Requirements Manager

Enrico Marchionni
enrico.marchionni@studio.unibo.it

August 24, 2024

Abstract

ReM[1] is an administrative software that makes you minimize time and maximize result in the *Software Engineering* process of collecting and creating requirements.

[1] *ReM*, Requirements Manager.

Contents

Analysis

It's requested to realize a database that allows a team to optimize Software's Analysis by managing Requirements creation. Each one of them descends from a Request that could be created by a team member or by the project customers.

Interview

The aim is to manage requirements during all phases of software production and revision storing away all final and in production data relative to releases already published or yet to publish. Everything has to start from a *Request*. Once it has been created the relative *Requirement/s* can be developed.

The system admin has to create users and the relative database linked to a software. Requests can be created by a team member or a customer. Requirements can be developed by team members only. Requests and therefore Requirements can obviously be added or even edited in later releases. It's not important to keep track of every update or change, the important is to keep track of final data just before a release. The timeline can be added by a team member only and domain conditions has to be respected.

Requirements have to be structured with a title, a type (functional, non functional), a version, a description, a body, one or more files, progress data, a creation time and a last modified time. Moreover for a requirement it has to be known the user that did the first and last modifies and an history of updates with times and users that did them. Furthermore is important to consider that requirements could be arranged in a tree structure in the most complex cases.

Requests structure consists of a title, a type, a version, a description, a body, one or more files, a creation time and a last modified time. Besides for a request it has to be known the user that did the first and last modifies and an history of updates with times and users that did them.

A requirement has to be associated with one and only one request, a request can be associated with multiple requirements.

A request has to be approved by a team member before developing it into a requirement. After the approval a customer has to accept the request approval, only then the relative requirement can be developed. When the approval is done the request mustn't be able to be modified.

Users has to be saved with personal info such as username, name, surname, email, phone and optionally the company name.

The releases should also save a short description and a name that has to be an identifier between the timelines. It's important to permit the creation of a new timeline only if every requirement was completed, or this mustn't be permitted. In addition every request has to be approved, formalized into requirement and completed before the version creation.

Every data that was affected by the version cannot be overridden so their last modified time should be previous to the version creation time.

Actions

The mainly requested actions are:

1. Subscribe a new user;
2. View a request;
3. Create a request;
4. Update a request;
5. Approve a request;
6. View a requirement;
7. Create a requirement;
8. Update a requirement;
9. Disable a requirement;
10. Create a release;
11. Show requirements and requests relative to a specific release;
12. View single requirement history (every change for each release);
13. Obtain an average of the progress time relative to a branch of the requirements tree;

Design

Conceptual Design

Glossary

A glossary of terms with description, synonyms and links used in Interview is here explained in Table 1.

Term	Description	Synonyms	Links
Request	Customer request of what the system should do		Requirement, release
Requirement	Formal description of what the system should do		Request, release
	Requirements + Requests	Data	
Editor	User that can add Requirements	Team member	
Guest	User that can only add Requests	Customer	
User	Editor + Guest	User	Editing, versioning
Release	A timeline that confirms the fact that a software version was released	Version, timeline	Data

Table 1: Glossary of terms

Requirements

This section aims to restructure requirements according to the Interview phase.

Functional

- General
  - It is demanded to realize a database that manages requirements;

  - Information that have to be stored are **Requests**, **Requirements**, **Users** and **Releases**;

- Users
  - **Users** are structured with username, name, surname, email, phone and optionally the company name;
  - **Users** are divided into two groups:
    * **Editors** - they approve *Requests*, develop relative *Requirements* and decides *Releases*; in some cases they could also create *Requests*;
    * **Guests** - they create *Requests* that once approved cannot be edited but will be developed into *Requirements*;

- Requests
  - **Requests** are structured with a title, a type, a version (positive integer), a description, a body, one or more files, a creation time and a last modified time;

- Requirements
  - **Requirements** are structured with a title, a type (functional, non functional), a version (positive integer), a description, a body, one or more files, progress data (a percentage), a creation time and a last modified time;
  - **Requirements** could be arranged in a tree structure;

- Releases
  - **Releases** are structured with a unique name (software version name), a short description and the creation time;
  - **Releases** are timelines that can be created only if all *Requests* had been approved and converted into already fully developed *Requirements*;

- Conditions
  - all **Requests** and **Requirements** relative to a **Release** have to be historicized;
  - **Requests** and **Requirements** could be modified or even invalidated in later **Releases**;
  - a **Requirement** has to be associated with one and only one **Request**, a **Request** can be associated with multiple **Requirements**.

