TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Melissa Eenmaa 175096IDDR

# GIFTER API

Project for “Building Distributed Systems”

Supervisor: Andres Käver

Tallinn 2020

# Table of contents

# List of abbrevations and terms

| | |
|---|---|
| NULL | In SQL it means that there is no data for the corresponding field. Null equals null is not a legal comparison, because nothing cannot be compared to nothing as both are unknown. There is nothing to compare. |
| PK | Primary key, which is a unique table column, added to ensure the uniqueness of each record. It can be composite, meaning its value could be combined from multiple other columns in the same table. PK is an identifier, which means it must to be comparable, therefore logically it can never be null as null is not comparable, nor can things be identified by undefined values. |
| FK | Foreign key, which is a column to reference another table. It must be in sync with the referenced table's PK, which means foreign key needs to have a value that is existing in a primary key in another table. This is ensured by the database engine and is called referential integrity. Foreign key can be null in case of optional relationship but is generally not. |
| Datetime2 | A data type in SQL which default value is not NULL but some out-of-reach date value. |
| Index | It is related to binary search and creates a sorted catalogue of fields – database generates a copy of the column and stores the record there. When index is set as unique then it means there cannot be duplicates – value in the column must be unique. Index can be composite over several columns. |
| ORM | Object Relational Mapper. E.g. Entity Framework |
| CRUD | Create-Read-Update-Delete, database basic functions |
| POC | Proof of concept |

| | |
|---|---|
| DDD | Domain Driven Development |
| DAO | Database Access Object |
| EF | Entity Framework |
| UOW | Unit of Work |
| DRY principle | Don’t Repeat Yourself principle |
| SRP principle | Single-Responsibility principle |
| JWT | JSON Web Token (authentication) |
| API | Application Programming Interface |
| GET | Request method for getting existing data |
| POST | Request method for adding/creating new data |
| PUT | Request method for editing existing data |
| DELETE | Request method for deleting existing data |
| DAL | Data Access Layer |
| BLL | Business Logic Layer |
| DTO | Data Transfer Object |
| Controllers | Classes with CRUD methods for handling requests between client and business logic layer |
| UI / UX | User interface / User experience |
| FAQ | Frequently Asked Questions |

# 1 Introduction

The idea of this application is to facilitate the process of gift giving between people.

It is useful for each occasion involving gifts, especially birthdays and holidays such as Christmas. However, it could be used independent from any holidays in order to pleasantly surprise another person.

The goal is to reduce the stress of wondering what to purchase and prevent purchasing something that the receiver might not like, just for the sake of getting a present.

Furthermore, the application will make the user’s life better in the sense that they will be receiving only the things they actually need or want, not something they would have to give away or have it disarray their home. The element of surprise on what they will receive, and from who, will still be maintained.

Another good use of the platform would be the campaigns, for instance during holiday seasons, which let the users donate gifts to children in need.

In summary, everyone has their own profile where they can list the things they wish to have, with or without gift descriptions, images and URLs to corresponding web shops. Someone who wishes to give another person a gift can choose something from their list and purchase or make it for them, while being sure the receiver will be satisfied. There will be a feature implemented that will prevent double gifts - so that multiple people do not plan the same gift for the same person. There will be a convenient overview of all the presents the user has decided to make for others as well as other useful features to make gift giving less stressful.

# 2 Initially planned features

## 2.1 Basic functionality

Basic functionality covers the minimum viable product (MVP) of the project scope.

1. **Home page**
    - Contact info and FAQ
    - Campaign advertisement(s)
    - Personal user notifications
2. **Profile page** with two sections: general info and wishlist.
    - General info
        - Info shown to others is username, full name, age, gender, profile picture, description (main interests/hobbies, liked things).
        - Private info that is not shown to anyone is mandatory e-mail address *and optionally home address (and/or closest SmartPost/Omniva and phone number).*
    - Wishlist
        - A list of specific items / gifts the user would like to receive. Basically, a bucket list of things they would like to have but can't buy/don't feel like buying.
        - There are certain things to fill in about each list item
            - **full name** of the thing (mandatory)
            - **description** (optional)
            - **URL** for a shop where it could be purchased (optional)
            - **image** of the item (optional)
        - There are actions/options you can do with each item
            - **"edit/delete/mark as received"** when it's your list
            - **"will gift/mark as gifted/cancel gift"** when it's someone else's list you are interacting with.
        - Each item can be in one of 3 states
            - **active** (shown). Initial state when added to the list. Everyone can see it.
            - **reserved** (disabled/marked). When someone decides to gift it, they can mark it as reserved. It will also show how much time has passed since reservation. Everyone can see it except for the receiver.
            - **archieved** (hidden/gifted). When someone has given the gift, they can mark it as gifted which will hide it from the

list. The owner of the list gets a notification and can confirm they have gotten it - this will remove it from the list and move it to the “gotten gifts” archieve, “given gifts” for the other user respectively. If they claim they haven’t gotten it, the item will be shown in their list as active again.

3. **To gift/reserved gifts page**
   - User has a private feed on their app where gifts are listed that they have marked as “will gift” so they can keep track of what to buy to whom.
   - There are buttons for marking it further as “gifted” or “cancel gift”.
     - Gifted - will send it to archieve not shown in wishlist anymore.
     - Cancelled - will send it back to the feed of the receiver, shown as active so others could buy it instead.
4. **Archieve page** - 2 sections. User can delete things from their own archieve.
   - Received things
     - Items can be marked as received by yourself on your profile
     - Items in your list that are marked as “gifted” by others will appear in your archieve with a confirmation where you can choose “confirm” (i got it as a gift indeed, keep in archieve) or “deny” (i haven’t gotten it, send it back to my wishlist as active)
   - Gifted things
     - Things you have marked as “gifted” in other people’s wishlists will appear here.
5. **Friends list page** - 3 sections and 3 views under this tab.
   - Sections
     - Search
       - Can search for a user to add them as friends or check their profile
       - Search results will show in the place of friends list
     - Invitation link
       - Can invite people to join who don’t have an account yet - based on e-mail or phone number
     - List of people
       - By default shows a list of friends - their profile picture, name, username and last active date
       - When searching people will show everyone who corresponds to query, with “Add friend” button present if they’re not already added
   - Views
     - Search + invitation link + friends list
     - Search + invitation link + people you searched for
     - A specific person’s profile

## 2.2 Advanced functionality

Advanced functionality is something to be done in the future as it is out of scope for current project due to limited time.

