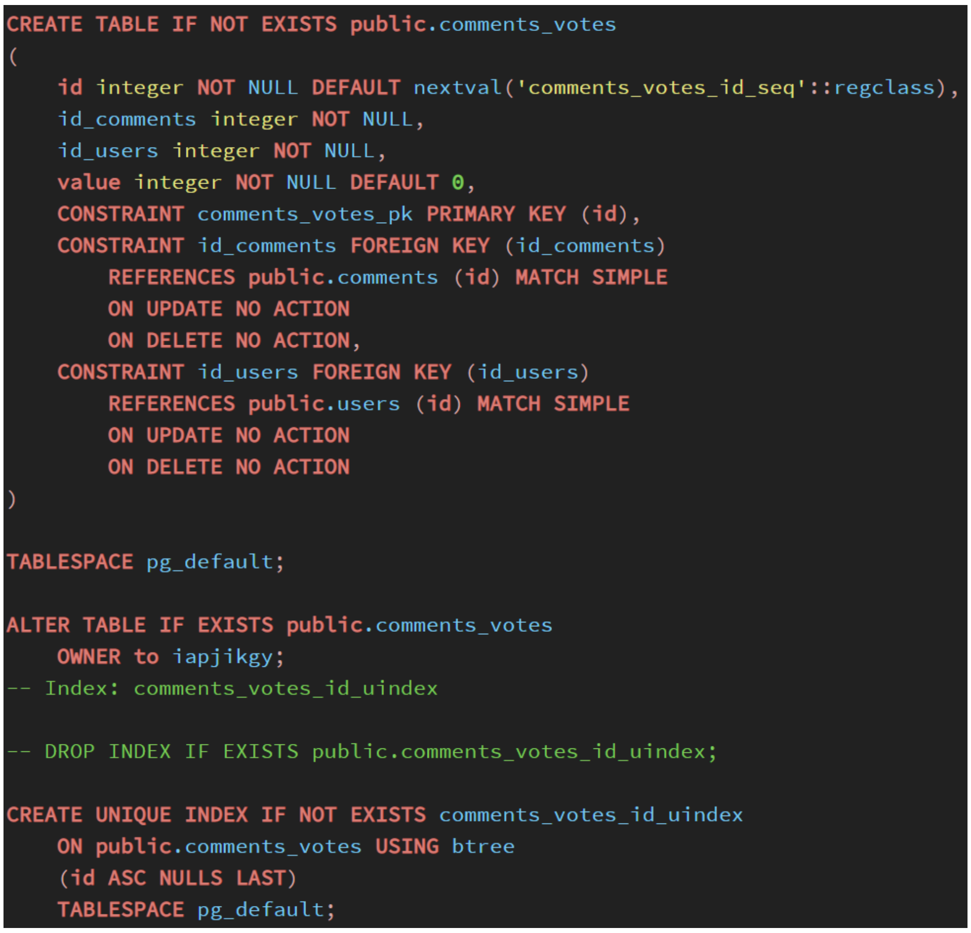
```sql
CREATE TABLE IF NOT EXISTS public.comments
(
    id integer NOT NULL DEFAULT nextval('comments_id_seq'::regclass),
    id_users integer NOT NULL,
    comment_content text COLLATE pg_catalog."default" NOT NULL,
    created_at timestamp without time zone NOT NULL DEFAULT now(),
    CONSTRAINT comments_pk PRIMARY KEY (id),
    CONSTRAINT id_users FOREIGN KEY (id_users)
        REFERENCES public.users (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.comments
    OWNER to iapjikgy;
-- Index: comments_id_uindex

-- DROP INDEX IF EXISTS public.comments_id_uindex;

CREATE UNIQUE INDEX IF NOT EXISTS comments_id_uindex
    ON public.comments USING btree
    (id ASC NULLS LAST)
    TABLESPACE pg_default;
```

b. Tabela comments_in_posts



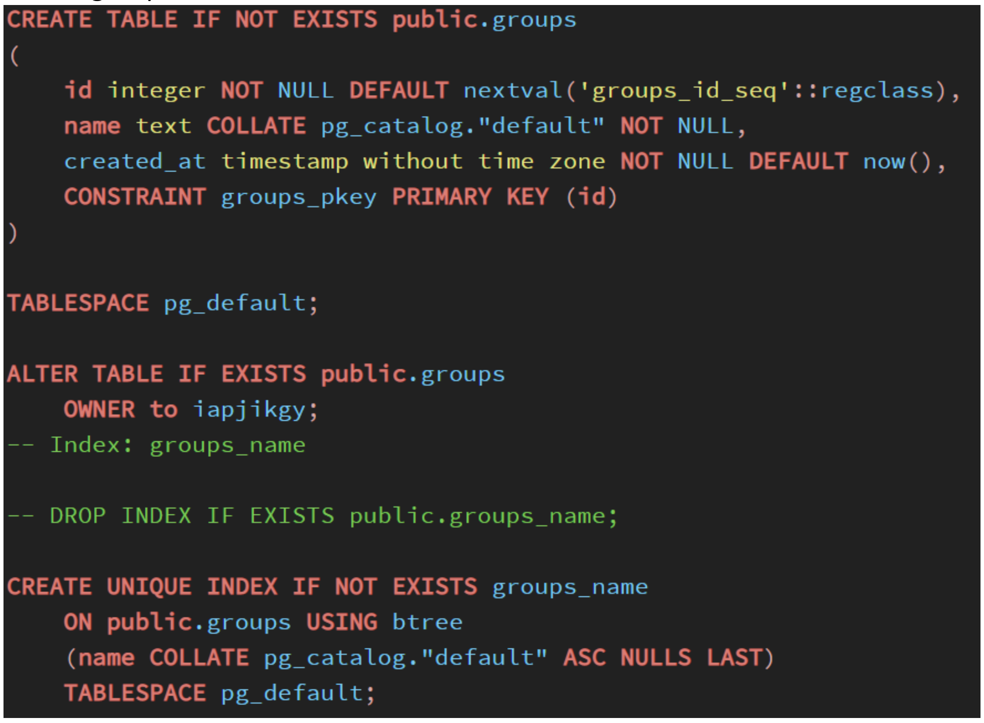
```sql
CREATE TABLE IF NOT EXISTS public.comments_in_posts
(
    id integer NOT NULL DEFAULT nextval('comments_in_posts_id_seq'::regclass)
    id_posts integer NOT NULL,
    id_comments integer NOT NULL,
    CONSTRAINT comments_in_posts_pk PRIMARY KEY (id),
    CONSTRAINT id_comments FOREIGN KEY (id_comments)
        REFERENCES public.comments (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT id_posts FOREIGN KEY (id_posts)
        REFERENCES public.posts (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.comments_in_posts
    OWNER to iapjikgy;
-- Index: comments_in_posts_id_uindex

-- DROP INDEX IF EXISTS public.comments_in_posts_id_uindex;

CREATE UNIQUE INDEX IF NOT EXISTS comments_in_posts_id_uindex
    ON public.comments_in_posts USING btree
    (id ASC NULLS LAST)
    TABLESPACE pg_default;
```