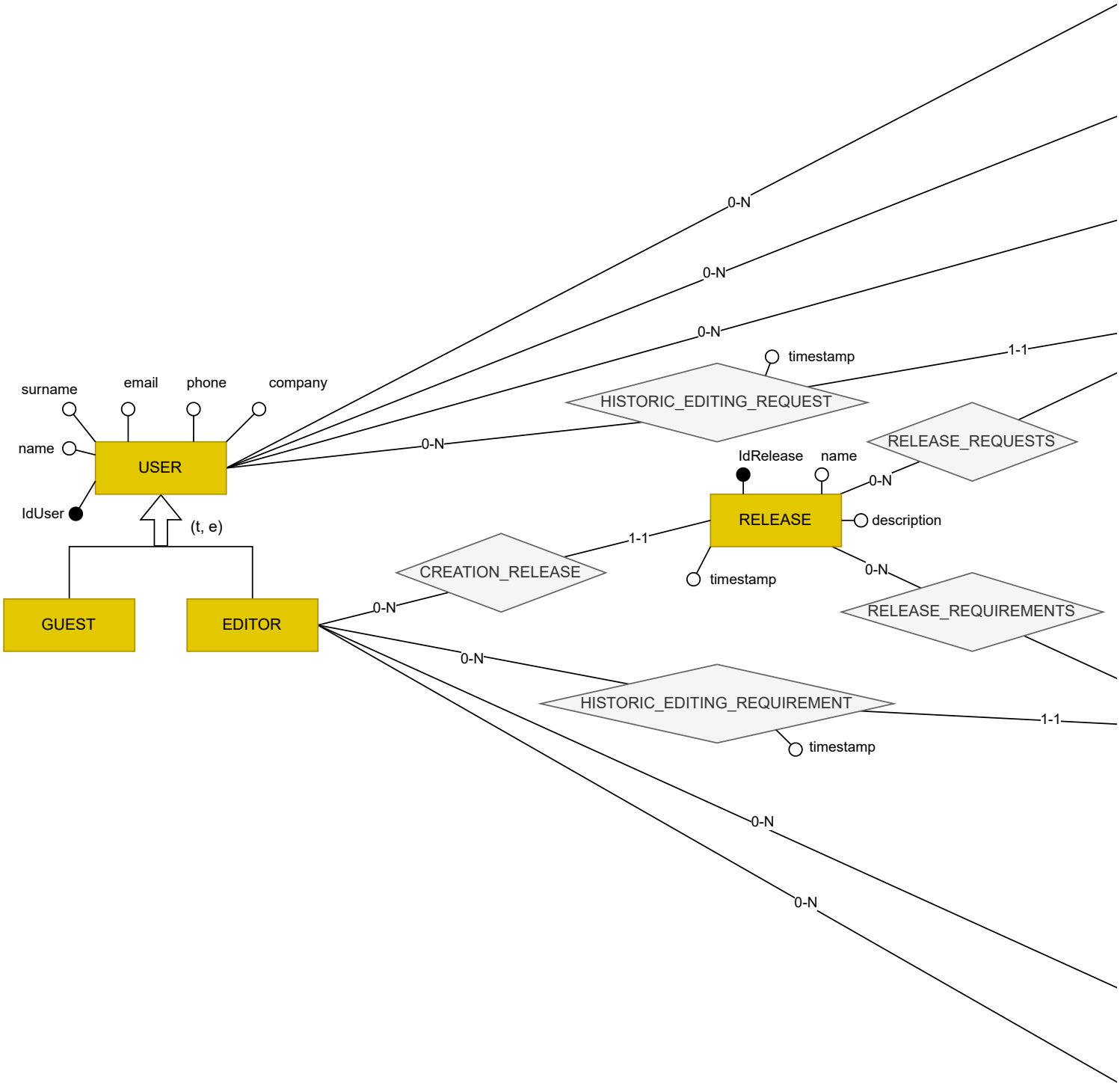
Non-Functional

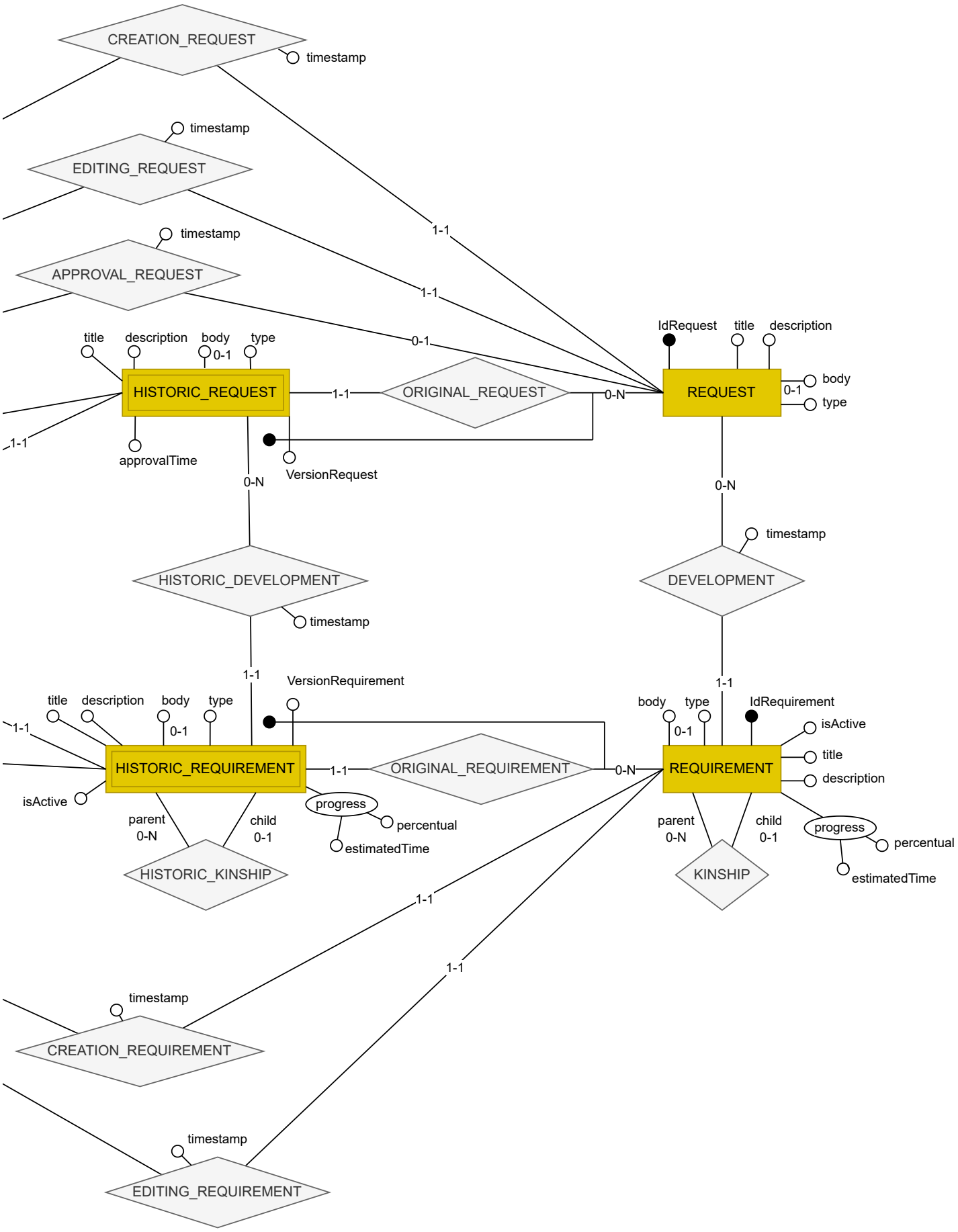
- Data access has to be really fast;
- CRUD[2] operations should be non blocking in final GUI[3];

E-R diagram

It was chosen a mixed strategy to build the E-R diagram.