1. **“Find on amazon” button** for each item on the wishlist
    - Potential monetization oppurtunity (a deal with Amazon - or other Partner - get some money from either from each visit, each buy or both)
    - Item’s name set in the wishlist will be inserted into Amazon search automatically but the person can change it as needed by themselves, of course
    - In case url to webshop is not provided... or always?

2. **An option to “request confirmation” on “reserved gift” items**
    - Will send a notification to the one who marked it as such - automatically asking them whether they still plan to gift it or not. They can choose to:
        i. Leave it reserved - if they still plan on getting it
        ii. Mark it as gifted - if they have already given it away - so it will be removed from the gift list
        iii. Choose “cancel gift” so it will be active in the wishlist again and someone else (the one who requested confirmation) will know they can gift it instead.
    - Point is to “nudge” people to not leave/forget items reserved without actually ever getting the gift. If it’s been reserved for a long time and someone thinks they could gift it instead, they can confirm it with the one who reserved it.
3. **Creating campaigns for donations**
    - For example during Christmas, to see a list of children from the orphanage or just poor families, and what they’d like to get.
    - You can mark gifts reserved in the app as usual
    - When you buy the gift put the kid’s name on it and send it to the creator of the campaign as per their instructions
    - These campaigns should somehow go through the company or be made by certified members - for safety measures (avoid scams).
4. **“Send gift via post” option**
    - To be able to send gifts to people via post without knowing their address or other shipping information
    - Shipping process
        i. Done by the maintainers of the app - they will receive the gift you send and forward it to the correct address - like a proxy to keep the addresses private

ii. Or there could be boxes to pick them up from like SmartPost - or reuse SmartPost/Omniva service somehow

# 3 Choosing SQL soft delete/update approach

The architecture of the application's database should be analysed in order to choose the optimal approach for the process of updating and deleting existing data.

A good practise would be to not delete any data from the database without having a backup solution – otherwise there is no way to restore it, should it be needed, and no good overview of the application's history. That is, unless it is legally required for the data to be deleted permanently. The other concern is accountability for action.

There are multiple approaches for accomplishing the aforementioned goals, most popular being the creation of regular backups of the database, using the *soft delete* method and having audit logs.

This chapter will focus on soft delete, which itself has different ways of implementation. There will be an analysis on the topic to find the optimal approach for current use case.

## 3.1 Analysis on soft delete / update relevance

There should be a good reason for using everything – "just in case" is not a good argument – especially when there are other alternatives for handling the same issue that the method is trying to solve. Considering that, the soft delete method too should not be used as the default blindly.

If the reason for using soft deletes is data loss, then what is the argument for using it instead of regular database backups, which wouldn't complicate the existing state of the data by mixing active and inactive records, and could as well provide the lost data when needed?

If the reason is the need to still interact with the deleted data within the application due to business value, the more appropriate term for such state would be *archived* instead of *deleted*. In that case, a separate history table could be used for such data, instead of polluting existing tables with additional columns, combined keys, irrelevant rows and therefore inconvenient join queries.

As for other negative sides of soft deletion besides complication of the existing tables, using them in combination with UNIQUE index could cause unfortunate edge-cases. E.g. trying to register a deleted username will fail, even though it is not in use. Another concern would be performance regarding filtering, storage space due to grow-only behaviour, noise regarding records – there might be millions of them while only couple of them could still be legitimate, inconvenience during usage, etc.

One common database feature that could not be used with soft delete is cascade delete, which is an efficient way to automatically delete all tables that are only connected to the table that was deleted – basically child tables to a parent one. Clearing out these references to deleted data prevents foreign key violations.
E.g. in current application we have users and they have a list of gifts on their profile. Once the user is deleted, all their data should be deleted as well – manually it could become difficult to find all that was connected to them and be sure it is all handled. With cascade delete it would be quick and simple – if the user no longer exists then nor should their profile and the gifts mentioned in it. This is logical and mostly the case in these situations.

Soft deletion would also complicate a situation where it would be legally required to remove all data correlating to a certain person from the database. If everything is built up on soft deletion it is not that easy to do – it would require an exception, a workaround to the usual deletion process.

Positive side of the soft delete is that it makes it convenient to recover data in a situation where invalid deletion has catastrophic consequences. Another positive aspect of the method is being able to track every change that ever happens to the data. With this method it is convenient and easy to handle the deleted and modified data – with alternatives such as using multiple archival tables or even different databases it might get quite complicated.

In summary, hard deletion would make the application's architecture and usage much simpler and easier. It would require less analysing and covering of edge-cases while also being more straight-forward to use. It is quite common to assume that unnecessary complications should be avoided, and the simpler the application's design and codebase is kept, the better. There is also the aspect of smaller database and faster queries. Therefore, the conclusion is that soft deletion should be avoided unless it is justified for the use case or it should be done in a way that would consider all the negative aspects and work around them in the best possible way, whether it be using some existing solution or creating a thorough analysis on the architecture of the database, providing a solution for each rising issue.

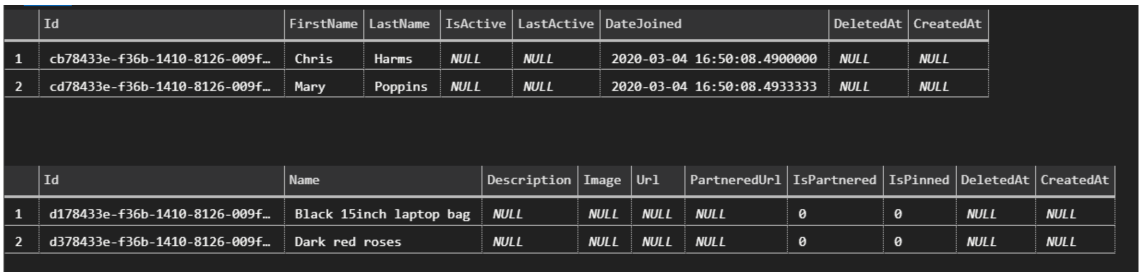
### 3.1.1 **Sources**

https://stackoverflow.com/questions/2549839/are-soft-deletes-a-good-idea
https://jameshalsall.co.uk/posts/why-soft-deletes-are-evil-and-what-to-do-instead
https://ayende.com/blog/4157/avoid-soft-deletes
https://weblogs.asp.net/fbouma/soft-deletes-are-bad-m-kay

## 3.2 Analysis on soft delete / soft update approach

Assuming the current application will use soft deletes to keep the version history, the optimal approach for this should be determined.

