A6: Indexes, triggers, user functions, transactions and population

UConnect is a project aiming to create an academic social network allowing efficient communication and connectivity between all parts of the university context. Its implementation creates a unique platform in which every activity related to studying is included.

This artifact contains the database's physical schema, the estimation of the database's workload, the performance indexes we chose and its description. There is also the definition of triggers and transactions to ensure the integrity of the data giving any circumstances. The SQL code to create the database, indexes, triggers, transactions and population data is also included.

1. Database Workload

1.1. Tuple Estimation

Relation reference	Relation Name	Order of magnitude	Estimated growth
R01	user	thousands	dozens per day
R02	admin	units	no growth
R03	regular_user	thousands	dozens per day
R04	student	thousands	dozens per day
R05	teacher	hundreds	units per day
R06	organization	dozens	units per month
R07	event	hundreds	dozens per month
R08	group	hundreds	units per month
R09	post	tens of thousands	hundreds per day
R10	file	thousands	dozens per day
R11	image	thousands	dozens per day
R12	comment	hundreds of thousands	thousands per day
R13	chat	tens of thousands	dozens per day
R14	message	millions	thousands per day

Relation reference	Relation Name	Order of magnitude	Estimated growth
R15	user_in_chat	tens of thousands	dozens per day
R16	notification	millions	thousands per day
R17	notified_user	tens of millions	tens of thousands per day
R18	user_in_group	tens of thousands	dozens per day
R19	report	thousands	dozens per day
R20	friend	hundreds of thousands	dozens per day
R21	user_interested_in_event	tens of thousands	dozens per day

1.2. Frequent Queries

Query	SELECT01
Description	Get user's info
Frequency	thousands per day

```
SELECT "user_id", "name", "email"
    FROM "user"
    WHERE "email" = $email;
```

Query	SELECT02
Description	Get a user's posts ordered by date
Frequency	thousands per day

```
SELECT "title", "body", "date", "upvotes", "downvotes", TYPE, "event_id", "group_id"
    FROM "post"
    WHERE "author_id" = $userId
    ORDER BY "date" DESC;
```

Query	SELECT03
Description	Get a group's posts ordered by date
Frequency	tens of thousands per day

```
SELECT "author_id", "title", "body", "date", "upvotes", "downvotes", TYPE, "event_id",
    FROM "post"
    WHERE "group_id" = $groupId
    ORDER BY "date" DESC;
```

Query	SELECT04
Description	Get an event's posts ordered by date
Frequency	tens of thousands per day

```
SELECT "author_id", "title", "body", "date", "upvotes", "downvotes", TYPE, "event_id",
    FROM "post"
    WHERE "event_id" = $eventId
    ORDER BY "date" DESC;
```

Query	SELECT05
Description	Get a post's comments ordered by date
Frequency	hundreds of thousands per day

```
SELECT "comment_id", "user_id", "comment_to_id", "body", "date", "upvotes", "downvotes
    FROM "comment"
    WHERE "post_id" = $postId
    ORDER BY "date" DESC;
```

Query	SELECT06
Description	Select a user's notifications
Frequency	millions per day

```
SELECT "origin_user_id", "description", "link", "date"
    FROM "notification"
    INNER JOIN "notified_user" ON "notification"."notification_id" ="notified_user"."n
    WHERE "user_notified" = $userId;
```

Query	SELECT07
Description	Select an organization's events

Query	SELECT07
Frequency	dozens per day

```
SELECT "name", "location", "date", "information"
    FROM "event"
    WHERE "organization_id" = $orgId;
```

Query	SELECT08
Description	Select all the friends of a user
Frequency	thousands per day

```
SELECT "friend_id2"
    FROM "friend"
    WHERE (TYPE = 'accepted' AND "friend_id1" = $friendId);
```

Query	SELECT09
Description	Select all pending friend requests of a user
Frequency	hundreds per day

```
SELECT "friend_id2"
    FROM "friend"
    WHERE (TYPE = 'pending' and "friend_id1" = $friendId);
```

Query	SELECT10
Description	Get a chat's messages ordered by date
Frequency	millions per day

```
SELECT "sender_id", "body", "date"
    FROM "message"
    where "chat_id" = $chatId
    ORDER BY "date" DESC;
```

Query	SELECT11
Description	Select a post's file path
Frequency	thousands per day

```
SELECT "file_path"
    FROM "file"
    INNER JOIN "post" ON "post"."post_id" = "file"."post_id"
    WHERE "post"."post_id" = $postId;
```

Query	SELECT12
Description	Select information of the groups a user belongs
Frequency	thousands per day

```
Select "group"."group_id" , "name", "information", TYPE
    from "user_in_group" INNER JOIN "group"
    on "user_in_group"."group_id" = "group"."group_id"
    where "user_id" = $userId;
```

Query	SELECT13
Description	Select information of events a user is interested on
Frequency	thousands per day

```
Select "event"."event_id", "organization_id", "name", "location", "date", "information
    FROM "event" INNER JOIN "user_interested_in_event"
    on "event"."event_id" = "user_interested_in_event"."event_id"
    Where "user_interested_in_event"."user_id" = $userId;
```

Query	SELECT14
Description	Select posts for the feed (users, groups and events)
Frequency	millions per day

```
Select "posts"."post_id","author_id", "title", "body", "date", "upvotes", "downvotes",
       "posts".TYPE, "event_id", "posts"."group_id", COALESCE("count",0) as "comment_c
    Select "post_id","author_id", "title", "body", "date", "upvotes", "downvotes",
               "post".TYPE, "event_id", "post"."group_id"
         from "post" INNER JOIN "user_in_group"
                            on "post"."group_id" = "user_in_group"."group_id"
         where "user_in_group"."user_id" = $userId
    UNION
    Select "post_id","author_id", "title", "body", "date", "upvotes", "downvotes",
               "post".TYPE, "event_id", "post"."group_id"
         from "post" INNER JOIN "friend"
                            on "friend"."friend_id2" = "post"."author_id"
         WHERE "friend".TYPE = 'accepted' AND "friend"."friend_id1" = $friendId
    UNION
    Select "post_id","author_id", "title", "body", "date", "upvotes", "downvotes",
               "post".TYPE, "post"."event_id", "post"."group_id"
         from "post" INNER JOIN "user_interested_in_event"
                     on "post"."event_id" = "user_interested_in_event"."event_id"
         WHERE "user_interested_in_event"."user_id" = $userId
    ) as "posts"
    LEFT JOIN
    (Select "post_id",count("post_id") as "count"
                from "comment" GROUP BY "post_id") as "comments"
    on "posts"."post_id" = "comments"."post_id";
```

Query	SELECT15
Description	Select the student id giving the regular user id
Frequency	hundreds per day

```
Select "student_id" from "student" where "regular_user_id" = $regUserId;
```

Query	SELECT16
Description	Select the teacher id giving the regular user id
Frequency	dozens per day

```
Select "teacher_id" from "teacher" where "regular_user_id" = $regUserId;
```

Query	SELECT17
Description	Select the organization id giving the regular user id
Frequency	dozens per day

```
Select "organization_id" from "organization" where "regular_user_id" = $regUserId;
```

Query	SELECT18
Description	Select comments to a comment
Frequency	hundreds per day

```
SELECT "comment_id", "user_id", "body", "date", "upvotes", "downvotes"
    FROM "comment"
    WHERE ("post_id" = $postId AND  "comment_to_id" = $commentId)
    ORDER BY "date" ASC;
```