c. Tabela comments_votes



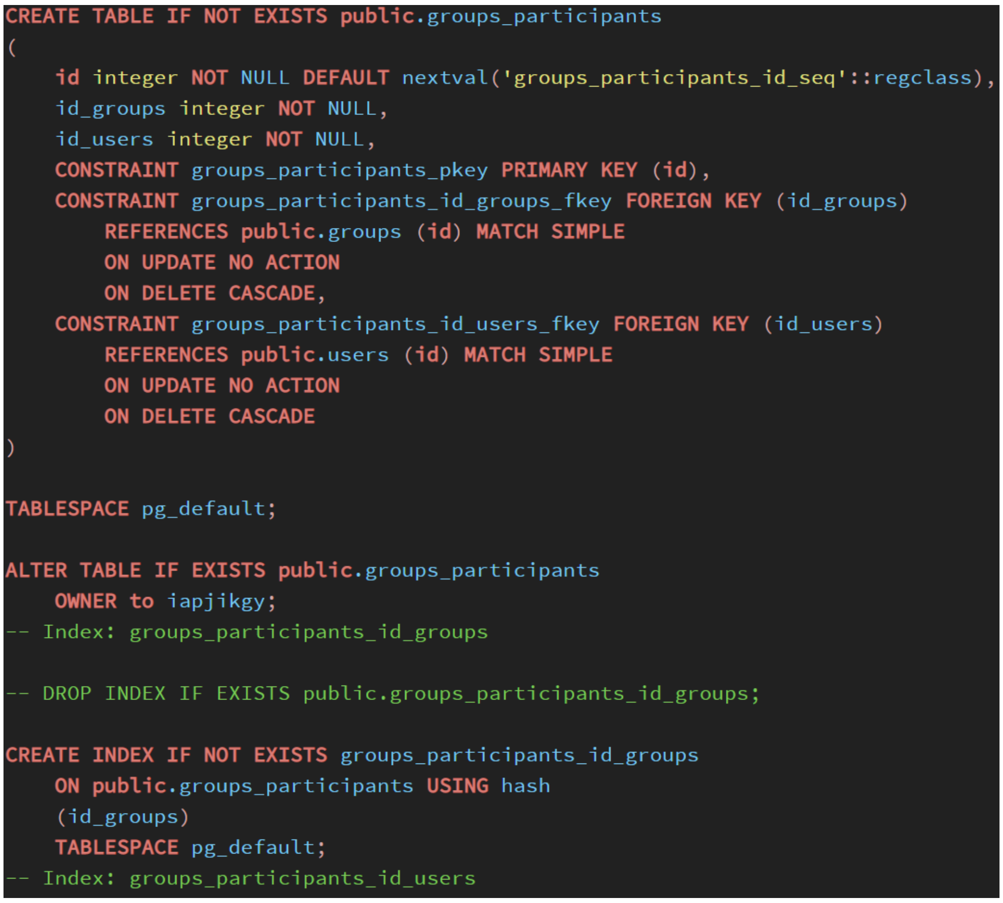
```sql
CREATE TABLE IF NOT EXISTS public.comments_votes
(
    id integer NOT NULL DEFAULT nextval('comments_votes_id_seq'::regclass),
    id_comments integer NOT NULL,
    id_users integer NOT NULL,
    value integer NOT NULL DEFAULT 0,
    CONSTRAINT comments_votes_pk PRIMARY KEY (id),
    CONSTRAINT id_comments FOREIGN KEY (id_comments)
        REFERENCES public.comments (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT id_users FOREIGN KEY (id_users)
        REFERENCES public.users (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.comments_votes
    OWNER to iapjikgy;
-- Index: comments_votes_id_uindex

-- DROP INDEX IF EXISTS public.comments_votes_id_uindex;

CREATE UNIQUE INDEX IF NOT EXISTS comments_votes_id_uindex
    ON public.comments_votes USING btree
    (id ASC NULLS LAST)
    TABLESPACE pg_default;
```