[2] *CRUD*, Create, read, update, delete.
[3] *GUI*, Graphical user interface.


surname
email
phone
company
name
USER
IdUser
(t, e)
GUEST
EDITOR
0-N
0-N
0-N
0-N
timestamp
HISTORIC_EDITING_REQUEST
1-1
RELEASE_REQUESTS
IdRelease
name
RELEASE
description
0-N
1-1
CREATION_RELEASE
timestamp
0-N
RELEASE_REQUIREMENTS
0-N
0-N
HISTORIC_EDITING_REQUIREMENT
1-1
timestamp
0-N
0-N


CREATION_REQUEST
timestamp
EDITING_REQUEST
timestamp
APPROVAL_REQUEST
timestamp
1-1
1-1
0-1
title
description
body
0-1
type
HISTORIC_REQUEST
1-1
ORIGINAL_REQUEST
0-N
IdRequest
title
description
REQUEST
body
0-1
type
1-1
approvalTime
VersionRequest
0-N
0-N
timestamp
HISTORIC_DEVELOPMENT
DEVELOPMENT
timestamp
1-1
VersionRequirement
1-1
title
description
body
type
0-1
body
type
IdRequirement
0-1
isActive
title
description
HISTORIC_REQUIREMENT
1-1
ORIGINAL_REQUIREMENT
0-N
REQUIREMENT
isActive
parent
0-N
child
0-1
progress
percentual
estimatedTime
HISTORIC_KINSHIP
parent
0-N
child
0-1
progress
percentual
estimatedTime
KINSHIP
1-1
1-1
timestamp
CREATION_REQUIREMENT
timestamp
EDITING_REQUIREMENT

Logical Design

Table of volumes

Data and expected volumes are described in Table 2.

Concept	Construct	Volume
USER	E	100
GUEST	E	90
EDITOR	E	10
REQUEST	E	50
HISTORIC_REQUEST	E	40
REQUIREMENT	E	75
HISTORIC_REQUIREMENT	E	50
RELEASE	E	5
DEVELOPMENT	R	70
HISTORIC_DEVELOPMENT	R	50
KINSHIP	R	5
HISTORIC_KINSHIP	R	5
CREATION_RELEASE	R	5
RELEASE_REQUESTS	R	40
RELEASE_REQUIREMENTS	R	50
CREATION_REQUEST	R	50
EDITING_REQUEST	R	50
HISTORIC_EDITING_REQUEST	R	40
APPROVAL_REQUEST	R	45
CREATION_REQUIREMENT	R	70
EDITING_REQUIREMENT	R	70
HISTORIC_EDITING_REQUIREMENT	R	50
ORIGINAL_REQUEST	R	40
ORIGINAL_REQUIREMENT	R	50

Table 2: Volumes of data

Operations and frequency

Operations and frequency are described in Table 3.

Code	Operation	Frequency
1	Subscribe a new user	10 every month
2	View a request	200 every day
3	Create a request	2 every day
4	Update a request	5 every day
5	Approve a request	1 every day
6	View a requirement	200 every day
7	Create a requirement	2 every day
8	Update a requirement	50 every day
9	Disable a requirement	1 every month
10	View a release	2 every day
11	Create a release	2 every year
12	Show requirements and requests relative to a specific release	50 every day
13	View single requirement history (every change for each release)	10 every day
14	Obtain an average of the progress time relative to a branch of the requirements tree	20 every day

Table 3: Operations and frequency

Navigation and access

1. Subscribe a new user

User subscription described in Table 4.

Concept	Construct	Access	Type
User	E	1	W
		Total: 1W → 10 every month	

Table 4: Subscribe a new user

2. View a request

Request visualization described in Table 5.

Concept	Construct	Access	Type
Request	E	1	R
		Total: 1R → 200 every day	

Table 5: View a request

3. Create a request

Request creation described in Table 6.

Concept	Construct	Access	Type
Request	E	1	W
		Total: 1W → 2 every day	

Table 6: Create a request

4. Update a request

Request updating described in Table 7.

Concept	Construct	Access	Type
Request	E	1	R
Request	E	1	W
		Total: 1R + 1W → 10 every day	

Table 7: Update a request

5. Approve a request

Request approval described in Table 8.

Concept	Construct	Access	Type
Request	E	1	R
Request	E	1	W
		Total: 1R + 1W → 2 every day	

Table 8: Approve a request

6. View a requirement

Requirement visualization described in Table 9.

Concept	Construct	Access	Type
Requirement	E	1	R
		Total: 1R → 200 every day	

Table 9: View a requirement

7. Create a requirement

Requirement creation described in Table 10.