Query	SELECT19
Description	Query regular users and posts whose name or title or body has a given string
Frequency	millions per day

```
Select * from
        (Select "name" , "user"."user_id" , "regular_user"."regular_user_id"
                from "user"
        INNER JOIN "regular_user" on "regular_user"."user_id" = "user"."user_id"
        where "user"."name" LIKE '%$str%')
        as "t1"
        FULL OUTER JOIN
        (Select "post_id","author_id", "title", "body", "date",
                 "upvotes", "downvotes", "post".TYPE, "post"."event_id",
                 "post"."group_id" from
            "post" where "post"."body" LIKE '%$str%' or "post"."title" LIKE '%$str%'
        ) as "t2" on "t1"."name" = "body";
```

1.3. Frequent Updates

Query	UPDATE01
Description	Update post status to deleted
Frequency	dozens per day

```
UPDATE "post"
    SET TYPE = 'deleted'
    WHERE "post_id" = $postId;
```

Query	UPDATE02
Description	Update post status to blocked
Frequency	dozens per day

```
UPDATE "post"
    SET TYPE = 'blocked'
    WHERE "post_id" = $postId;
```

Query	UPDATE03

Query	UPDATE03
Description	Update user status to deleted
Frequency	dozens per month

```
UPDATE "user"
    SET TYPE = 'deleted'
    WHERE "user_id" = $userId;
```

Query	UPDATE05
Description	Update user status to blocked
Frequency	dozens per month

```
UPDATE "user"
    SET TYPE = 'blocked'
    WHERE "user_id" = $userId;
```

Query	UPDATE06
Description	Update group status to deleted
Frequency	units per month

```
UPDATE "group"
    SET TYPE = 'deleted'
    WHERE "group_id" = $groupId;
```

Query	UPDATE07
Description	Update group status to blocked
Frequency	units per month

```
UPDATE "group"
    SET TYPE = 'blocked'
    WHERE "group_id" = $groupId;
```

Query	UPDATE08
Description	Update organization status to approved
Frequency	units per month

```
UPDATE "organization"
    SET "approval" = TRUE
    WHERE "organization_id" = $orgId;
```

Query	UPDATE09
Description	Update the user notification to seen
Frequency	millions per day

```
UPDATE "notified_user"
    SET "seen" = TRUE
    WHERE ("notification_id" = $notId AND "user_notified" = $userId);
```

Query	UPDATE10
Description	Update a report approval to true
Frequency	units per day

```
UPDATE "report"
    SET "approval" = TRUE
    WHERE "report_id" = $reportId;
```

Query	UPDATE11
Description	Update a report approval to false
Frequency	dozens per day

```
UPDATE "report"
    SET "approval" = FALSE
    WHERE "report_id" = $reportId;
```

Query	UPDATE12
Description	Update a friendship status between two users
Frequency	dozens per day

```
UPDATE "friend"
    SET TYPE = 'accepted'
    WHERE ("friend_id1" = $userId AND "friend_id2" = $userId2);
```

Query	INSERT01
Description	Register a user
Frequency	dozens per day

```
INSERT INTO "user" ("name", "email", "password")
    VALUES ($name, $email, $password);
```

Query	INSERT02
Description	Create a post
Frequency	hundreds per day

```
INSERT INTO "post" ("author_id", "title", "body", "date", "upvotes", "downvotes", TYPE
    VALUES ($userId, $title, $body, current_timestamp, 0, 0, DEFAULT, DEFAULT, DEFAULT
```

Query	INSERT03
Description	Insert a comment
Frequency	thousands per day

```
INSERT INTO "comment" ("user_id","post_id", "comment_to_id", "body", "date", "upvotes"
    VALUES ($userId, $postId, DEFAULT, $body, current_timestamp, 0, 0);
```

Query	INSERT04
Description	Send a message in a chat
Frequency	thousands per day

```
INSERT INTO "message" ("sender_id", "chat_id", "body", "date")
    VALUES ($userId, $chatId, $body, current_timestamp);
```

Query	INSERT05
Description	New notification
Frequency	thousands per day

```
INSERT INTO "notification" ("origin_user_id", "description", "link", "date")
    VALUES ($userId, $notDescription, $link, current_timestamp);
```

Query	INSERT05
Description	New friendship
Frequency	dozens per day

```
INSERT INTO "friend"
 VALUES ($userId, $userId2, DEFAULT);
```

Query	DELETE01
Description	Remove a comment
Frequency	units per day

```
DELETE FROM "comment"
 WHERE "comment_id" = $commentId;
```

Query	DELETE02
Description	Delete a chat
Frequency	units per month

```
DELETE FROM "chat"
 WHERE "chat_id" = $chatId;
```

Query	DELETE03
Description	Delete a message from a chat
Frequency	dozens per day

```
DELETE FROM "message"
 WHERE "message_id" = $messageId;
```

Query	DELETE04
Description	Delete a user interest in an event

Query	DELETE04
Frequency	dozens per month

```
    DELETE FROM "user_interested_in_event"
 WHERE ("user_id" = $userId AND "event_id" = $eventId);
```

2. Proposed Indices

2.1. Performance Indices

Index	IDX01
Related queries	SELECT06
Relation	notified_user
Attribute	user_notified
Type	B-tree
Cardinality	High
Clustering	No
Justification	Table is very large; query SELECT06, used to search the notifications of an user, has to be fast because it's executed many times

```
CREATE INDEX "user_notified" ON "notified_user"("user_notified");
```

Index	IDX02
Related queries	SELECT06
Relation	notified_user
Attribute	notification_id
Type	Hash
Cardinality	High
Clustering	No

Index	IDX02
Justification	Table is very large; query SELECT06, used to search the notifications of an user, has to be fast because it's executed many times ; cardinality is high because notification_id is a foreign key (primary key of the notification table); it's not a good candidate for clustering.

```
CREATE INDEX "notified_user_notification_id"
       ON "notified_user"
       USING hash("notification_id");
```

Index	IDX03
Related queries	SELECT01
Relation	user
Attribute	email
Type	Hash
Cardinality	High
Clustering	No
Justification	Query SELECT01 has to be fast as it is executed many times (login, register, etc); cardinality is high because email is an unique key; it's not a good candidate for clustering.

```
CREATE INDEX "user_email" ON "user"
       USING hash("email");
```

Index	IDX04
Related queries	SELECT15
Relation	student
Attribute	regular_user_id
Type	Hash
Cardinality	High
Clustering	No

Index	IDX04
Justification	Query SELECT15 has to be fast as it is executed many times; cardinality is high because regular_user_id is a foreign key (primary key of the regular_user table); it's not a good candidate for clustering.

```
CREATE INDEX "student_regular_id" ON "student"
       USING hash("regular_user_id");
```

Index	IDX05
Related queries	SELECT16
Relation	teacher
Attribute	regular_user_id
Type	Hash
Cardinality	High
Clustering	No
Justification	Query SELECT16 has to be fast as it is executed many times; cardinality is high because regular_user_id is a foreign key (primary key of the regular_user table); it's not a good candidate for clustering.

```
CREATE INDEX "teacher_regular_id" ON "teacher"
       USING hash("regular_user_id");
```

Index	IDX06
Related queries	SELECT17
Relation	organization
Attribute	regular_user_id
Type	Hash
Cardinality	High
Clustering	No