d. Tabela groups



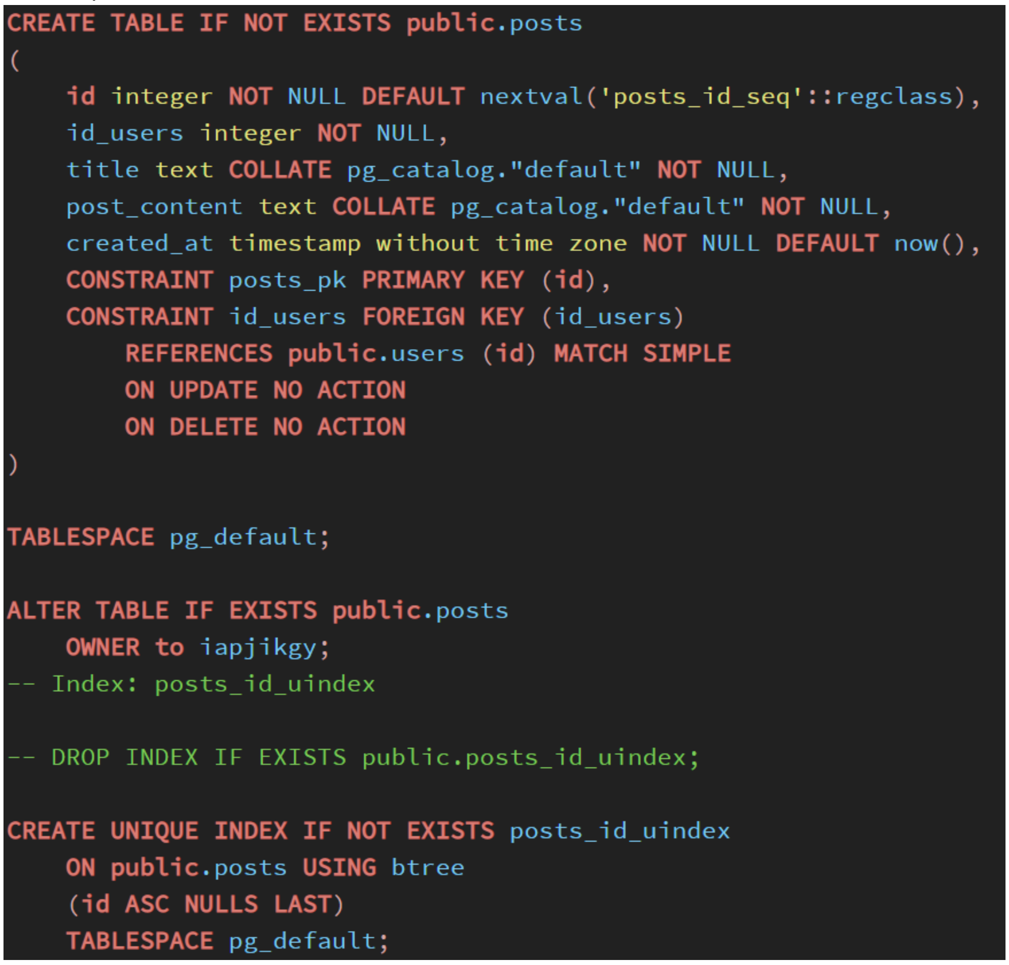
```sql
CREATE TABLE IF NOT EXISTS public.groups
(
    id integer NOT NULL DEFAULT nextval('groups_id_seq'::regclass),
    name text COLLATE pg_catalog."default" NOT NULL,
    created_at timestamp without time zone NOT NULL DEFAULT now(),
    CONSTRAINT groups_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.groups
    OWNER to iapjikgy;
-- Index: groups_name

-- DROP INDEX IF EXISTS public.groups_name;

CREATE UNIQUE INDEX IF NOT EXISTS groups_name
    ON public.groups USING btree
    (name COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

e. Tabela groups_participants



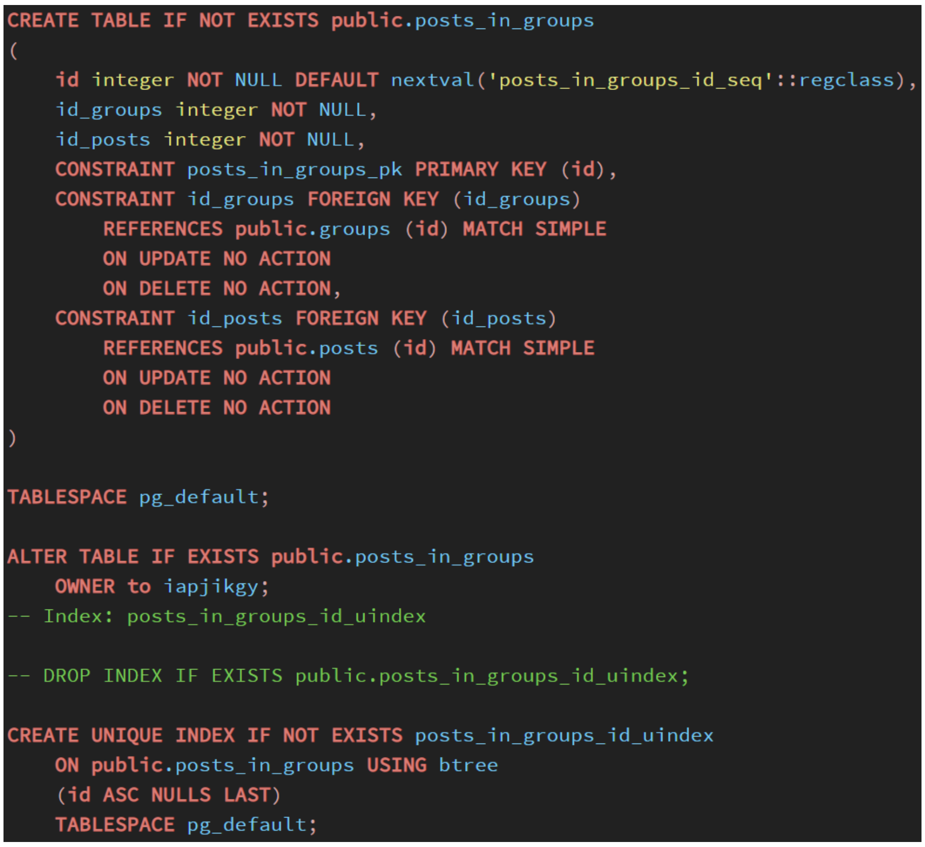
```sql
CREATE TABLE IF NOT EXISTS public.groups_participants
(
    id integer NOT NULL DEFAULT nextval('groups_participants_id_seq'::regclass),
    id_groups integer NOT NULL,
    id_users integer NOT NULL,
    CONSTRAINT groups_participants_pkey PRIMARY KEY (id),
    CONSTRAINT groups_participants_id_groups_fkey FOREIGN KEY (id_groups)
        REFERENCES public.groups (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT groups_participants_id_users_fkey FOREIGN KEY (id_users)
        REFERENCES public.users (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.groups_participants
    OWNER to iapjikgy;
-- Index: groups_participants_id_groups

-- DROP INDEX IF EXISTS public.groups_participants_id_groups;

CREATE INDEX IF NOT EXISTS groups_participants_id_groups
    ON public.groups_participants USING hash
    (id_groups)
    TABLESPACE pg_default;
-- Index: groups_participants_id_users
```

f. Tabela posts



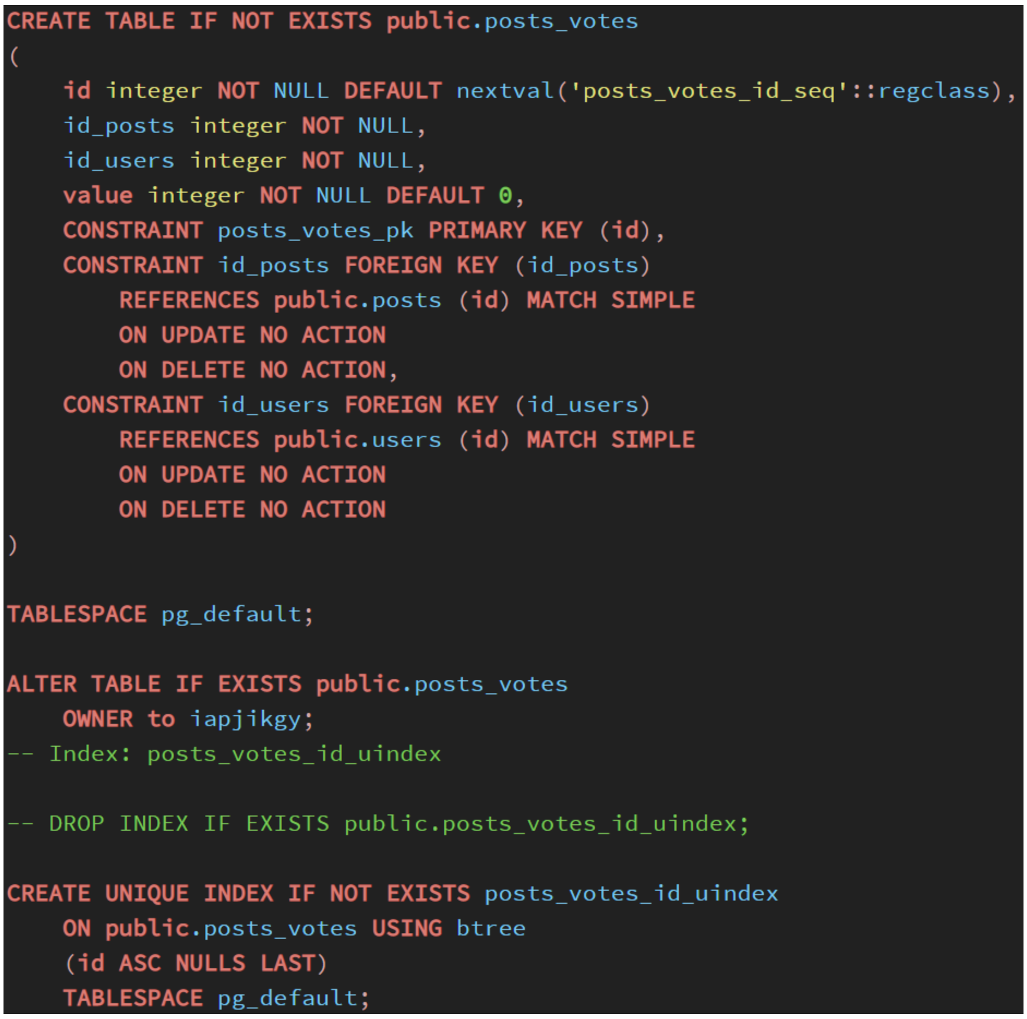
```sql
CREATE TABLE IF NOT EXISTS public.posts
(
    id integer NOT NULL DEFAULT nextval('posts_id_seq'::regclass),
    id_users integer NOT NULL,
    title text COLLATE pg_catalog."default" NOT NULL,
    post_content text COLLATE pg_catalog."default" NOT NULL,
    created_at timestamp without time zone NOT NULL DEFAULT now(),
    CONSTRAINT posts_pk PRIMARY KEY (id),
    CONSTRAINT id_users FOREIGN KEY (id_users)
        REFERENCES public.users (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.posts
    OWNER to iapjikgy;
-- Index: posts_id_uindex