Concept	Construct	Access	Type
Requirement	E	1	W
		Total: 1W → 2 every day	

Table 10: Create a requirement

8. Update a requirement

Requirement updating described in Table 11.

Concept	Construct	Access	Type
Requirement	E	1	R
Requirement	E	1	W
		Total: 1R + 1W → 100 every day	

Table 11: Update a requirement

9. Disable a requirement

Requirement disabling described in Table 12.

Concept	Construct	Access	Type
Requirement	E	1	R
Requirement	E	1	W
		Total: 1R + 1W → 2 every month	

Table 12: Disable a requirement

10. View a release

Release visualization described in Table 13.

Concept	Construct	Access	Type
Release	E	1	R
		Total: 1R → 2 every day	

Table 13: View a release

11. Create a release

Release creation described in Table 14.

Concept	Construct	Access	Type
Release	E	1	W
		Total: 1W → 2 every year	

Table 14: Create a release

12. Show requirements and requests relative to a specific release

Requests and requirements visualization described in Table 15.

Concept	Construct	Access	Type
Release	E	1	R
Release_requests	E	8 (40 / 5)	R
Requests	E	8	R
Release_requirements	E	10 (50 / 5)	R
Requirements	E	10	R
		Total: 37R → 1850 every day	

Table 15: Show requirements and requests relative to a specific release

13. View single requirement history

Requests and requirements visualization described in Table 16.

Concept	Construct	Access	Type
Requirement	E	1	R
Original_requirement	E	0.7 (50 / 75)	R
Historic_requirement	E	0.7	R
		Total: 2.4R → 24 every day	

Table 16: View single requirement history

14. Obtain an average of the progress time relative to a branch of the requirements tree

Requests and requirements visualization described in Table 17.

Concept	Construct	Access	Type
Requirement	E	1	R
Kinship	E	0.07 (5 / 75)	R
Historic_requirement	E	0.07	R
		Total: 1.14R → 22.8 every day	

Table 17: Obtain an average of the progress time relative to a branch of the requirements tree

Redundancy Analysis

10. View a Release

Release visualization without redundancy described in Table 18.

Concept	Construct	Access	Type
Release	E	1	R
Release_requests	E	8 (40 / 5)	R
Release_requirements	E	10 (50 / 5)	R
		Total: 19R → 38 every day	