Index	IDX06
Justification	Query SELECT17 has to be fast as it is executed many times; cardinality is high because regular_user_id is a foreign key (primary key of the regular_user table); it's not a good candidate for clustering.

```
CREATE INDEX "organization_regular_id" ON "organization"
       USING hash("regular_user_id");
```

Index	IDX07
Related queries	SELECT02, SELECT14
Relation	post
Attribute	author_id
Type	Hash
Cardinality	High
Clustering	No
Justification	Queries SELECT02 and SELECT14 have to be fast as they are executed many times; cardinality is high because author_id is a foreign key (primary key of the regular_user table); it's not a good candidate for clustering.

```
CREATE INDEX "post_author_id" ON "post"
       USING hash("author_id");
```

Index	IDX08
Related queries	SELECT04
Relation	post
Attribute	event_id
Type	Hash
Cardinality	High
Clustering	No

Index	IDX08
Justification	Query SELECT04 has to be fast as it is executed many times; cardinality is high because event_id is a foreign key (primary key of the event table); it's not a good candidate for clustering.

```
CREATE INDEX "post_event_id" ON "post"
       USING hash("event_id")
       WHERE "event_id" IS NOT NULL;
```

Index	IDX09
Related queries	SELECT03
Relation	post
Attribute	group_id
Type	Hash
Cardinality	High
Clustering	No
Justification	Query SELECT03 has to be fast as it is executed many times; cardinality is high because group_id is a foreign key (primary key of the group table); it's not a good candidate for clustering.

```
CREATE INDEX "post_group_id" ON "post"
       USING hash("group_id")
       WHERE "group_id" IS NOT NULL;
```

Index	IDX10
Related queries	SELECT05, SELECT14
Relation	comment
Attribute	post_id
Type	Hash
Cardinality	High
Clustering	No

Index	**IDX10**
Justification	Queries SELECT05 and SELECT14 have to be fast as they are executed many times; cardinality is high because post_id is a foreign key (primary key of the post table); it's not a good candidate for clustering.

```
CREATE INDEX "comment_post_id" ON "comment"
       USING hash("post_id")
       WHERE "post_id" IS NOT NULL;
```

Index	**IDX11**
Related queries	SELECT18
Relation	comment
Attribute	comment_to_id
Type	Hash
Cardinality	High
Clustering	No
Justification	Query SELECT18 has to be fast as it is executed many times; cardinality is high because comment_to_id is a foreign key (primary key of the comment table); it's not a good candidate for clustering.

```
CREATE INDEX "comment_comment_to_id" ON "comment"
       USING hash("comment_to_id")
       WHERE "comment_to_id" IS NOT NULL;
```

Index	**IDX12**
Related queries	SELECT07
Relation	event
Attribute	organization_id
Type	Hash
Cardinality	High
Clustering	No

Index	IDX12
Justification	Query SELECT07 has to be fast as it is executed many times; cardinality is high because organization_id is a foreign key (primary key of the organization table); it's not a good candidate for clustering.

```
CREATE INDEX "event_organizer" ON "event"
       USING hash("organization_id");
```

Index	IDX13
Related queries	SELECT08, SELECT14
Relation	friend
Attribute	friend_id1
Type	Hash
Cardinality	High
Clustering	No
Justification	Queries SELECT08 and SELECT14 have to be fast as they are executed many times; cardinality is high because friend_id1 is a foreign key (primary key of the regular_user table); it's not a good candidate for clustering.

```
CREATE INDEX "accepted_friendship" ON "friend"
       USING hash("friend_id1")
       WHERE TYPE = 'accepted';
```

Index	IDX14
Related queries	SELECT09
Relation	friend
Attribute	friend_id1
Type	Hash
Cardinality	High
Clustering	No

Index	IDX14
Justification	Query SELECT09 has to be fast as it is executed many times; cardinality is high because friend_id1 is a foreign key (primary key of the regular_user table); it's not a good candidate for clustering.

```sql
CREATE INDEX "pending_friendship" ON "friend"
       USING hash("friend_id1")
       WHERE TYPE = 'pending';
```

Index	IDX15
Related queries	SELECT10
Relation	message
Attribute	chat_id
Type	Hash
Cardinality	High
Clustering	No
Justification	Query SELECT10 has to be fast as it is executed many times; cardinality is high because chat_id is a foreign key (primary key of the chat table); it's not a good candidate for clustering.

```sql
CREATE INDEX "message_chat" ON "message"
       USING hash("chat_id");
```

Index	IDX16
Related queries	SELECT11
Relation	file
Attribute	post_id
Type	Hash
Cardinality	High
Clustering	No

Index	IDX16
Justification	Query SELECT11 has to be fast as it is executed many times; cardinality is high because post_id is a foreign key (primary key of the post table); it's not a good candidate for clustering.

```
CREATE INDEX "file_post_id" ON "file"
       USING hash("post_id");
```

2.2. Full-text Search Indices

Index	IDX17
Related queries	SELECT19
Relation	post
Attribute	title
Type	GiST
Clustering	No
Justification	To improve the performance of full text searches while searching for post titles; GiST because it's better for dynamic data.

```
CREATE INDEX "search_post_titles" ON "post"
       USING GIST(to_tsvector('english', "title"));
```

Index	IDX18
Related queries	SELECT19
Relation	user
Attribute	name
Type	GiST
Clustering	Yes
Justification	To improve the performance of full text searches while searching for user names; GiST because it's better for dynamic data.

```
CREATE INDEX "search_user_names" ON "user"
       USING GIST(to_tsvector('english', "name"));
```

Index	**IDX19**
Related queries	SELECT19
Relation	group
Attribute	name
Type	GiST
Clustering	Yes
Justification	To improve the performance of full text searches while searching for group names; GiST because it's better for dynamic data.

```
CREATE INDEX "search_group_names" ON "group"
       USING GIST(to_tsvector('english', "name"));
```

Index	**IDX20**
Related queries	SELECT19
Relation	event
Attribute	name
Type	GiST
Clustering	Yes
Justification	To improve the performance of full text searches while searching for event names; GiST because it's better for dynamic data.

```
CREATE INDEX "search_event_names" ON "event"
       USING GIST(to_tsvector('english', "name"));
```

3. Triggers

Trigger	**TRIGGER01**
Description	Update a group's posts status if its own status changes

```
CREATE FUNCTION update_group_posts() RETURNS TRIGGER AS
$BODY$
BEGIN
    UPDATE public."post" SET public."post".TYPE = public."group".TYPE WHERE public."po
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;


CREATE TRIGGER update_group_posts
    AFTER UPDATE ON public."group"
    FOR EACH ROW
    EXECUTE PROCEDURE update_group_posts();
```

Trigger	TRIGGER02
Description	Update an event's posts status if its own status changes

```
CREATE FUNCTION update_event_posts() RETURNS TRIGGER AS
$BODY$
BEGIN
    UPDATE public."post" SET public."post".TYPE = public."event".TYPE WHERE public."po
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;


CREATE TRIGGER update_event_posts
    AFTER UPDATE ON public."event"
    FOR EACH ROW
    EXECUTE PROCEDURE update_event_posts();
```

Trigger	TRIGGER03
Description	Update a user's posts status if its own status changes

```
CREATE FUNCTION update_user_posts() RETURNS TRIGGER AS
$BODY$
BEGIN
    UPDATE public."post" SET public."post".TYPE = public."user".TYPE WHERE public."pos
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;
```

```
CREATE TRIGGER update_user_posts
    AFTER UPDATE ON public."user"
    FOR EACH ROW
    EXECUTE PROCEDURE update_user_posts();
```