-- DROP INDEX IF EXISTS public.posts_id_uindex;

CREATE UNIQUE INDEX IF NOT EXISTS posts_id_uindex
    ON public.posts USING btree
    (id ASC NULLS LAST)
    TABLESPACE pg_default;
```

g. Table posts_in_groups



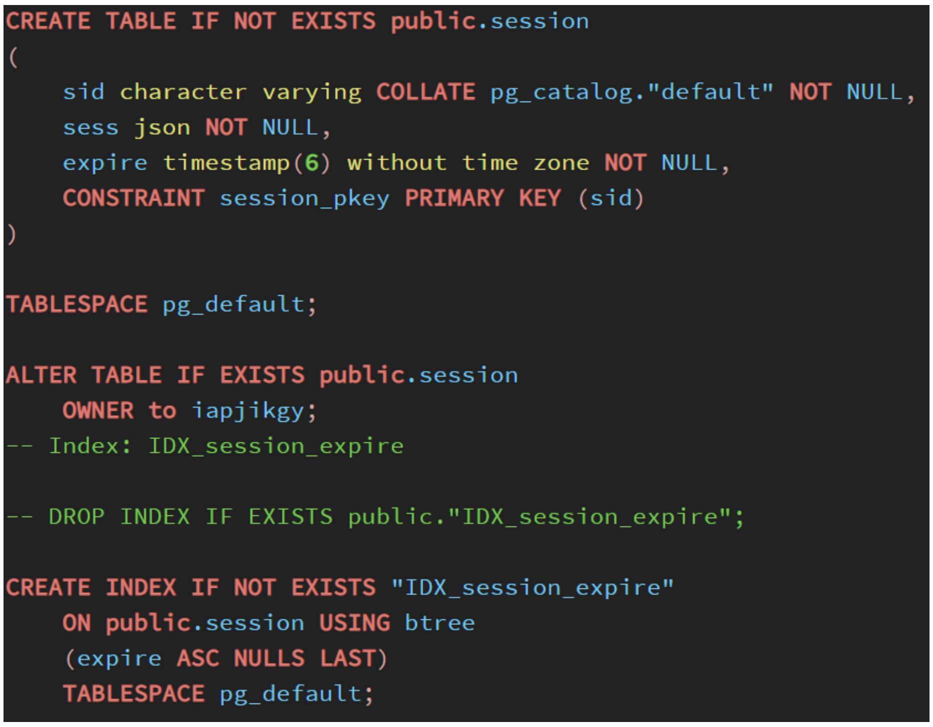
```
CREATE TABLE IF NOT EXISTS public.posts_in_groups
(
    id integer NOT NULL DEFAULT nextval('posts_in_groups_id_seq'::regclass),
    id_groups integer NOT NULL,
    id_posts integer NOT NULL,
    CONSTRAINT posts_in_groups_pk PRIMARY KEY (id),
    CONSTRAINT id_groups FOREIGN KEY (id_groups)
        REFERENCES public.groups (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT id_posts FOREIGN KEY (id_posts)
        REFERENCES public.posts (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.posts_in_groups
    OWNER to iapjikgy;
-- Index: posts_in_groups_id_uindex

-- DROP INDEX IF EXISTS public.posts_in_groups_id_uindex;

CREATE UNIQUE INDEX IF NOT EXISTS posts_in_groups_id_uindex
    ON public.posts_in_groups USING btree
    (id ASC NULLS LAST)
    TABLESPACE pg_default;
```

h. Tabela posts_votes



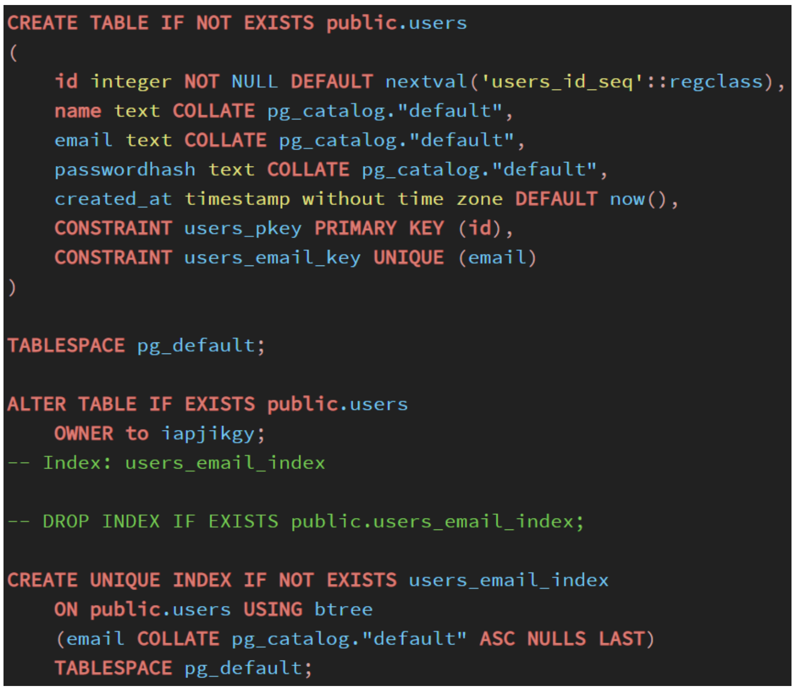
```
CREATE TABLE IF NOT EXISTS public.posts_votes
(
    id integer NOT NULL DEFAULT nextval('posts_votes_id_seq'::regclass),
    id_posts integer NOT NULL,
    id_users integer NOT NULL,
    value integer NOT NULL DEFAULT 0,
    CONSTRAINT posts_votes_pk PRIMARY KEY (id),
    CONSTRAINT id_posts FOREIGN KEY (id_posts)
        REFERENCES public.posts (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT id_users FOREIGN KEY (id_users)
        REFERENCES public.users (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.posts_votes
    OWNER to iapjikgy;
-- Index: posts_votes_id_uindex