The following image is a presentation of an example database that has DeletedAt and CreatedAt columns added to support soft deletion.



| | Id | FirstName | LastName | IsActive | LastActive | DateJoined | DeletedAt | CreatedAt |
|---|---|---|---|---|---|---|---|---|
| 1 | cb78433e-f36b-1410-8126-009f… | Chris | Harms | NULL | NULL | 2020-03-04 16:50:08.4900000 | NULL | NULL |
| 2 | cd78433e-f36b-1410-8126-009f… | Mary | Poppins | NULL | NULL | 2020-03-04 16:50:08.4933333 | NULL | NULL |

| | Id | Name | Description | Image | Url | PartneredUrl | IsPartnered | IsPinned | DeletedAt | CreatedAt |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | d178433e-f36b-1410-8126-009f… | Black 15inch laptop bag | NULL | NULL | NULL | NULL | 0 | 0 | NULL | NULL |
| 2 | d378433e-f36b-1410-8126-009f… | Dark red roses | NULL | NULL | NULL | NULL | 0 | 0 | NULL | NULL |

We should create a composite Primary Key out of Id itself and DeletedAt date for us to be able to track the changes – have multiple versions of the same record while the key is still unique.

Creating a primary key from columns Id and DeletedAt would mean that DeletedAt could not be null – primary key can never be null, but it would be if one of its components is. This is a problem because DeletedAt must be null for us to know it is an active data – DeletedAt should only have the date if the data has actually been “deleted”. Of course, another approach would be to set some default date value for the DeletedAt, for example a date in the far future, but this would create an exception or unusual workaround that must be remembered by everyone working on the project, hence creating a complication and making the queries error prone. There is an approach for soft deletes that does not contain a date at all – only marks it as deleted or not – a column called IsDeleted. But that could not be used as a part of a composite PK as it is binary – not many combinations. It also doesn’t preserve history in the sense that you cannot see when the item was deleted.

Similar problems appear when using this logic combining CreatedAt with Id. Question arises about the behaviour of CreatedAt column: should it change on every update or always contain the initial creation date? The first approach would break the existing relationships as they still reference the previous primary key not the updated one. The latter approach, however, would impose a problem regarding the uniqueness of the primary key. Both the old and the new data would have the same primary key, but it should always be unique. This is why we cannot use CreatedAt as a part of composite key.

Let’s assume we’re going to use Id and DeletedAt (DAT) columns to create the composite key. We have a record “a” that we want to update, so we will create a new record “b” as a copy from “a”, except we change the DAT to current date. This will be historical data. Then we update the actual “a” record value. If we want to change it again, we repeat the process by creating a copy of “a”, this time with Id “c” and DAT as current date. This approach seems fine but might introduce problems regarding relationships between tables.

Let’s say we have tables Person and Contact and they have one-to-many relationship. Updating the child table Contact is trivial and is done the same way as described before. However, updating the parent table would mean the child table has lost the reference to the old data – it is pointing to the updated value only. This doesn’t matter from the SQL queries viewpoint as FKs can be ignored, joins will be done in any case, but from the Entity Framework point of view it is a problem regarding includes. Indexes will not help in this case. If this solution is still preferable, the same things done in SQL would have to be implemented in the EF manually, one by one, so it doesn’t seem like the best solution. Also, if parent record is soft deleted then children will have to be soft deleted as well and this will also be manual.

If we use on update cascade then we would have to still manually mark the child deleted and it will be a problem if there are many records corresponding to the changed record.

Another solution would be to split the Foreign Key in the child table – only use the Id part, find the corresponding records in the parent table and then find the correct one from the parent table based on that. But this would be using the FK incorrectly, not as intended.

For one-to-one relationships, the relationship must be optional. Otherwise no records could be added to neither of the tables. As an example, if we have Person and Photo tables and both require the other one to exist, how can we add a Person if there are no images yet, but image is required? We cannot add both at the same time. So, this relationship needs to be optional – either Photo doesn't have to have a Person attached to it or Person doesn't have to have a Photo. The latter makes more sense from the business logic point of view, so, let's say Photo is optional. This means every photo needs to have a person, therefore foreign key should be on the Photo side. To ensure the relationship really is one-to-one, not one-to-many (person can only have one photo, not multiple), we will have to set the PersonId foreign key as UNIQUE.

And so the new problem arises – how to enforce uniqueness in case of soft deletes? Using UNIQUE constraint becomes an issue with soft delete. Because if something is unique it cannot be entered again, but when soft deleting it is not actually removed from the table, which will prevent the same thing from being added again. So once you delete something it cannot be added again unless it is done via reverting of the deleted field.

All these edge-case problems lead to thinking that someone must have gone through all the same steps many times before, and there must already be a better solution for this than visiting the same problems once again. Common issues require general solutions so they could be reused, therefore it seems reasonable to search for an existing tool handling the soft deletes.

## 3.3 Chosen approach - Temporal Tables

Since 2016 there is a new solution in SQL Server called Temporal Tables which allows the SQL Server database engine to handle the soft deletes and updates. A new history record is created in the corresponding history table whenever a record in the primary table is updated, regardless of how you perform the update – whether it be directly in SQL or via Entity Framework.

### 3.3.1 **How it works**

How it works is that there is a history table for each active table which keeps the rows and their "start" and "end" times, starting time being the last update date and ending time being the time of deletion. When the record has not been deleted yet, the end date is by default "9999-12-31 23:59:59.9999999", not NULL.

The active table itself, with regular queries, will show active records only, but the historical records can be queried through it with special temporal queries utilizing SYSTEM_TIME which is set to be the period between "start" and "end" time of records in the table. During a query, it can be specified what the time period should be (SELECT … FROM … FOR SYSTEM_TIME BETWEEN '{0}' and '{1}' WHERE …). But you can select all changes by using "FOR SYSTEM_TIME ALL". The temporal query against the active table will be resolved by the database engine, which will look through the additional columns in the history table and the content of the primary table as well.