Trigger	TRIGGER04
Description	Update a user's posts status if its own status changes, as stated in BR05 (Deleted Account)

```
CREATE FUNCTION update_user_posts() RETURNS TRIGGER AS
$BODY$
BEGIN
    UPDATE public."post" SET public."post".TYPE = public."user".TYPE WHERE public."pos
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;


CREATE TRIGGER update_user_posts
    AFTER UPDATE ON public."user"
    FOR EACH ROW
    EXECUTE PROCEDURE update_user_posts();
```

Trigger	TRIGGER05
Description	A post date must be less than or equal to the actual date, as stated in BR04 (Post Date)

```
CREATE FUNCTION post_date() RETURNS trigger AS
$BODY$
BEGIN
    IF new."date" > now() THEN
    RAISE EXCEPTION 'Invalid post date.';
    END IF;
    RETURN NEW;

END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER post_date
    AFTER INSERT ON public."post"
    FOR EACH ROW
    EXECUTE PROCEDURE post_date();
```

Trigger	TRIGGER06
Description	A new event can only be scheduled for a future date, as stated in BR02 (Event Date)

```
CREATE FUNCTION event_date() RETURNS trigger AS
$BODY$
BEGIN
    IF new."date" < now() THEN
    RAISE EXCEPTION 'Invalid event date.';
    END IF;
    RETURN NEW;


END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER event_date
    AFTER INSERT ON public."event"
    FOR EACH ROW
    EXECUTE PROCEDURE event_date();
```

Trigger	TRIGGER07
Description	Remove a report that has been refused

```
CREATE FUNCTION delete_refused_report() RETURNS trigger AS
$BODY$
BEGIN
    IF new."approval" = FALSE THEN
        DELETE FROM public."report"
        WHERE "report_id" = old.public."report_id";
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER delete_refused_report
    AFTER UPDATE ON public."report"
    EXECUTE PROCEDURE delete_refused_report();
```

Trigger	TRIGGER08
Description	Add/delete a friendship depending on the status of the invitation

```
CREATE FUNCTION friend_status() RETURNS trigger AS
$$
BEGIN
    IF new."friendship_status" = 'accepted' THEN
        Insert into public."friend" ("friend_id1","friend_id2","friendship_status") va
    ELSEIF new."friendship_status" = 'refused' THEN
        DELETE FROM public."friend"
        WHERE ("friend_id1" = old.public."friend_id1" AND "friend_id2" = old.public."f
    END IF;
    RETURN NEW;
END
$$
LANGUAGE plpgsql;


Create TRIGGER friend_status
    AFTER UPDATE ON public."friend"
    EXECUTE PROCEDURE friend_status();
```

Trigger	TRIGGER09
Description	There can only be a single approved account for a given organization, as stated in BR01 (Unique Organization)

```
CREATE FUNCTION unique_org() RETURNS trigger AS
$BODY$
BEGIN
    IF new."approval" = TRUE THEN
        IF EXISTS (Select *
                   from (SELECT *
                         FROM "organization"
                         INNER JOIN "regular_user"
                         on "organization"."regular_user_id" = "regular_user"."regular
             INNER JOIN "user"
                         on "regular_user"."user_id" = "user"."user_id"
                         where "organization"."approval" = TRUE) as "t1"
               INNER JOIN (SELECT *
                               FROM "organization"
                               INNER JOIN "regular_user"
                               on "organization"."regular_user_id" = "regular_user"."r
                               INNER JOIN "user"
                               on "regular_user"."user_id" = "user"."user_id"
                               where "organization"."organization_id" = new."organizat
                    on t1."name" = "t2"."name")
        THEN
            RAISE EXCEPTION 'Two organizations approved with same name';
        END IF;
    END IF;
    RETURN NEW;

END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER unique_org
    BEFORE UPDATE ON public."organization"
    EXECUTE PROCEDURE unique_org();
```

4. Transactions

T01	Register a student
Justification	Given that the registry of a student has different insertions on the database, a transaction is needed to mantain the consistency and to prevent an erroneous code execution. The isolation level is Repeatable Read is used to prevent inconsistent data from being stored.
Isolation level	REPEATABLE READ

```
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- Insert user
 INSERT INTO "user" ("name", "email", "password")
  VALUES ($name, $email, $password);

-- Insert regular_user
 INSERT INTO "regular_user" ("user_id", "personal_info")
  VALUES (currval('user_id_seq'), $info);

--Insert student
 INSERT INTO "student" ("regular_user_id")
  VALUES (currval('regular_user_id_seq'));

COMMIT;
```

T02	Register a teacher
Justification	Given that the registry of a teacher has different insertions on the database, a transaction is needed to mantain the consistency and to prevent an erroneous code execution. The isolation level is Repeatable Read is used to prevent inconsistent data from being stored.

T02	Register a teacher
Isolation level	REPEATABLE READ

```
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- Insert user
 INSERT INTO "user" ("name", "email", "password")
  VALUES ($name, $email, $password);

-- Insert regular_user
 INSERT INTO "regular_user" ("user_id", "personal_info")
  VALUES (currval('user_id_seq'), $info);

--Insert teacher
 INSERT INTO "teacher" ("regular_user_id")
  VALUES (currval('regular_user_id_seq'));

COMMIT;
```

T03	Register an organization
Justification	Given that the registry of an organization has different insertions on the database, a transaction is needed to mantain the consistency and to prevent an erroneous code execution. The isolation level is Repeatable Read is used to prevent inconsistent data from being stored.
Isolation level	REPEATABLE READ

```
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- Insert user
 INSERT INTO "user" ("name", "email", "password")
  VALUES ($name, $email, $password);

-- Insert regular_user
 INSERT INTO "regular_user" ("user_id", "personal_info")
  VALUES (currval('user_id_seq'), $info);

--Insert organization
 INSERT INTO "organization" ("regular_user_id")
  VALUES (currval('regular_user_id_seq'));

COMMIT;
```