-- DROP INDEX IF EXISTS public.posts_votes_id_uindex;

CREATE UNIQUE INDEX IF NOT EXISTS posts_votes_id_uindex
    ON public.posts_votes USING btree
    (id ASC NULLS LAST)
    TABLESPACE pg_default;
```

i. Tabela session



```
CREATE TABLE IF NOT EXISTS public.session
(
    sid character varying COLLATE pg_catalog."default" NOT NULL,
    sess json NOT NULL,
    expire timestamp(6) without time zone NOT NULL,
    CONSTRAINT session_pkey PRIMARY KEY (sid)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.session
    OWNER to iapjikgy;
-- Index: IDX_session_expire

-- DROP INDEX IF EXISTS public."IDX_session_expire";

CREATE INDEX IF NOT EXISTS "IDX_session_expire"
    ON public.session USING btree
    (expire ASC NULLS LAST)
    TABLESPACE pg_default;
```

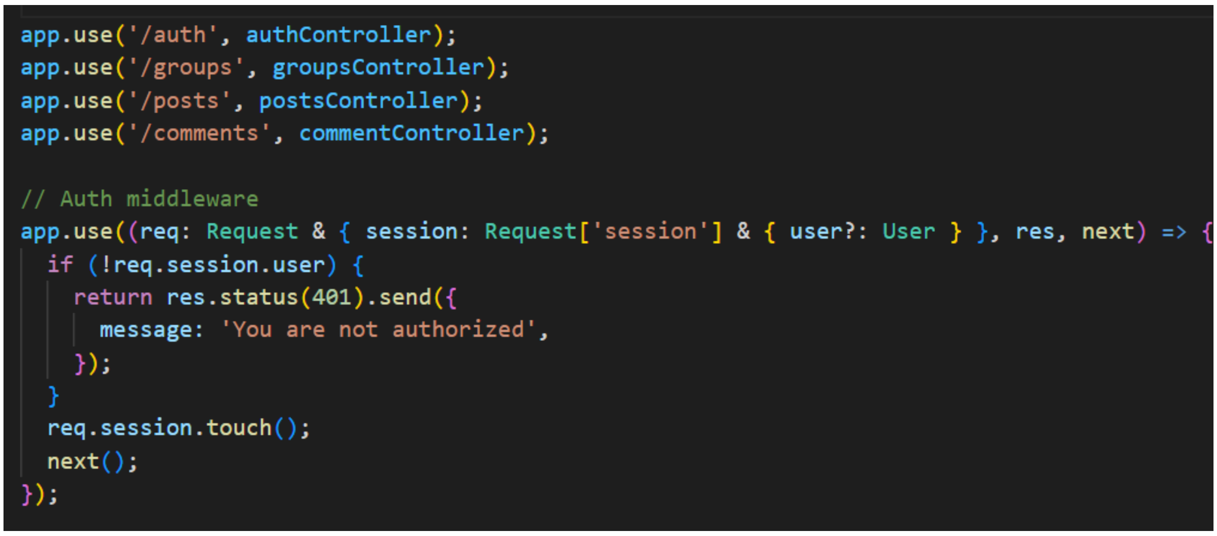
j. Table users



```
CREATE TABLE IF NOT EXISTS public.users
(
    id integer NOT NULL DEFAULT nextval('users_id_seq'::regclass),
    name text COLLATE pg_catalog."default",
    email text COLLATE pg_catalog."default",
    passwordhash text COLLATE pg_catalog."default",
    created_at timestamp without time zone DEFAULT now(),
    CONSTRAINT users_pkey PRIMARY KEY (id),
    CONSTRAINT users_email_key UNIQUE (email)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.users
    OWNER to iapjikgy;
-- Index: users_email_index

-- DROP INDEX IF EXISTS public.users_email_index;

CREATE UNIQUE INDEX IF NOT EXISTS users_email_index
    ON public.users USING btree
    (email COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

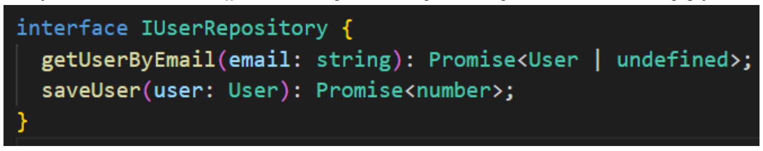
Warstwa usługi/logiki biznesowej

Wykorzystujemy 4 rodzaje kontrolerów



```
app.use('/auth', authController);
app.use('/groups', groupsController);
app.use('/posts', postsController);
app.use('/comments', commentController);

// Auth middleware
app.use((req: Request & { session: Request['session'] & { user?: User } }, res, next) => {
  if (!req.session.user) {
    return res.status(401).send({
      message: 'You are not authorized',
    });
  }
  req.session.touch();
  next();
});
```

a. AuthController + Auth middleware
   req.session.touch() informuje o sesja dalej trwa i trzeba ją przesłużyć



```
interface IUserRepository {
  getUserByEmail(email: string): Promise<User | undefined>;
  saveUser(user: User): Promise<number>;
}
```