When you cascade delete a parent table of a child table, when both have version history turned on, then the child will be deleted as well but both of their histories preserved. The same applies to updating data.

However, some things to consider are that history table cannot have constraints (primary key, foreign key, table or column constraints), indexed views are not supported on top of temporal queries (queries that use FOR SYSTEM_TIME clause), SYSTEM_TIME period columns cannot be manually modified with INSERT and UPDATE columns, they will be blocked. Regular queries only affect data in the current table. To query data in the history table, you must use temporal queries. AFTER triggers are permitted only on the current table. They are blocked on the history table to avoid invalidating the DML logic. The following objects or properties are not replicated from the current to the history table when history table is created: period definition, identity definition, indexes, statistics, check constraints, triggers, partitioning configuration, permissions, row-level security predicates. Cascade delete and update were not supported in the 2016 version but are now supported, starting from the SQL Server 2017. However, these things don't seem to matter in the context of the current application.

### 3.3.2 **Entity Framework with Temporal Tables**

Entity Framework Core itself does not yet fully support temporal tables, but it is being worked on by the EF team. In the meanwhile, there are still ways how it can be used without making the application too complex. Currently the best approach seems to be to use the library or NuGet package made by George Findulov, which provides the necessary support and basic functions for easy usage, called *EntityFrameworkCore.TemporalTables*. It has existed since 2018 and is still being constantly updated, the average downloads per day is 28 and total downloads 15,143. Using it instead of trying to implement Temporal Tables support in EF myself seems logical, because someone has already worked on creating the valuable tools in order to make the usage easier, therefore it wouldn't make sense to "reinvent the wheel" and go through the same process, possibly facing the same issues and spending time on something that already exists and is more stable as it is continuously being worked on.

### 3.3.3 **Why it was chosen**

Why Temporal Tables seems like a better approach over manually creating the soft delete approach with  DeletedAt and CreatedAt fields in existing tables is that it is an official solution to target an existing problem which is soft updating being a lot of manual work and edge-case covering, while also polluting active tables with historical data. It supports cascade delete and update which are convenient, still keeping versions of data, as well as unique constraint in 1:0-1 relationships without preventing us from adding the same soft deleted data again. Another good thing is that there is no need for composite keys, which complicate the database and tend to be used more in legacy systems than newer applications and are difficult to migrate from.

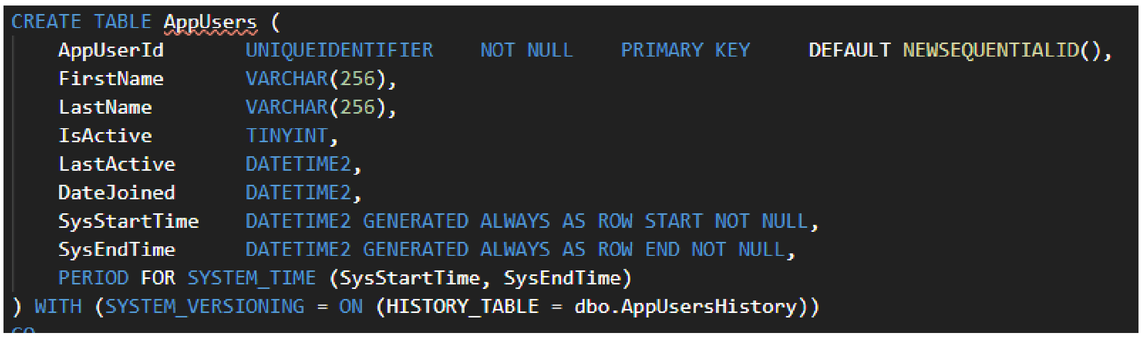
Temporal tables are trying to make soft updating more out-of-the-box solution and therefore more convenient for the user. It would make sense that the database engine itself would handle the synchronization, indexes and other necessary aspects of versioned data as it is a common and basic need for handling database data versions, therefore needed anyway. Letting the users make their own specific solutions in order to solve the same problems produces many different database designs that are error-prone and not compliant with each other, also meaning that new developers are not familiar with it. There doesn't seem to be much reason to justify "reinventing the wheel" if there could be generic official solution which is easy to learn and use and is kept robust and updated by Microsoft itself. Of course, one solution can never cover every use case and be perfect but having a solution which would be good enough for many of use cases at least is always welcomed. As of now it is not established that Temporal Tables corresponds to this criteria but it is a tool made with this intention in mind and is actively being worked on.

Considering the nature of the project at hand it seems to be the best solution for the use case.

### 3.3.4 **Usage examples**

When creating a table, SysStartTime and SysEndTime columns need to be added (can be named otherwise, but the idea should remain the same), each respectively having autopopulated values using "DATETIME2 GENERATED ALWAYS AS ROW **START** NOT NULL" and "DATETIME2 GENERATED ALWAYS AS ROW **END** NOT NULL". There should also be marked that these values are representing a period for SYSTEM_TIME by saying "PERIOD FOR SYSTEM_TIME(SysStartTime, SysEndTime). The table should end with a clause that turns on the system versioning and gives a name to the corresponding historical table. This is done via "WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = *dbo.AppUsersHistory*). These are the only exceptions when creating a table or adding data and constraints to it. Everything else is done as usual.

Example query is shown in the following picture.



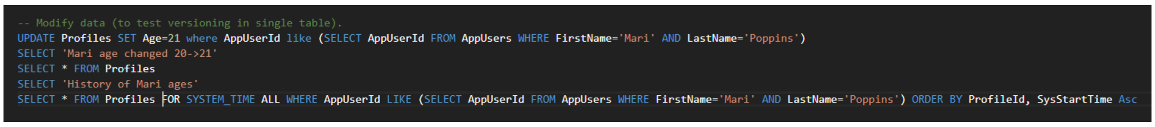
```sql
CREATE TABLE AppUsers (
    AppUserId       UNIQUEIDENTIFIER    NOT NULL    PRIMARY KEY     DEFAULT NEWSEQUENTIALID(),
    FirstName       VARCHAR(256),
    LastName        VARCHAR(256),
    IsActive        TINYINT,
    LastActive      DATETIME2,
    DateJoined      DATETIME2,
    SysStartTime    DATETIME2 GENERATED ALWAYS AS ROW START NOT NULL,
    SysEndTime      DATETIME2 GENERATED ALWAYS AS ROW END NOT NULL,
    PERIOD FOR SYSTEM_TIME (SysStartTime, SysEndTime)
) WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.AppUsersHistory))
```

Regular SQL queries, which if targeted towards the regular AppUsers table, target only the active data. Temporal queries are something that give us a possibility to target both data – active and historical. Temporal queries are queries that include a clause “FOR SYSTEM_TIME”.