5. SQL Code

5.1. Database schema

```sql
DROP TABLE IF EXISTS public."user_interested_in_event";
DROP TABLE IF EXISTS public."friend";
DROP TABLE IF EXISTS public."report";
DROP TABLE IF EXISTS public."notified_user";
DROP TABLE IF EXISTS public."notification";
DROP TABLE IF EXISTS public."user_in_chat";
DROP TABLE IF EXISTS public."user_in_group";
DROP TABLE IF EXISTS public."message";
DROP TABLE IF EXISTS public."chat";
DROP TABLE IF EXISTS public."comment";
DROP TABLE IF EXISTS public."image";
DROP TABLE IF EXISTS public."file";
DROP TABLE IF EXISTS public."post";
DROP TABLE IF EXISTS public."group";
DROP TABLE IF EXISTS public."event";
DROP TABLE IF EXISTS public."organization";
DROP TABLE IF EXISTS public."teacher";
DROP TABLE IF EXISTS public."student";
DROP TABLE IF EXISTS public."regular_user";
DROP TABLE IF EXISTS public."admin";
DROP TABLE IF EXISTS public."user";

DROP TYPE IF EXISTS "friendship_status";
DROP TYPE IF EXISTS status;

CREATE TYPE status AS ENUM ('normal', 'blocked', 'deleted');
CREATE TYPE "friendship_status" AS ENUM ('accepted', 'pending', 'refused');

DROP INDEX IF EXISTS "user_notified";
DROP INDEX IF EXISTS "notified_user_notification_id";
DROP INDEX IF EXISTS "user_email";
DROP INDEX IF EXISTS "student_regular_id";
DROP INDEX IF EXISTS "teacher_regular_id";
DROP INDEX IF EXISTS "organization_regular_id";
DROP INDEX IF EXISTS "post_author_id";
DROP INDEX IF EXISTS "post_event_id";
DROP INDEX IF EXISTS "post_group_id";
DROP INDEX IF EXISTS "comment_post_id";
DROP INDEX IF EXISTS "comment_comment_to_id";
DROP INDEX IF EXISTS "event_organizer";
DROP INDEX IF EXISTS "accepted_friendship";
DROP INDEX IF EXISTS "pending_friendship";
DROP INDEX IF EXISTS "message_chat";
DROP INDEX IF EXISTS "file_post_id";
DROP INDEX IF EXISTS "search_post_titles";
DROP INDEX IF EXISTS "search_user_names";
DROP INDEX IF EXISTS "search_group_names";
DROP INDEX IF EXISTS "search_event_names";

CREATE TABLE public."user"
```

```
(
    "user_id" serial NOT NULL,
    "name" text NOT NULL,
    "email" text NOT NULL,
    "password" text NOT NULL,
    TYPE status NOT NULL DEFAULT 'normal',
    CONSTRAINT "user_pkey" PRIMARY KEY ("user_id"),
    CONSTRAINT "user_email_key" UNIQUE ("email")
);

CREATE TABLE public."admin"
(
    "admin_id" serial NOT NULL,
    "user_id" integer NOT NULL REFERENCES public."user"("user_id") ON DELETE CASCADE,
    CONSTRAINT "admin_pkey" PRIMARY KEY ("admin_id")
);

CREATE TABLE public."regular_user"
(
    "regular_user_id" serial NOT NULL,
    "user_id" integer NOT NULL REFERENCES public."user"("user_id") ON DELETE CASCADE,
    "personal_info" text,
    CONSTRAINT "regular_user_pkey" PRIMARY KEY ("regular_user_id")
);

CREATE TABLE public."student"
(
    "student_id" serial NOT NULL,
    "regular_user_id" integer NOT NULL REFERENCES public."regular_user"("regular_user_
    CONSTRAINT "student_pkey" PRIMARY KEY ("student_id")
);

CREATE TABLE public."teacher"
(
    "teacher_id" serial NOT NULL,
    "regular_user_id" integer NOT NULL REFERENCES public."regular_user"("regular_user_
    CONSTRAINT "teacher_pkey" PRIMARY KEY ("teacher_id")
);

CREATE TABLE public."organization"
(
    "organization_id" serial NOT NULL,
    "regular_user_id" integer NOT NULL REFERENCES public."regular_user"("regular_user_
    "approval" boolean NOT NULL DEFAULT FALSE,
    CONSTRAINT "organization_pkey" PRIMARY KEY ("organization_id")
);

CREATE TABLE public."event"
(
    "event_id" serial NOT NULL,
    "organization_id" integer NOT NULL REFERENCES public."organization"("organization_
    "name" text NOT NULL,
    "location" text NOT NULL,
    "date" timestamp with time zone NOT NULL,
    "information" text NOT NULL,
    CONSTRAINT "event_id_pkey" PRIMARY KEY ("event_id")
);

CREATE TABLE public."group"
(
    "group_id" serial NOT NULL,
    "name" text NOT NULL,
    "information" text NOT NULL,
    TYPE status NOT NULL DEFAULT 'normal',
    CONSTRAINT "group_id_pkey" PRIMARY KEY ("group_id")
);

CREATE TABLE public."user_in_group"
(
    "user_id" integer NOT NULL REFERENCES public."regular_user"("regular_user_id") ON
    "group_id" integer NOT NULL REFERENCES public."group"("group_id") ON DELETE CASCAD
    "admin" boolean NOT NULL DEFAULT FALSE,
    CONSTRAINT "user_in_group_pkey" PRIMARY KEY ("user_id", "group_id")
);

CREATE TABLE public."post"
(
    "post_id" serial NOT NULL,
    "author_id" integer NOT NULL REFERENCES public."regular_user"("regular_user_id") O
    "title" text NOT NULL,
    "body" text NOT NULL,
    "date" timestamp with time zone NOT NULL DEFAULT now(),
    "upvotes" integer NOT NULL DEFAULT 0,
    "downvotes" integer NOT NULL DEFAULT 0,
    TYPE status NOT NULL DEFAULT 'normal',


    "event_id" integer DEFAULT NULL References public."event"("event_id") ON DELETE CA
    "group_id" integer DEFAULT NULL References public."group"("group_id") ON DELETE CA

    CONSTRAINT "post_id_pkey" PRIMARY KEY ("post_id"),
    CONSTRAINT "date_ck" CHECK ( "date" <= now() ),
    CONSTRAINT "upvotes_ck" CHECK ( "upvotes" >= 0 ),
    CONSTRAINT "downvotes_ck" CHECK ( "downvotes" >= 0 ),
    CONSTRAINT "bellong_ck" CHECK ( NOT ( ("event_id" IS NOT NULL) AND ( "group_id" IS
);

CREATE TABLE public."file"
(
    "file_id" serial NOT NULL,
    "post_id" integer REFERENCES public."post"("post_id"),
    "file_path" text NOT NULL,

    CONSTRAINT "file_id_pkey" PRIMARY KEY ("file_id")
);

CREATE TABLE public."image"
(
    "image_id" serial NOT NULL,
    "file_id" integer NOT NULL REFERENCES public."file"("file_id") ON DELETE CASCADE,