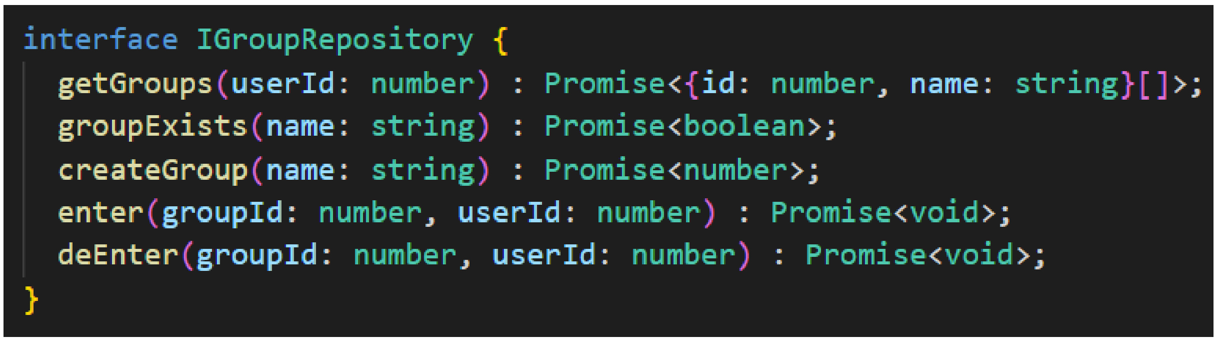
Dostępne endpointy:

```
authController.post('/login',
      służy do logowania się, w body przekazywanie jest email i hasło.
      Najpierw server sprawdza czy parametry są przekazane,
      Potem czy podany email istnieje w tabeli ‘Users’, a następnie porównuje
      Hashe haseł. W przypadku sukcesu podpina użytkownika pod sesje.


authController.post('/logout',
      niszczy aktualną sesję
authController.post('/register',
      służy do rejestrowania się, w body przekazywane jest email i hasło.
      Server sprawdza czy podany email nie jest już zajęty, jeżeli nie tworzy   nowego
użytkownika w bazie
authController.get('/me',
      zwraca aktualną sesje
```

Middleware

```
export default function auth(req: express.Request & { session: any }, res:
express.Response, next: express.NextFunction) {
  if (req.session && req.session.user) {
    return next();
  }
  return res.status(401).send('Not authorized');
}
```

b. GroupsController – użytkownik musi być zalogowany

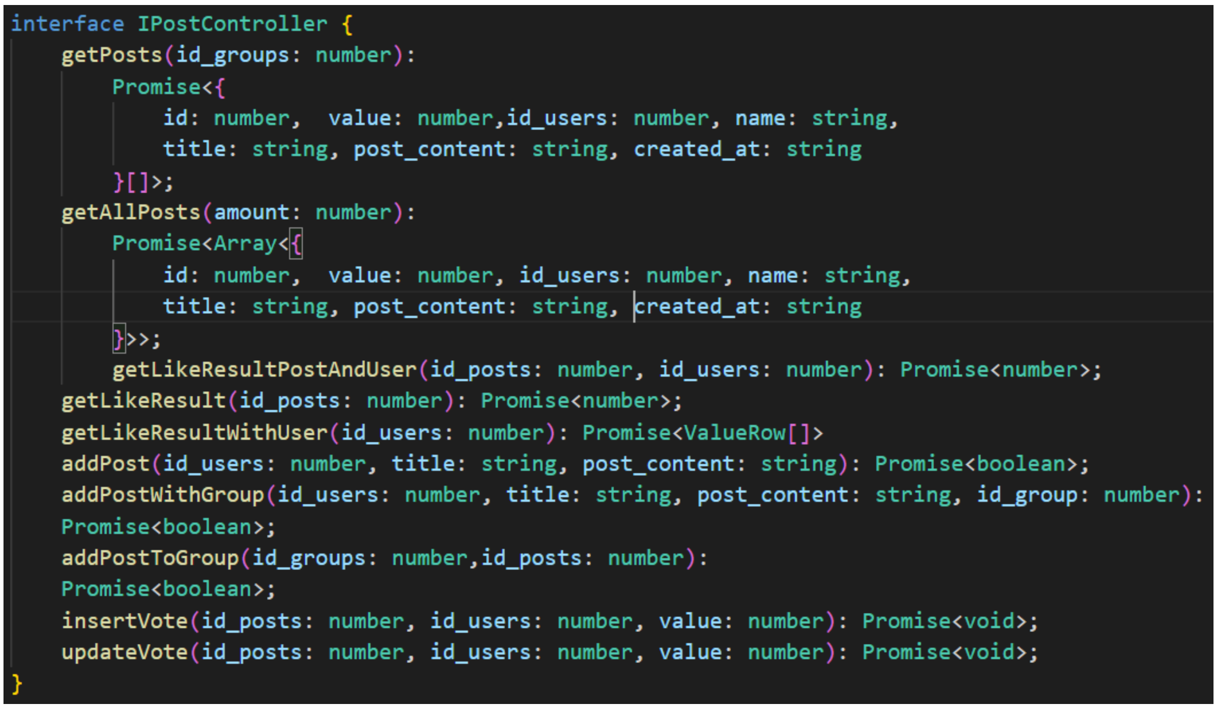


```
interface IGroupRepository {
  getGroups(userId: number) : Promise<{id: number, name: string}[]>;
  groupExists(name: string) : Promise<boolean>;
  createGroup(name: string) : Promise<number>;
  enter(groupId: number, userId: number) : Promise<void>;
  deEnter(groupId: number, userId: number) : Promise<void>;
}
```

Dostępne endpointy:

```
groupController.get('',
      pobiera wszystkie dostępne grupy dla danego uzytkownika
groupController.post('/enter',
      dodaje użytkownika z sesji do grupy, której id jest przekazany w body zapytania
groupController.post('/deenter',
      usuwa użytkownika z sesji z grupy, której id jest przekazane w body zapytania
groupController.post('/register',
      tworzy nową grupę o nazwie przekazanej w body zapytania
```

c. PostsController

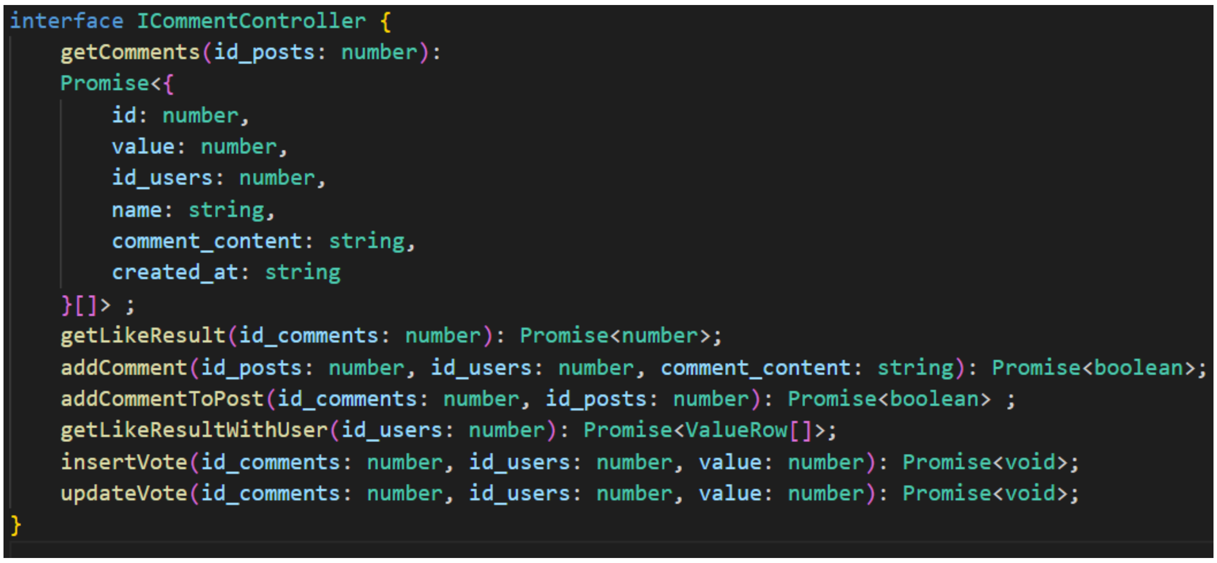


```
interface IPostController {
    getPosts(id_groups: number):
        Promise<{
            id: number,  value: number,id_users: number, name: string,
            title: string, post_content: string, created_at: string
        }[]>;
    getAllPosts(amount: number):
        Promise<Array<{
            id: number,  value: number, id_users: number, name: string,
            title: string, post_content: string, created_at: string
        }>>;
        getLikeResultPostAndUser(id_posts: number, id_users: number): Promise<number>;
    getLikeResult(id_posts: number): Promise<number>;
    getLikeResultWithUser(id_users: number): Promise<ValueRow[]>
    addPost(id_users: number, title: string, post_content: string): Promise<boolean>;
    addPostWithGroup(id_users: number, title: string, post_content: string, id_group: number):
    Promise<boolean>;
    addPostToGroup(id_groups: number,id_posts: number):
    Promise<boolean>;
    insertVote(id_posts: number, id_users: number, value: number): Promise<void>;
    updateVote(id_posts: number, id_users: number, value: number): Promise<void>;
}
```