Query “SELECT * FROM” Profiles would give us only the currently active data. For example Mari who has age 21, even though it was previously 20. You wouldn’t know this information based on this query only.

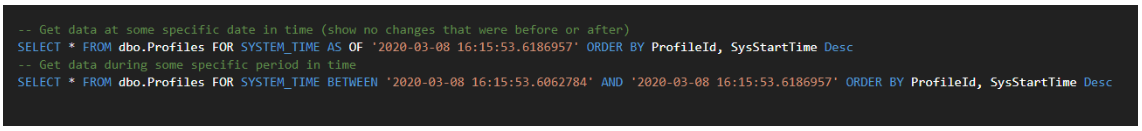
If you want to know all the ages that Mari has set on their profile, you can run a query “SELECT * FROM Profiles FOR SYSTEM_TIME ALL WHERE AppUserId LIKE ….” and this would give you multiple entries – Mari age 20 as well as Mari age 21. Based on SysEndTime it can be determined whether it is still an active data or not. Active has the date set as “9999-12-31 23:59:59.9999999”, inactive has some date in the past – the moment when it was deleted or modified. SysStartTime shows the time when the specific record became active, whether it be creation or updating.

Example SELECT queries are shown in the following picture.



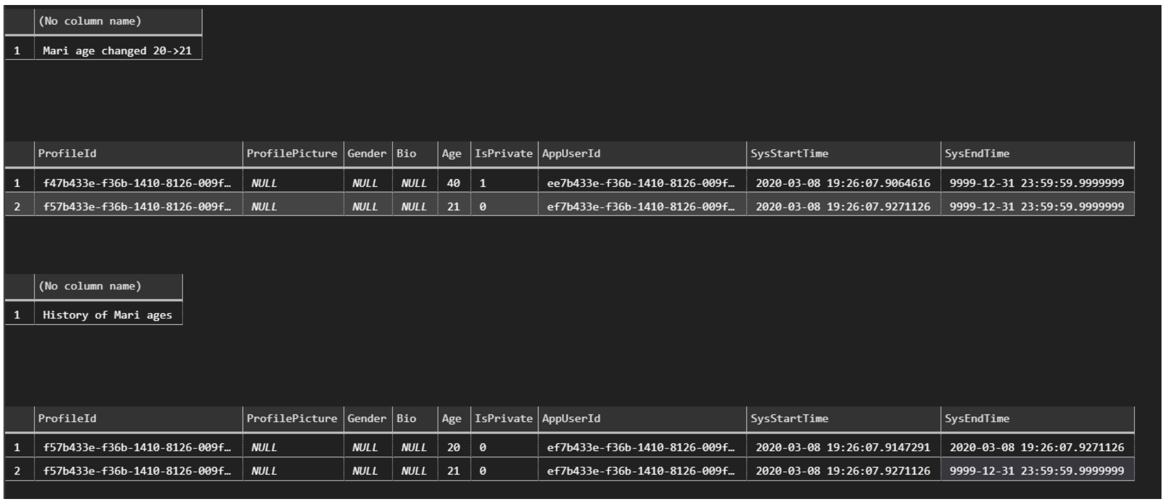
```sql
-- Modify data (to test versioning in single table).
UPDATE Profiles SET Age=21 where AppUserId like (SELECT AppUserId FROM AppUsers WHERE FirstName='Mari' AND LastName='Poppins')
SELECT 'Mari age changed 20->21'
SELECT * FROM Profiles
SELECT 'History of Mari ages'
SELECT * FROM Profiles FOR SYSTEM_TIME ALL WHERE AppUserId LIKE (SELECT AppUserId FROM AppUsers WHERE FirstName='Mari' AND LastName='Poppins') ORDER BY ProfileId, SysStartTime Asc
```

Instead of “ALL” keyword, “AS OF <date>” or “BETWEEN <date1> AND <date2>” could be used, as shown in the following picture.



```sql
-- Get data at some specific date in time (show no changes that were before or after)
SELECT * FROM dbo.Profiles FOR SYSTEM_TIME AS OF '2020-03-08 16:15:53.6186957' ORDER BY ProfileId, SysStartTime Desc
-- Get data during some specific period in time
SELECT * FROM dbo.Profiles FOR SYSTEM_TIME BETWEEN '2020-03-08 16:15:53.6062784' AND '2020-03-08 16:15:53.6186957' ORDER BY ProfileId, SysStartTime Desc
```

Example outputs of SELECT queries are shown in the following picture. The upper table shows all active data regarding profiles – there Mari is shown to have age 21. The table below shows all profile versions Mari has had – with age 20 and age 21 as well.



| | (No column name) |
|---|---|
| 1 | Mari age changed 20->21 |

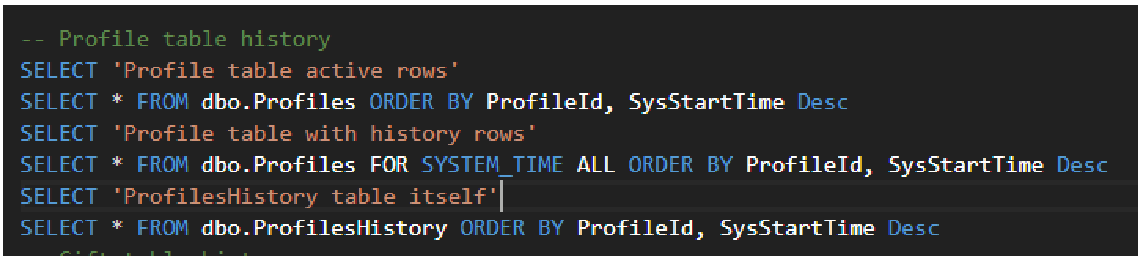
| | ProfileId | ProfilePicture | Gender | Bio | Age | IsPrivate | AppUserId | SysStartTime | SysEndTime |
|---|---|---|---|---|---|---|---|---|---|
| 1 | f47b433e-f36b-1410-8126-009f… | *NULL* | *NULL* | *NULL* | 40 | 1 | ee7b433e-f36b-1410-8126-009f… | 2020-03-08 19:26:07.9064616 | 9999-12-31 23:59:59.9999999 |
| 2 | f57b433e-f36b-1410-8126-009f… | *NULL* | *NULL* | *NULL* | 21 | 0 | ef7b433e-f36b-1410-8126-009f… | 2020-03-08 19:26:07.9271126 | 9999-12-31 23:59:59.9999999 |