    "group_id" integer DEFAULT NULL REFERENCES public."group"("group_id") ON DELETE CA
    "event_id" integer DEFAULT NULL REFERENCES public."event"("event_id") ON DELETE CA
    "regular_user_id" integer  DEFAULT NULL REFERENCES public."regular_user"("regular_
    "post_id" integer DEFAULT NULL REFERENCES public."post"("post_id") ON DELETE CASCA

    CONSTRAINT "image_id_pkey" PRIMARY KEY ("image_id"),
    CONSTRAINT "bellong_ck" CHECK ( (Case when ("group_id" iS NOT NULL) then 1 else 0
                                    (Case when ("event_id" iS NOT NULL) then 1 else 0
                                    (Case when ("regular_user_id" iS NOT NULL) then 1
                                    (Case when ("post_id" iS NOT NULL) then 1 else 0 e
);

CREATE TABLE public."comment"
(
    "comment_id" serial NOT NULL,
    "user_id" integer NOT NULl, --NEW
    "post_id" integer DEFAULT NULL REFERENCES public."post"("post_id") ON DELETE CASCA
    "comment_to_id" integer DEFAULT NULL REFERENCES public."comment"("comment_id") ON
    "body" text NOT NULL,
    "date" timestamp with time zone NOT NULL DEFAULT now(),
    "upvotes" integer NOT NULL DEFAULT 0,
    "downvotes" integer NOT NULL DEFAULT 0,


    CONSTRAINT "comment_id_pkey" PRIMARY KEY ("comment_id"),
    CONSTRAINT "dif_cmmt" CHECK ( "comment_id" != "comment_to_id" ),
    CONSTRAINT "date_ck" CHECK ( "date" <= now() ),
    CONSTRAINT "upvotes_ck" CHECK ( "upvotes" >= 0 ),
    CONSTRAINT "downvotes_ck" CHECK ( "downvotes" >= 0 ),
    CONSTRAINT "bellong_ck" CHECK ( (("post_id" is NOT NULL) AND ("comment_to_id" IS N
);

CREATE TABLE public."chat"
(
    "chat_id" serial NOT NULL,
    CONSTRAINT "chat_id_pkey" PRIMARY KEY ("chat_id")
);

CREATE TABLE public."message"
(
    "message_id" serial NOT NULL,
    "sender_id" integer NOT NULL REFERENCES public."regular_user"("regular_user_id") O
    "chat_id" integer NOT NULL REFERENCES public."chat"("chat_id") ON DELETE CASCADE,
    "body" text NOT NULL,
    "date" timestamp with time zone NOT NULL DEFAULT now(),

    CONSTRAINT "message_id_pkey" PRIMARY KEY ("message_id"),
    CONSTRAINT "date_ck" CHECK ( "date" <= now() )
);

CREATE TABLE public."user_in_chat"
(
    "user_id" integer NOT NULL REFERENCES public."regular_user"("regular_user_id") ON
    "chat_id" integer NOT NULL REFERENCES public."chat"("chat_id") ON DELETE CASCADE,
    CONSTRAINT "user_in_chat_pkey" PRIMARY KEY ("user_id", "chat_id")
```

```
);

CREATE TABLE public."notification"
(
    "notification_id" serial NOT NULL,
    "origin_user_id" integer NOT NULL REFERENCES public."regular_user"("regular_user_i
    "description" text NOT NULL,
    "link" text NOT NULL,
    "date" timestamp with time zone NOT NULL DEFAULT now(),

    CONSTRAINT "notification_id_pkey" PRIMARY KEY ("notification_id"),
    CONSTRAINT "date_ck" CHECK ( "date" <= now() )
);

CREATE TABLE public."notified_user"
(
    "notification_id" integer NOT NULL REFERENCES public."notification"("notification_
    "user_notified" integer NOT NULL REFERENCES public."regular_user"("regular_user_id
    "seen" boolean DEFAULT FALSE NOT NULL,
    CONSTRAINT "notified_user_pkey" PRIMARY KEY ("notification_id", "user_notified")
);


CREATE TABLE public."report"
(
    "report_id"  serial NOT NULL,
    "reporter_id" integer NOT NULL REFERENCES public."regular_user"("regular_user_id")
    "approval" boolean DEFAULT NULL,
    "reason" text NOT NULL,

    "reported_user_id" integer REFERENCES public."regular_user"("regular_user_id") ON
    "reported_event_id" integer REFERENCES public."event"("event_id") ON DELETE CASCAD
    "reported_post_id" integer REFERENCES public."post"("post_id") ON DELETE CASCADE,
    "reported_comment_id" integer REFERENCES public."comment"("comment_id") ON DELETE
    "reported_group_id" integer REFERENCES public."group"("group_id") ON DELETE CASCAD

    CONSTRAINT "report_pkey" PRIMARY KEY ("report_id"),
        CONSTRAINT "bellong_ck" CHECK ( (Case when ("reported_user_id" iS NOT NULL) th
                                    (Case when ("reported_event_id" iS NOT NULL) then
                                    (Case when ("reported_post_id" iS NOT NULL) then 1
                                    (Case when ("reported_comment_id" iS NOT NULL) the
                                    (Case when ("reported_group_id" iS NOT NULL) then
);


CREATE TABLE public."friend"
(
    "friend_id1" integer NOT NULL REFERENCES public."regular_user"("regular_user_id")
    "friend_id2" integer NOT NULL REFERENCES public."regular_user"("regular_user_id")
    TYPE "friendship_status" NOT NULL DEFAULT 'pending',
    CONSTRAINT "friend_pkey" PRIMARY KEY ("friend_id1", "friend_id2")
);

CREATE TABLE public."user_interested_in_event"
(
    "user_id" integer NOT NULL REFERENCES public."user"("user_id") ON DELETE CASCADE,
    "event_id" integer NOT NULL REFERENCES public."event"("event_id") ON DELETE CASCAD
    CONSTRAINT "user_interested_in_event_pkey" PRIMARY KEY ("user_id", "event_id")
);



-- USER NOTIFIED INDEX
CREATE INDEX "user_notified" ON "notified_user"("user_notified");
CREATE INDEX "notified_user_notification_id" ON "notified_user" USING hash("notificati

-- LOGIN EMAIL INDEX
CREATE INDEX "user_email" ON "user" USING hash("email");

-- REGULAR USER ID FOREIGN KEYS INDEXES
CREATE INDEX "student_regular_id" ON "student" USING hash("regular_user_id");
CREATE INDEX "teacher_regular_id" ON "teacher" USING hash("regular_user_id");
CREATE INDEX "organization_regular_id" ON "organization" USING hash("regular_user_id")
CREATE INDEX "post_author_id" ON "post" USING hash("author_id");