Table 18: View a request without redundancy

Restructured Diagram

Hierarchy elimination

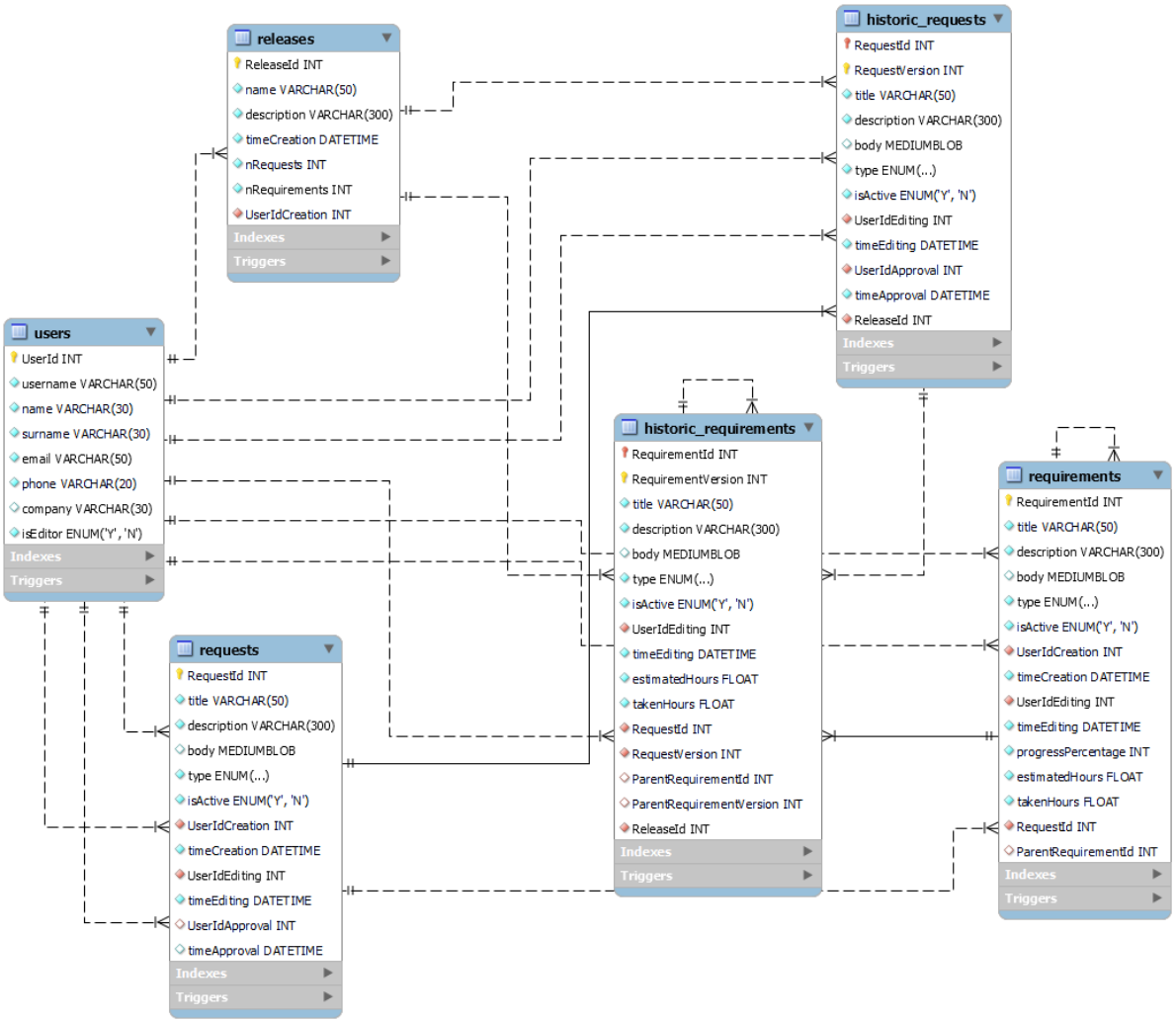
It was chosen the collapse to the top for the necessity to have a unique identifier for the users (this helps in references). The distinction between Guest and Editor users is given to a flag that indicates if a user is also an Editor.

Associations many-to-many

In this case it was chosen to translate them by putting an external identifier in the 1-1 relation to refer to the other entity.

The E-R diagram restructured is explained in figure 1.

Figure 1: Logic



Entity and Relationships Translation in Relations

USERS(UserId, username, name, surname, email, phone, company*, isEditor)

NOTES:
Unique(username)
Unique(email)

RELEASES(ReleaseId, name, description, timeCreation, nRequests, nRequirements, UserId-Creation: *USERS*)

NOTES:
Unique(name)

REQUESTS(RequestId, title, description, body*, type, isActive, UserIdCreation: *USERS*, timeCreation, UserIdEditing: *USERS*, timeEditing, UserIdApproval*: *USERS*, timeApproval*)

NOTES:
A constraint should make 'UserIdApproval' and 'timeApproval' to be both present or absent at the same time.

REQUIREMENTS(RequirementId, title, description, body*, type, isActive, UserIdCreation: *USERS*, timeCreation, UserIdEditing: *USERS*, timeEditing, progressPercentage, estimatedHours, takenHours, RequestId: *REQUESTS*, ParentRequirementId*: *REQUIREMENTS*)

HISTORIC_REQUESTS(RequestId: *REQUESTS*, RequestVersion, title, description, body*, type, isActive, UserIdEditing: *USERS*, timeEditing, timeApproval, ReleaseId: *RELEASES*)

HISTORIC_REQUIREMENTS(RequirementId: *REQUIREMENTS*, RequirementVersion, title, description, body*, type, isActive, UserIdEditing: *USERS*, timeEditing, estimatedHours, takenHours, (RequestId, RequestVersion): *HISTORIC_REQUESTS*, (ParentRequirementId, ParentRequirementVersion)*: *HISTORIC_REQUIREMENTS*, ReleaseId: *RELEASES*)

Relations Analysis

Attributes An attribute of a relation A can be set in an instance with a value that is its domain $dom(A)$. Let's consider $T = \{A_1, A_2, ..., A_n\}$ a set of attributes.

Tuples A tuple t on T is a function that maps for each $A_i \in T$ a value in $dom(A_i)$.