Dostępne endpointy:

```
postController.get('',

      pobierane posty dla

postController.get('/groupPost/:id_group',

      pobierana posty dla danej grupy

postController.get('/likeResult/:id_posts',

      pobierane wyniki polubieni dla danego posta

postController.get('/likeResultUser',
postController.post('/insertVote',
      tworzy upvota lub downvota dla danej osoby
postController.post('/updateVote',
      modyfikuje upvota lub downvota dla danej osoby
postController.get('/likeResultWithUserAndPost/:id_posts/',

postController.post('/addPost',
      tworzy posta przekazane w body zapytania
postController.post('/addPostToGroup',
      dodaje posta lub przenosi go do danej grupy przekazanej w body zapytania
```

d. CommentController

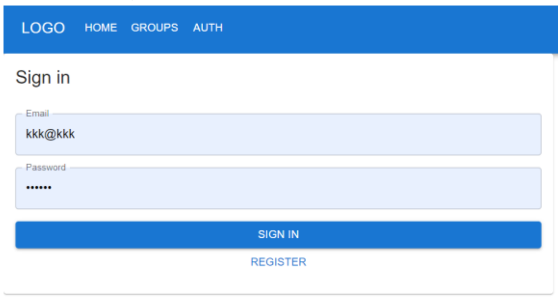


```
interface ICommentController {
    getComments(id_posts: number):
    Promise<{
        id: number,
        value: number,
        id_users: number,
        name: string,
        comment_content: string,
        created_at: string
    }[]> ;
    getLikeResult(id_comments: number): Promise<number>;
    addComment(id_posts: number, id_users: number, comment_content: string): Promise<boolean>;
    addCommentToPost(id_comments: number, id_posts: number): Promise<boolean> ;
    getLikeResultWithUser(id_users: number): Promise<ValueRow[]>;
    insertVote(id_comments: number, id_users: number, value: number): Promise<void>;
    updateVote(id_comments: number, id_users: number, value: number): Promise<void>;
}
```

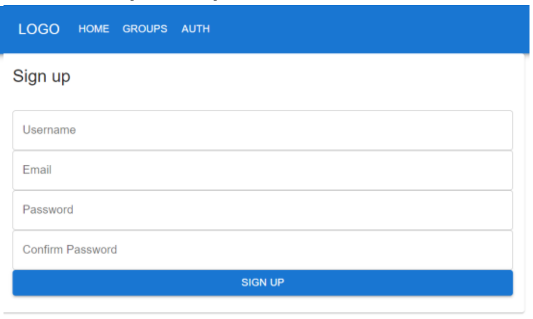
```
commentController.get('/commentsForPost/:id_posts',
      pobiera komentarze dla danego posta
commentController.post('/insertVote',
      dodaje upvote lub downvota do komentarza, id komentarza jak i czy to upvota czy
downvote przekazana w body zapytania
commentController.post('/updateVote'
      updatuje upvote lub downvota do komentarza, id komentarza jak i czy to upvota czy
downvote przekazana w body zapytania
commentController.get('/likeResultUser', k
commentController.post('/addComment',
      tworzy komentarz przekazany w body zapytania
commentController.post('/addCommentToPost',
      dodaje komentarz lub przenosi go do danego posta przekazanego w body zapytania
```

4. Warstwa prezentacji

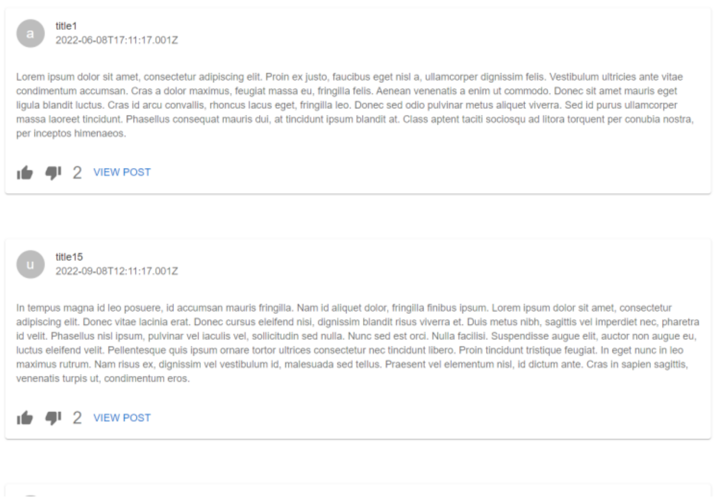
Logowanie



Rejestracja



Widok postów


a
title1
2022-06-08T17:11:17.001Z

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin ex justo, faucibus eget nisl a, ullamcorper dignissim felis. Vestibulum ultricies ante vitae condimentum accumsan. Cras a dolor maximus, feugiat massa eu, fringilla felis. Aenean venenatis a enim ut commodo. Donec sit amet mauris eget ligula blandit luctus. Cras id arcu convallis, rhoncus lacus eget, fringilla leo. Donec sed odio pulvinar metus aliquet viverra. Sed id purus ullamcorper massa laoreet tincidunt. Phasellus consequat mauris dui, at tincidunt ipsum blandit at. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.