-- POST FOREIGN KEYS INDEXES
CREATE INDEX "post_event_id" ON "post" USING hash("event_id") WHERE "event_id" IS NOT
CREATE INDEX "post_group_id" ON "post" USING hash("group_id") WHERE "group_id" IS NOT

-- COMMENT FOREIGN KEYS INDEXES
CREATE INDEX "comment_post_id" ON "comment" USING hash("post_id") WHERE "post_id" IS N
CREATE INDEX "comment_comment_to_id" ON "comment" USING hash("comment_to_id") WHERE "c

-- EVENT ORGANIZER INDEX
CREATE INDEX "event_organizer" ON "event" USING hash("organization_id");

-- FRIENDSHIP INDEX
CREATE INDEX "accepted_friendship" ON "friend" USING hash("friend_id1") WHERE TYPE = '
CREATE INDEX "pending_friendship" ON "friend" USING hash("friend_id1") WHERE TYPE = 'p

-- CHAT INDEX
CREATE INDEX "message_chat" ON "message" USING hash("chat_id");

-- FILE INDEX
CREATE INDEX "file_post_id" ON "file" USING hash("post_id");

-- GIST SEARCH INDEXES
CREATE INDEX "search_post_titles" ON "post" USING GIST(to_tsvector('english', "title")
CREATE INDEX "search_user_names" ON "user" USING GIST(to_tsvector('english', "name"));
CREATE INDEX "search_group_names" ON "group" USING GIST(to_tsvector('english', "name")
CREATE INDEX "search_event_names" ON "event" USING GIST(to_tsvector('english', "name")



DROP TRIGGER IF EXISTS update_group_posts ON "group" CASCADE;
DROP TRIGGER IF EXISTS update_event_posts ON "event" CASCADE;
DROP TRIGGER IF EXISTS update_user_posts ON "user" CASCADE;
DROP TRIGGER IF EXISTS friend_status ON "friend" CASCADE;
DROP TRIGGER IF EXISTS delete_refused_report ON "report" CASCADE;
DROP TRIGGER IF EXISTS event_date ON "event" CASCADE;
DROP TRIGGER IF EXISTS post_date ON "post" CASCADE;
DROP TRIGGER IF EXISTS unique_org ON "organization" CASCADE;

DROP FUNCTION IF EXISTS update_group_posts() CASCADE;
DROP FUNCTION IF EXISTS update_event_posts() CASCADE;
DROP FUNCTION IF EXISTS update_user_posts() CASCADE;
DROP FUNCTION IF EXISTS friend_status() CASCADE;
DROP FUNCTION IF EXISTS delete_refused_report() CASCADE;
DROP FUNCTION IF EXISTS event_date() CASCADE;
DROP FUNCTION IF EXISTS post_date() CASCADE;
DROP FUNCTION IF EXISTS unique_org() CASCADE;

CREATE FUNCTION update_group_posts() RETURNS TRIGGER AS
$BODY$
BEGIN
    UPDATE public."post" SET public."post".TYPE = public."group".TYPE WHERE public."po
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;


CREATE TRIGGER update_group_posts
    AFTER UPDATE ON public."group"
    FOR EACH ROW
    EXECUTE PROCEDURE update_group_posts();



CREATE FUNCTION update_event_posts() RETURNS TRIGGER AS
$BODY$
BEGIN
    UPDATE public."post" SET public."post".TYPE = public."event".TYPE WHERE public."po
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;


CREATE TRIGGER update_event_posts
    AFTER UPDATE ON public."event"
    FOR EACH ROW
    EXECUTE PROCEDURE update_event_posts();



CREATE FUNCTION update_user_posts() RETURNS TRIGGER AS
$BODY$
BEGIN
    UPDATE public."post" SET public."post".TYPE = public."user".TYPE WHERE public."pos
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;
```

```
CREATE TRIGGER update_user_posts
    AFTER UPDATE ON public."user"
    FOR EACH ROW
    EXECUTE PROCEDURE update_user_posts();



CREATE FUNCTION friend_status() RETURNS trigger AS
$$
BEGIN
    IF new."friendship_status" = 'accepted' THEN
        Insert into public."friend" ("friend_id1","friend_id2","friendship_status") va
    ELSEIF new."friendship_status" = 'refused' THEN
        DELETE FROM public."friend"
        WHERE ("friend_id1" = old.public."friend_id1" AND "friend_id2" = old.public."f
    END IF;
    RETURN NEW;
END
$$
LANGUAGE plpgsql;


Create TRIGGER friend_status
    AFTER UPDATE ON public."friend"
    EXECUTE PROCEDURE friend_status();



CREATE FUNCTION delete_refused_report() RETURNS trigger AS
$BODY$
BEGIN
    IF new."approval" = FALSE THEN
        DELETE FROM public."report"
        WHERE "report_id" = old.public."report_id";
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER delete_refused_report
    AFTER UPDATE ON public."report"
    EXECUTE PROCEDURE delete_refused_report();



CREATE FUNCTION event_date() RETURNS trigger AS
$BODY$
BEGIN
    IF New."date" < now() THEN
    RAISE EXCEPTION 'Invalid event date.';
    END IF;
    RETURN NEW;
```

```
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER event_date
    AFTER INSERT ON public."event"
    FOR EACH ROW
    EXECUTE PROCEDURE event_date();



CREATE FUNCTION post_date() RETURNS trigger AS
$BODY$
BEGIN
    IF new."date" > now() THEN
    RAISE EXCEPTION 'Invalid post date.';
    END IF;
    RETURN NEW;

END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER post_date
    AFTER INSERT ON public."post"
    FOR EACH ROW
    EXECUTE PROCEDURE post_date();



CREATE FUNCTION unique_org() RETURNS trigger AS
$BODY$
BEGIN
    IF new."approval" = TRUE THEN
        IF EXISTS (Select * from
                    (SELECT * FROM "organization" INNER JOIN "regular_user" on "organi
                        INNER JOIN "user" on "regular_user"."user_id" = "user"."user_i
                    INNER JOIN
                    (SELECT * FROM "organization" INNER JOIN "regular_user" on "organi
                        INNER JOIN "user" on "regular_user"."user_id" = "user"."user_i
                    on t1."name" = "t2"."name"
                    )
        THEN
            RAISE EXCEPTION 'Two organizations approved with same name';
        END IF;
    END IF;
    RETURN NEW;

END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER unique_org
```

```
    BEFORE UPDATE ON public."organization"
    EXECUTE PROCEDURE unique_org();
```