Schema of Relation A schema of relation on T is defined by the name of the relation R and a set of attributes T, an it is denoted by $R(T)$. An extension of it can be denoted by r.

Functional Dependency A functional dependency (FD), considering a schema $R(T)$ and $X, Y \subseteq T$ with $X, Y \neq \emptyset$, is defined as $X \rightarrow Y$ (X functionally determines Y) if:

$$\forall\, t_1, t_2 \in r : t_1[X] = t_2[Y] \Rightarrow t_1[Y] = t_2[Y]$$

Note that FD is a feature of the $R(T)$ and not its extension r.

Trivial Functional Dependency A functional dependency is trivial if $Y \subseteq X$.

Super Key $K \subseteq T$ is a super-key of $R(T) \Leftrightarrow K \rightarrow T$.

Key A key is a super-key without subset of attributes that is another super-key.

Attributes and Keys Attributes can be:

1. *prime*: is part of at least one key in the schema.
2. *non-prime*: it is not part of any key in the schema.

Normal Forms It was considered the following normal forms:

1. First Normal Form (1NF):
   (a) A schema $R(T)$ is in 1NF if and only if the domain of each attribute contains only atomic values (simple, indivisible), and the value of each attribute in a tuple is a single value from the domain of that attribute.
2. Second Normal Form (2NF):
   (a) A schema $R(T)$ with constraints F is in 2NF if and only if every non-prime attribute depends entirely (not partially) on every candidate key of the schema, meaning there is no partial dependency of a non-prime attribute on a key.
3. Third Normal Form (3NF):
   (a) A schema $R(T)$ with constraints F is in 3NF if and only if every non-prime attribute does not depend transitively on any key, meaning there is no transitive dependency of a non-prime attribute on a key.
   (b) Every relation can reach this level after modifies of splitting or merging.
4. Boyce-Codd Normal Form (BCNF):
   (a) A schema $R(T)$ with constraints F is in BCNF if and only if, for every non-trivial functional dependency $X \to Y$ defined on it, X is a super-key of $R(T)$.
   (b) It extends what seen till now to prime attributes.
   (c) Some relationships cannot reach this level even after modifies of splitting or merging.

All relations considered are in BCNF.

Deployment

MySQL Queries

In this section the previously described operations are translated into MySQL queries.

Subscribe a new user operation query described in Listing 1.

```
-- UserId was set as AUTO_INCREMENT, so it can be omitted here
INSERT INTO `USERS`
    (`username`, `name`, `surname`,
     `email`, `phone`, `company`,
     `isEditor`)
VALUES
    (?, ?, ?, ?, ?, ?, ?);
```

Listing 1: op. 1

View a request operation query described in Listing 2.

```
SELECT *
FROM `REQUESTS`
WHERE `RequestId` = ?;
```

Listing 2: op. 2

Create a request operation query described in Listing 3.

```
INSERT INTO `REQUESTS`
    (`title`, `description`, `type`,
     `isActive`, `UserIdCreation`,
     `UserIdEditing`, `timeEditing`)
VALUES
    (?, ?, ?, ?, ?, ?, NOW());
```

Listing 3: op. 3

Update a request operation query described in Listing 4.

```
UPDATE `REQUESTS`
SET `title` = ?,
    `description` = ?,
    `body` = ?,
    `type` = ?,
    `isActive` = ?,
    `UserIdEditing` = ?,
    `timeEditing` = NOW()
```

```
    WHERE `RequestId` = ?;
```

Listing 4: op. 4

Approve a request operation query described in Listing 5.

```
    UPDATE `REQUESTS`
    SET `UserIdApproval` = ?,
        `timeApproval` = NOW(),
        `UserIdEditing` = ?,
        `timeEditing` = NOW()
    WHERE `RequestId` = ?;
```

Listing 5: op. 5

View a requirement operation query described in Listing 6.

```
    SELECT *
    FROM `REQUIREMENTS`
    WHERE `RequirementId` = ?;
```

Listing 6: op. 6

Create a requirement operation query described in Listing 8.

```
    INSERT INTO `REQUIREMENTS`
        (`title`, `description`, `type`,
         `isActive`, `UserIdCreation`,
         `UserIdEditing`, `timeEditing`
         `RequestId`, `ParentRequirementId`)
    VALUES
        (?, ?, ?, ?, ?, ?, NOW(), ?, ?);
```