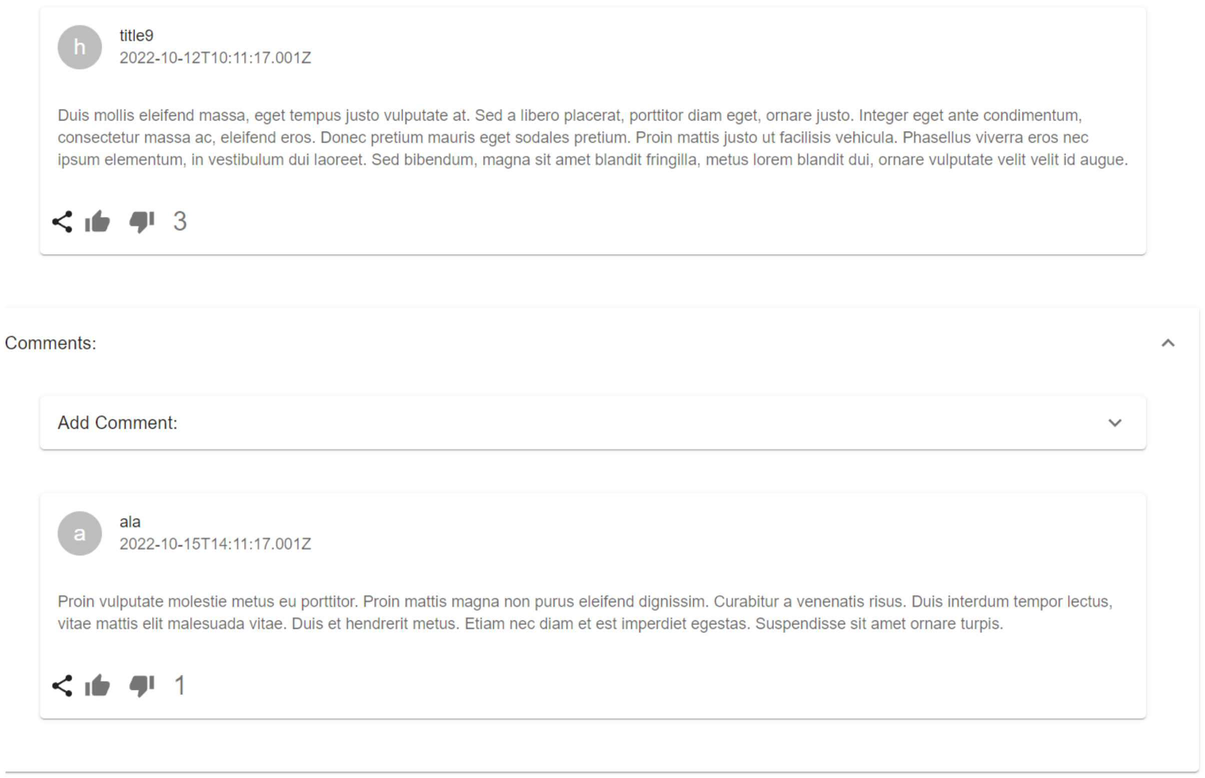
2 VIEW POST

u
title15
2022-09-08T12:11:17.001Z

In tempus magna id leo posuere, id accumsan mauris fringilla. Nam id aliquet dolor, fringilla finibus ipsum. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec vitae lacinia erat. Donec cursus eleifend nisi, dignissim blandit risus viverra et. Duis metus nibh, sagittis vel imperdiet nec, pharetra id velit. Phasellus nisl ipsum, pulvinar vel iaculis vel, sollicitudin sed nulla. Nunc sed est orci. Nulla facilisi. Suspendisse augue elit, auctor non augue eu, luctus eleifend velit. Pellentesque quis ipsum ornare tortor ultrices consectetur nec tincidunt libero. Proin tincidunt tristique feugiat. In eget nunc in leo maximus rutrum. Nam risus ex, dignissim vel vestibulum id, malesuada sed tellus. Praesent vel elementum nisl, id dictum ante. Cras in sapien sagittis, venenatis turpis ut, condimentum eros.

2 VIEW POST

Specyfikacja danego posta


h
title9
2022-10-12T10:11:17.001Z

Duis mollis eleifend massa, eget tempus justo vulputate at. Sed a libero placerat, porttitor diam eget, ornare justo. Integer eget ante condimentum, consectetur massa ac, eleifend eros. Donec pretium mauris eget sodales pretium. Proin mattis justo ut facilisis vehicula. Phasellus viverra eros nec ipsum elementum, in vestibulum dui laoreet. Sed bibendum, magna sit amet blandit fringilla, metus lorem blandit dui, ornare vulputate velit velit id augue.

3

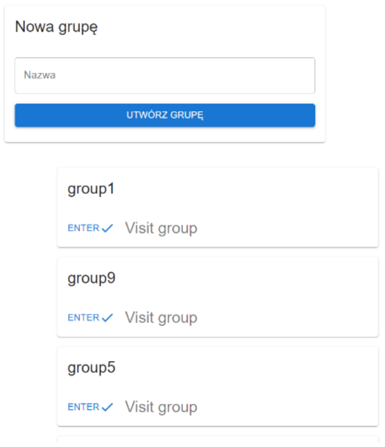
Comments:

Add Comment:

a
ala
2022-10-15T14:11:17.001Z

Proin vulputate molestie metus eu porttitor. Proin mattis magna non purus eleifend dignissim. Curabitur a venenatis risus. Duis interdum tempor lectus, vitae mattis elit malesuada vitae. Duis et hendrerit metus. Etiam nec diam et est imperdiet egestas. Suspendisse sit amet ornare turpis.

1


Nowa grupę

Nazwa

UTWÓRZ GRUPĘ

group1
ENTER ✓ Visit group

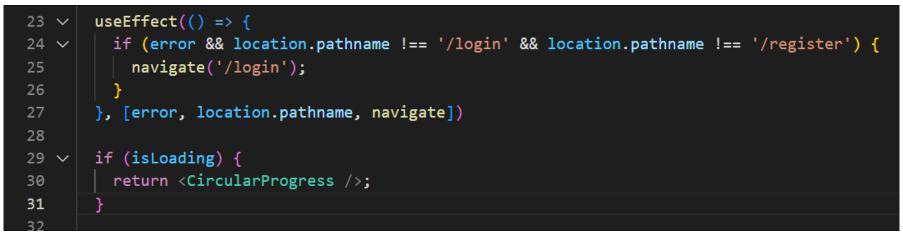
group9
ENTER ✓ Visit group

group5
ENTER ✓ Visit group

Można założyć nową grupę.
Do każdej grupy można się zapisać/wypisać.

Ciekawe rozwiązania:

1. w trakcie pobierania Zastosowanie kółka ładowanie danych np. logowanie, rejestracja



```
useEffect(() => {
  if (error && location.pathname !== '/login' && location.pathname !== '/register') {
    navigate('/login');
  }
}, [error, location.pathname, navigate])

if (isLoading) {
  return <CircularProgress />;
}
```