5.2. Database population

```
insert into public."user" ("name", "email", "password") values('Admin1', 'admin1@gg.pt
insert into public."user" ("name", "email", "password") values('Joaquim Rodrigues', 'j
insert into public."user" ("name", "email", "password") values('Paulo Tavares', 'paulo
insert into public."user" ("name", "email", "password") values('AEISEP', 'aeisep@isep.
insert into public."user" ("name", "email", "password") values('Gustavo Torres', 'tgus
insert into public."user" ("name", "email", "password") values('Pedro Esteves', 'pmest
insert into public."user" ("name", "email", "password") values ('Vitor Ventuzelos', 'b
insert into public."user" ("name", "email", "password") values ('José Martins', 'marti
insert into public."user" ("name", "email", "password") values ('Marta Camões', 'mcamo
insert into public."user" ("name", "email", "password") values ('Diana Magalhães', 'dm
insert into public."user" ("name", "email", "password") values ('Tiago Pessoa', 'tpess
insert into public."user" ("name", "email", "password") values ('Ricardo Pinto', 'rpin
insert into public."user" ("name", "email", "password") values ('Maria Soares', 'msoar
insert into public."user" ("name", "email", "password") values ('Francisco Costa', 'fc
insert into public."user" ("name", "email", "password") values ('José Silva', 'runcolh
insert into public."user" ("name", "email", "password") values ('Miguel Alvim', 'malvi
insert into public."user" ("name", "email", "password") values ('Dinis Pereira', 'dinp
insert into public."user" ("name", "email", "password") values ('Soraia Tavares', 'sta
insert into public."user" ("name", "email", "password") values ('Osvaldo Antunes', 'oa
insert into public."user" ("name", "email", "password") values ('Mariana Castro', 'mca
insert into public."user" ("name", "email", "password") values ('Ana Faria', 'afaria@f
insert into public."user" ("name", "email", "password") values ('Rui Cardoso', 'rcardo
insert into public."user" ("name", "email", "password") values ('AEFEUP', 'aefeup@feup
insert into public."user" ("name", "email", "password") values ('Admin2', 'admin2@admi


insert into public."admin" ("user_id") values (1);
insert into public."admin" ("user_id") values (24);


insert into public."regular_user" ("user_id", "personal_info") values (2, 'Just one re
insert into public."regular_user" ("user_id", "personal_info") values (3, 'Just anothe
insert into public."regular_user" ("user_id", "personal_info") values (4, 'Another one
insert into public."regular_user" ("user_id", "personal_info") values (5, 'DB user');
insert into public."regular_user" ("user_id", "personal_info") values (6, 'GameJam use
insert into public."regular_user" ("user_id", "personal_info") values (7, 'Another one
insert into public."regular_user" ("user_id", "personal_info") values (8, 'FLUP studen
insert into public."regular_user" ("user_id", "personal_info") values (9, 'FCUP teache
insert into public."regular_user" ("user_id", "personal_info") values (10, 'FLUP stude
insert into public."regular_user" ("user_id", "personal_info") values (11, 'I am an IS
insert into public."regular_user" ("user_id", "personal_info") values (12, 'FMUP stude
insert into public."regular_user" ("user_id", "personal_info") values (13, 'ICBAS stud
insert into public."regular_user" ("user_id", "personal_info") values (14, 'FFUP stude
insert into public."regular_user" ("user_id", "personal_info") values (15, 'Sports stu
insert into public."regular_user" ("user_id", "personal_info") values (16, 'FADEUP tea
insert into public."regular_user" ("user_id", "personal_info") values (17, 'FMUP stude
insert into public."regular_user" ("user_id", "personal_info") values (18, 'Pharmacy s
insert into public."regular_user" ("user_id", "personal_info") values (19, 'Sports stu
insert into public."regular_user" ("user_id", "personal_info") values (20, 'Science st
insert into public."regular_user" ("user_id", "personal_info") values (21, 'FEP studen
insert into public."regular_user" ("user_id", "personal_info") values (22, 'FEP teache


insert into public."student" ("regular_user_id") values (2);
insert into public."student" ("regular_user_id") values (5);
insert into public."student" ("regular_user_id") values (6);
insert into public."student" ("regular_user_id") values (7);
insert into public."student" ("regular_user_id") values (8);
insert into public."student" ("regular_user_id") values (10);
insert into public."student" ("regular_user_id") values (12);
insert into public."student" ("regular_user_id") values (13);
insert into public."student" ("regular_user_id") values (14);
insert into public."student" ("regular_user_id") values (15);
insert into public."student" ("regular_user_id") values (17);
insert into public."student" ("regular_user_id") values (18);
insert into public."student" ("regular_user_id") values (19);
insert into public."student" ("regular_user_id") values (20);
insert into public."student" ("regular_user_id") values (21);


insert into public."teacher" ("regular_user_id") values (3);
insert into public."teacher" ("regular_user_id") values (9);
insert into public."teacher" ("regular_user_id") values (11);
insert into public."teacher" ("regular_user_id") values (16);


insert into public."organization" ("regular_user_id", "approval") values (4, TRUE);


insert into public."event" ("organization_id", "name", "location", "date", "informatio


insert into public."group" ("name", "information", TYPE) values ('Grupo de LBAW', 'Gru
insert into public."group" ("name", "information", TYPE) values ('UP/IPP Group', 'Estu


insert into public."user_in_group" ("user_id", "group_id") values (2, 1);
insert into public."user_in_group" ("user_id", "group_id") values (5, 1);
insert into public."user_in_group" ("user_id", "group_id") values (6, 1);
insert into public."user_in_group" ("user_id", "group_id") values (7, 1);
insert into public."user_in_group" ("user_id", "group_id") values (2, 2);
insert into public."user_in_group" ("user_id", "group_id") values (5, 2);
insert into public."user_in_group" ("user_id", "group_id") values (6, 2);
insert into public."user_in_group" ("user_id", "group_id") values (7, 2);
insert into public."user_in_group" ("user_id", "group_id") values (8, 2);
insert into public."user_in_group" ("user_id", "group_id") values (9, 2);
insert into public."user_in_group" ("user_id", "group_id") values (10, 2);
insert into public."user_in_group" ("user_id", "group_id") values (12, 2);
insert into public."user_in_group" ("user_id", "group_id") values (13, 2);
insert into public."user_in_group" ("user_id", "group_id") values (14, 2);
insert into public."user_in_group" ("user_id", "group_id") values (15, 2);
insert into public."user_in_group" ("user_id", "group_id") values (17, 2);
insert into public."user_in_group" ("user_id", "group_id") values (18, 2);
insert into public."user_in_group" ("user_id", "group_id") values (19, 2);
insert into public."user_in_group" ("user_id", "group_id") values (20, 2);
insert into public."user_in_group" ("user_id", "group_id") values (21, 2);


insert into public."post" ("author_id", "title", "body", "date", "upvotes", "downvotes
insert into public."post" ("author_id", "title", "body", "date", "upvotes", "downvotes
insert into public."post" ("author_id", "title", "body", "date", "upvotes", "downvotes
insert into public."post" ("author_id", "title", "body", "date", "upvotes", "downvotes
insert into public."post" ("author_id", "title", "body", "date", "upvotes", "downvotes
insert into public."post" ("author_id", "title", "body", "date", "upvotes", "downvotes
insert into public."post" ("author_id", "title", "body", "date", "upvotes", "downvotes
insert into public."post" ("author_id", "title", "body", "date", "upvotes", "downvotes


insert into public."comment" ("user_id","post_id", "comment_to_id", "body", "date", "u
insert into public."comment" ("user_id","post_id", "comment_to_id", "body", "date", "u
insert into public."comment" ("user_id","post_id", "comment_to_id", "body", "date", "u


insert into public."chat" DEFAULT VALUES ;
insert into public."chat" DEFAULT VALUES ;


insert into public."user_in_chat" ("user_id", "chat_id") values (2,1);
insert into public."user_in_chat" ("user_id", "chat_id") values (5,1);
insert into public."user_in_chat" ("user_id", "chat_id") values (6,1);
insert into public."user_in_chat" ("user_id", "chat_id") values (7,1);
insert into public."user_in_chat" ("user_id", "chat_id") values (3,1);
insert into public."user_in_chat" ("user_id", "chat_id") values (9,1);
insert into public."user_in_chat" ("user_id", "chat_id") values (11,1);
insert into public."user_in_chat" ("user_id", "chat_id") values (16,1);


insert into public."message" ("sender_id", "chat_id", "body", "date") values (2, 1, 'B
insert into public."message" ("sender_id", "chat_id", "body", "date") values (5, 1, 'B
insert into public."message" ("sender_id", "chat_id", "body", "date") values (6, 1, 'E
insert into public."message" ("sender_id", "chat_id", "body", "date") values (7, 1, 'M
insert into public."message" ("sender_id", "chat_id", "body", "date") values (9, 2, 'C


insert into public."report" ("reporter_id", "approval", "reason", "reported_user_id",
insert into public."report" ("reporter_id", "approval", "reason", "reported_user_id",


insert into public."file" ("post_id", "file_path") values ( 7,'../files/test.txt');
insert into public."file" ("post_id", "file_path") values ( 8,'../files/image.png');


insert into public."image" ("file_id", "post_id") values (2, 8);


insert into public."notification" ("origin_user_id", "description", "link", "date") va
insert into public."notified_user" ("notification_id", "user_notified") values (1, 5);
insert into public."notified_user" ("notification_id", "user_notified") values (1, 7);
insert into public."notified_user" ("notification_id", "user_notified") values (1, 10)


insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (1, 4, DEFAULT)
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (1, 15, DEFAULT
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (1, 18, DEFAULT
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (2, 3, 'accepte
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (2, 4, 'accepte
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (2, 5, 'accepte
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (2, 6, 'accepte
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (2, 7, 'accepte
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (2, 8, DEFAULT)
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (2, 10, DEFAULT
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (4, 3, DEFAULT)
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (4, 6, DEFAULT)
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (5, 7, DEFAULT)
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (5, 8, DEFAULT)
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (6, 8, DEFAULT)
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (6, 19, DEFAULT
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (7, 12, DEFAULT
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (7, 17, DEFAULT
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (8, 11, DEFAULT
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (8, 10, DEFAULT
insert into public."friend" ("friend_id1", "friend_id2", TYPE ) values (10, 20, DEFAUL


insert into public."user_interested_in_event" ("user_id","event_id") values(2,1);
insert into public."user_interested_in_event" ("user_id","event_id") values(4,1);
insert into public."user_interested_in_event" ("user_id","event_id") values(7,1);
insert into public."user_interested_in_event" ("user_id","event_id") values(10,1);
```

The code is also in the git repository.

Revision history

Changes made to the first submission: (none)

GROUP2034, 29/03/2020

- Gustavo Torres, up201706473@fe.up.pt
- Joaquim Rodrigues, up201704844@fe.up.pt (Editor)
- Pedro Esteves, up201705160@fe.up.pt
- Vitor Ventuzelos, up201706403@fe.up.pt