| | (No column name) |
|---|---|
| 1 | History of Mari ages |

| | ProfileId | ProfilePicture | Gender | Bio | Age | IsPrivate | AppUserId | SysStartTime | SysEndTime |
|---|---|---|---|---|---|---|---|---|---|
| 1 | f57b433e-f36b-1410-8126-009f… | *NULL* | *NULL* | *NULL* | 20 | 0 | ef7b433e-f36b-1410-8126-009f… | 2020-03-08 19:26:07.9147291 | 2020-03-08 19:26:07.9271126 |
| 2 | f57b433e-f36b-1410-8126-009f… | *NULL* | *NULL* | *NULL* | 21 | 0 | ef7b433e-f36b-1410-8126-009f… | 2020-03-08 19:26:07.9271126 | 9999-12-31 23:59:59.9999999 |

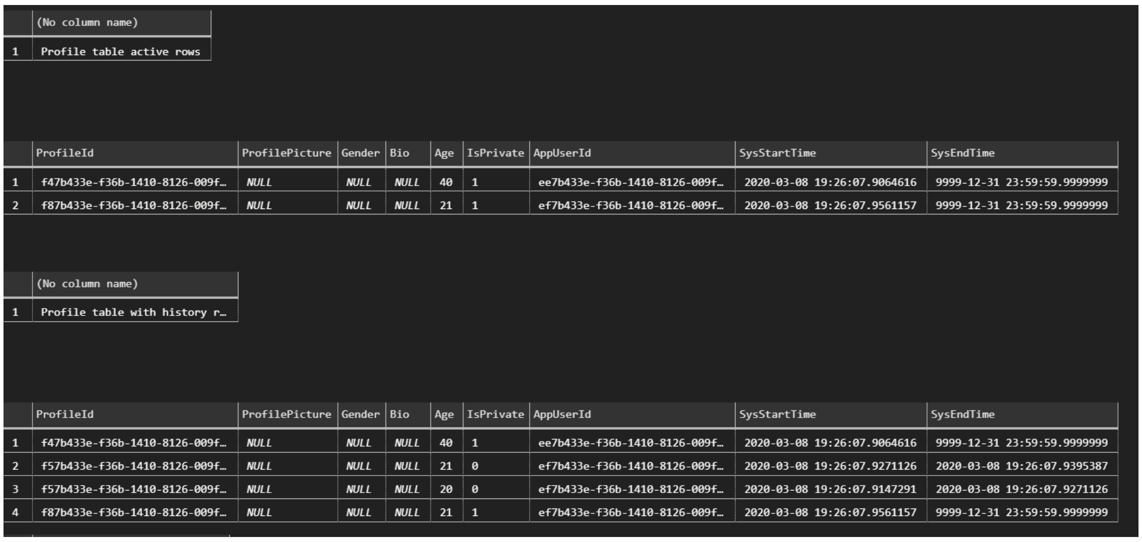
There is also a possibility to query the history table itself – this will give only the historical results, no currently active data.

Example query shown in the following picture. First one selects only the active rows, second one both active and historical together and the last one only the historical rows.



```sql
-- Profile table history
SELECT 'Profile table active rows'
SELECT * FROM dbo.Profiles ORDER BY ProfileId, SysStartTime Desc
SELECT 'Profile table with history rows'
SELECT * FROM dbo.Profiles FOR SYSTEM_TIME ALL ORDER BY ProfileId, SysStartTime Desc
SELECT 'ProfilesHistory table itself'
SELECT * FROM dbo.ProfilesHistory ORDER BY ProfileId, SysStartTime Desc
```

Example output in the following pictures.

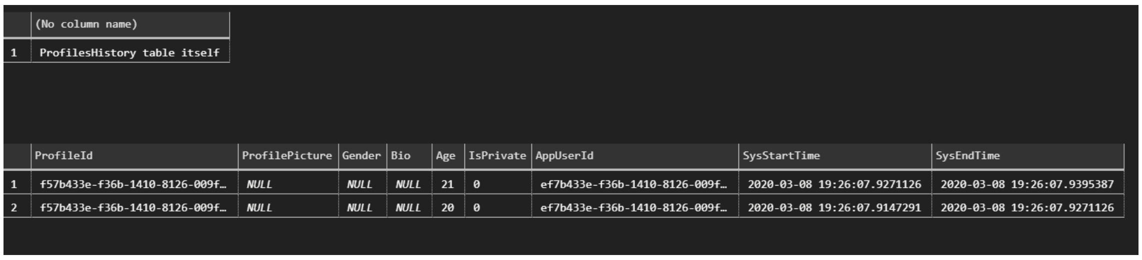


| | (No column name) |
|---|---|
| 1 | Profile table active rows |

| | ProfileId | ProfilePicture | Gender | Bio | Age | IsPrivate | AppUserId | SysStartTime | SysEndTime |
|---|---|---|---|---|---|---|---|---|---|
| 1 | f47b433e-f36b-1410-8126-009f… | *NULL* | *NULL* | *NULL* | 40 | 1 | ee7b433e-f36b-1410-8126-009f… | 2020-03-08 19:26:07.9064616 | 9999-12-31 23:59:59.9999999 |
| 2 | f87b433e-f36b-1410-8126-009f… | *NULL* | *NULL* | *NULL* | 21 | 1 | ef7b433e-f36b-1410-8126-009f… | 2020-03-08 19:26:07.9561157 | 9999-12-31 23:59:59.9999999 |