Listing 7: op. 7

Update a requirement operation query described in Listing 8.

```
    UPDATE `REQUIREMENTS`
    SET `title` = ?,
        `description` = ?,
        `body` = ?,
        `type` = ?,
        `isActive` = ?,
        `UserIdEditing` = ?,
        `timeEditing` = NOW(),
        `progressPercentage` = ?,
        `estimatedHours` = ?,
        `takenHours` = ?
    WHERE `RequestId` = ?;
```

Listing 8: op. 8

Disable a requirement operation query described in Listing 9.

```
    UPDATE `REQUIREMENTS`
    SET `isActive` = ?,
```

```
        `UserIdEditing` = ?,
        `timeEditing` = NOW(),
    WHERE `RequestId` = ?;
```

Listing 9: op. 9

View a release operation query described in Listing 10.

```
    SELECT *
    FROM `RELEASES`
    WHERE `ReleaseId` = ?;
```

Listing 10: op. 10

Create a release operation query described in Listing 11.

```
    INSERT INTO `RELEASES`
        (`name`, `description`, `UserIdCreation`)
    VALUES
        (?, ?, ?);
```

Listing 11: op. 11

Show requirements and requests relative to a specific release operation query described in Listing 12.

```
    SELECT
        h_requests.*,
        h_requirements.*
    FROM
        `RELEASES` releases,
        `HISTORIC_REQUESTS` h_requests,
        `HISTORIC_REQUIREMENTS` h_requirements
    WHERE
        releases.`ReleaseId` = h_requests.`ReleaseId` AND
        releases.`ReleaseId` = h_requirements.`ReleaseId`;
    -- is equivalent to
    SELECT
        h_requests.*,
        h_requirements.*
    FROM
        `RELEASES` releases,
    INNER JOIN `HISTORIC_REQUESTS` h_requests
        ON requests.`ReleaseId` = releases.`ReleaseId`
    INNER JOIN `HISTORIC_REQUIREMENTS` h_requirements
        ON h_requirements.`ReleaseId` = h_requirements.`ReleaseId`;
```

Listing 12: op. 12

View single requirement history operation query described in Listing 13.

```
    SELECT
        h_requirements.*
    FROM
```

```
        `REQUIREMENTS` requirements,
        `HISTORIC_REQUIREMENTS` h_requirements
    WHERE
        requirements.`RequirementId` = h_requirements.`RequirementId`
```

Listing 13: op. 13

Obtain an average of the progress time relative to a branch of the requirements tree operation query described in Listing 14.

```
    -- Common Table Expressions are supported from MySQL 8
    WITH RECURSIVE `RequirementTree` AS (
        -- Anchor member: Select the starting node (i.e. the root of the
           branch)
        SELECT *
        FROM `REQUIREMENTS`
        WHERE `RequirementId` = ? -- Replace ? with the ID of the root
           requirement
        UNION ALL
        -- Recursive member: Select all children of the current level
        SELECT r.*
        FROM `REQUIREMENTS` r
        INNER JOIN `RequirementTree` rt ON r.`ParentRequirementId` = rt.`
           RequirementId`
    )
    -- Final selection: Sum up the values from the CTE
    SELECT
        SUM(`progressPercentage`) AS `totalProgressPercentage`,
        SUM(`estimatedHours`) AS `totalEstimatedTime`,
        SUM(`takenHours`) AS `totalTakenTime`
    FROM `RequirementTree`;
```

Listing 14: op. 14

Application

It was chosen a C# .NET Core 8.0 application running on Windows with Windows Forms GUI. As anticipated with the queries the database is MySQL.

Following this report philosophy it was adopted a database first approach. So at first the Analysis was written, with: requirements, conceptual and logical design, and then the implementation started. A file named 'ReM.sql' was used as a script, partially derived from design and partially handwritten, to create schema (database) and tables in MySQL. It includes features to avoid illegal situations using checks and triggers, except for historic data that should be modified only behind the scenes in a real application.

So as to map data from MySQL to .NET Core 8.0 it was chosen to use Entity Framework Core. In particular with the process of 'Scaffolding' [4] it was possible to map directly the 'ReM' schema from MySQL to .NET Core 8.0 classes.

[4] https://dev.mysql.com/doc/connector-net/en/connector-net-entityframework-core-scaffold-example.html