| | (No column name) |
|---|---|
| 1 | Profile table with history r… |

| | ProfileId | ProfilePicture | Gender | Bio | Age | IsPrivate | AppUserId | SysStartTime | SysEndTime |
|---|---|---|---|---|---|---|---|---|---|
| 1 | f47b433e-f36b-1410-8126-009f… | *NULL* | *NULL* | *NULL* | 40 | 1 | ee7b433e-f36b-1410-8126-009f… | 2020-03-08 19:26:07.9064616 | 9999-12-31 23:59:59.9999999 |
| 2 | f57b433e-f36b-1410-8126-009f… | *NULL* | *NULL* | *NULL* | 21 | 0 | ef7b433e-f36b-1410-8126-009f… | 2020-03-08 19:26:07.9271126 | 2020-03-08 19:26:07.9395387 |
| 3 | f57b433e-f36b-1410-8126-009f… | *NULL* | *NULL* | *NULL* | 20 | 0 | ef7b433e-f36b-1410-8126-009f… | 2020-03-08 19:26:07.9147291 | 2020-03-08 19:26:07.9271126 |
| 4 | f87b433e-f36b-1410-8126-009f… | *NULL* | *NULL* | *NULL* | 21 | 1 | ef7b433e-f36b-1410-8126-009f… | 2020-03-08 19:26:07.9561157 | 9999-12-31 23:59:59.9999999 |



| | (No column name) |
|---|---|
| 1 | ProfilesHistory table itself |

| | ProfileId | ProfilePicture | Gender | Bio | Age | IsPrivate | AppUserId | SysStartTime | SysEndTime |
|---|---|---|---|---|---|---|---|---|---|
| 1 | f57b433e-f36b-1410-8126-009f… | *NULL* | *NULL* | *NULL* | 21 | 0 | ef7b433e-f36b-1410-8126-009f… | 2020-03-08 19:26:07.9271126 | 2020-03-08 19:26:07.9395387 |
| 2 | f57b433e-f36b-1410-8126-009f… | *NULL* | *NULL* | *NULL* | 20 | 0 | ef7b433e-f36b-1410-8126-009f… | 2020-03-08 19:26:07.9147291 | 2020-03-08 19:26:07.9271126 |

### 3.3.5 **Sources**

https://www.eidias.com/blog/2018/8/29/using-sql-temporal-tables-with-entity-framework-core

https://github.com/findulov/EntityFrameworkCore.TemporalTables

https://www.nuget.org/packages/EntityFrameworkCore.TemporalTables/
https://www.nuget.org/packages/EFCoreTemporalSupport/

https://docs.microsoft.com/en-us/sql/relational-databases/tables/temporal-table-usage-scenarios?view=sql-server-ver15
https://docs.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables?view=sql-server-ver15
https://docs.microsoft.com/en-us/sql/relational-databases/tables/getting-started-with-system-versioned-temporal-tables?view=sql-server-ver15
https://docs.microsoft.com/en-us/archive/msdn-magazine/2017/march/cutting-edge-soft-updates-with-temporal-tables

# 4 Repository layer

Repository layer is one of the approaches used to hide and separate the database CRUD logic behind an abstraction. Base repository will contain general CRUD methods and application specific repositories will contain additional methods concerning our custom domain models and their required logic regarding database access.

Repository base interface is separated into a single project for the sake of separation of concerns, the same with application specific interfaces, which are in another project. There are two more projects which contain repository base implementation and application specific repository implementations respectively. Each app specific repository extends BaseRepository and implements its own application specific interface. Generics are used to support reusability and flexibility.

## 4.1 Analysis

Repository pattern is one of the patterns that could be applied to a project when trying to achieve a goal – separation of concerns. The goal is honourable as it makes the application's architecture easier to maintain, especially when there is a need to change some of the major pieces, for example the database provider itself.

Creating the application specific repositories is sensible as, of course, our custom logic cannot be provided by the framework we are using and it would not be sensible to do all the database specific logic in another layer, which should only contain business logic, either. There should, without a doubt, be some kind of layer dedicated to mediate CRUD actions against our custom database tables.

Base repository makes sense as well, because there might be a need to do some general logic that would influence the whole application, not just a singular table. An example would be applying soft deletes to the whole database, which shouldn't be done manually, multiple times in different locations, as this would go against the development principle called Don't Repeat Yourself (DRY) as well as Single-Responsibility Principle (SRP) and therefore make the maintenance difficult. It is recommended to have general logic in one place and reuse it where needed.

However, the goal itself is clear, but there are different ways to achieve this abstraction, repository pattern not being the only viable option. Another approach could be using Data Access Object (DAO).

There are different opinions on both approaches and sometimes they are even confused with each other by the members of the developer community. The problem with each solution is exactly this – it is not just one specific and robust pattern to follow – the same thing is done differently by different developers.

However, the main purpose difference between DAO and Repository seems to be that DAO is closer to the database itself and more fine-grained while Repository is closer to the Domain and kind of a foundation to Domain Driven Design (DDD). Repository could be implemented using DAOs, but the opposite wouldn't make sense. It is also recommended to keep the Repository simpler and only have Get, Find and Add methods while the responsibility of DAO regarding interaction with database is more vague. Update method would be more relevant in case of DAO while with Repository pattern the tracking of changes to entities should be done by a separate Unit of Work (UOW).

Unit of Work is the barrier between actually sending updates to database – it should include SaveChanges method, not the repository, because with more complex service doing multiple Updates and Deletes, one of them might fail and we would have no way to roll back all changes done to the database before. UOW makes sure that everything is updated all at once or nothing gets updated at all.

## 4.2 Conclusion

To summarize, both approaches – Repository pattern and DAO – have their advantages and disadvantages in certain situations. In some cases, it would not be sensible to even add another abstraction between ORM (e.g. Entity Framework), which already is an abstraction over database itself, and the application code. This applies especially when they are just wrappers with no additional logic or purpose. As for the context of this project, Repository pattern seems the preferred approach, as custom logic will probably be needed, and DAO is not as well-defined in its responsibilities and is more used in Java – it would be best to follow the conventions of the C# language community. It is also more abstract, high-level and simpler solution. Even though it is worth mentioning that while implementing it, there are some common mistakes to be aware of and to avoid – not using generics and not using UOW being a couple of the most important ones of them. It seems that many problems stated with Repository pattern tend to come from substandard implementations which are quite common, so it is especially important to analyse everything and strive for the correct approach.

## 4.3 Sources

https://www.infoworld.com/article/3117713/design-patterns-that-i-often-avoid-repository-pattern.html

https://brianbu.com/2019/09/25/the-repository-pattern-isnt-an-anti-pattern-youre-just-doing-it-wrong/

https://thinkinginobjects.com/2012/08/26/dont-use-dao-use-repository/

https://gunnarpeipman.com/ef-core-repository-unit-of-work/

https://softwareengineering.stackexchange.com/questions/313188/if-repository-pattern-is-overkill-for-modern-orms-ef-nhibernate-what-is-a-be

https://softwareengineering.stackexchange.com/questions/181202/alternatives-to-the-repository-pattern-for-encapsulating-orm-logic

https://stackoverflow.com/questions/1578778/using-iqueryable-with-linq

https://stackoverflow.com/questions/39434878/how-to-include-related-tables-in-dbset-find

# 5 Business logic layer

Business logic layer (BLL) is the layer between Repository layer and API Controllers. This is where custom, business decision specific logic, should be located. While repositories are concerned with database-specific logic such as getting, updating and deleting data based on some field's data, BLL is concerned with interactions regarding this data such as modifying it, adding things and implementing some restrictions or conditions on it based on certain circumstances. This is all done before changing something in the database after POST, PUT or DELETE request, or after getting existing data from the database after GET request, using repositories via UOW. It is the middle layer that simply put mediates and regulates everything that goes on between public API and database, making sure everything happens as expected. BLL might also have some logic that has nothing to do with database, such as calculations.

BLL contains of services and like other layers, Mappers. Mappers convert DTOs from one type to another so BLL services return BLL DTOs, not DAL DTOs from repositories. API Controllers take these BLL DTOs and covert them to Public DTOs to send to the client.

# 6 API Controllers

API Controllers are located under WebApp. These have methods that are exposed to the client and through which they can interact with the application and its database. API Controllers are calling methods from BLL layer only, never directly from DAL. These controllers are versioned and when some logic changes, new version should be created to support backwards compatibility and keep the old ones working for the client in case they need them.

API Controllers should not contain any business logic, only data validation checks and BLL method calls. They return status codes such as "200 OK" or "400 Bad Request" to the client, with or without data, depending on the method so it is clear whether the request succeeded or failed and what were the results.

# 7 Functionality / features

## 7.1 Account and site related

### 7.1.1 **Register**

Registering is done using JSON Web Token (JWT). It is a free and convenient industry standard way to represent claims securely between two parties – front-end and back-end.

To register successfully, user needs to provide their e-mail, password and full name.

During registering, default Profile will be created for the user right away. This Profile contains an empty Wishlist as it is a required foreign key. Wishlist can be empty so it is – by default it contains no gifts. In case the user was invited by another existing user, they will get a notification about them joining.

If registration is successful, user is automatically logged in. Once the user is logged in and tries to access their profile, it is already there with defaults set, so there is no need to create it by themselves. What the user can do is update it with their personal information – bio, age, gender, gifts they wish to receive etc.

Person wishing to use the functionalities of the application needs to register. All pages except registering, login and logged-out view home page are hidden from users who are not logged in. This is done by checking the presence of JWT and redirecting the person accessing it to the login page in case the token is missing.

### 7.1.2 **Login**

Logging in is done using the JWT. To log in successfully, user needs to provide the e-mail and password they used when registering. It is worth to note that currently there is no way to retrieve or reset lost credentials, so the user has to be careful not to forget.

After logging in, the user will be able to interact with all functionality available to users of the app, except adding new Campaigns unless they acquire a Campaign Manager role. This role can currently be manually given by site owner per request, there is no automatic way to get it because not everyone should be able to become one.

In the backend side, API Controllers are protected with Authorize attribute which is relying on JWT token sent via each request's header with 'Bearer'. If JWT token is missing or wrong, the user will get '401 Unauthorized' error message or be redirected to login page if this is implemented in the front end. In current front-end application it is implemented.

### 7.1.3 **Theme and overall UI**

Home page contents differ when the user is not registered or logged in, containing less specific information compared to when the user is logged in.

Choosing between different themes, especially light and dark ones, is a common feature present in most modern applications. This lets the user feel some control over the site they are using and making it appeal to them more. The theme of the current app can be toggled whether the user is registered and logged in or not, so the user can get some feel of the site even before registering. There are two basic themes to toggle between – light and dark; light being the default one. The toggler itself is in the top navigation, right side. Clicking on it will instantly switch the whole site's theme.

All pages have the main content wrapped in card-style containers and most icons are material-style.

Both themes currently have light purple/lilac as the default accent colour for certain texts, buttons and other elements. For dark theme it is a bit warmer tone and for light it is colder.

Icons and body texts for both themes are mostly light grey. Most icons, buttons and inputs have a hover effect that is a blurry accent colour shadow.

Notifications button containing a dropdown with new notifications is located next to theme toggler and is empty when there are no notifications but filled-in where there are new ones.

### 7.1.4 **Site information**

Once logged in, home page will contain a FAQ that can also be accessed via “Help” button in the top navigation, right side next to “Log out” button. Clicking this will display a dropdown with easy-access links to “My profile”, “User search”, “FAQ”, “About site” and “Contact us”. About site contains some general information and the same links under “Help” dropdown. “FAQ” leads to a frequently asked questions page which has the same information that is in the home page. “User search” leads to search page that is also accessible via “Friends” tab in the top navigation, here it is just more direct. Lastly, “My profile” is both directly accessible via top navigation and via “Help” dropdown to provide convenience choices for the user.

Each page contains some short page-specific information on the top of the page before the content-card as well.

All the previously mentioned information is not stored and queried from the database, it is directly in the front-end HTML as it depends on the client’s needs and should be easily and quickly changeable by anyone. The exceptions are the search and the profile, as the results for these are, of course, stored in the database.

## 7.2 Profile related

### 7.2.1 **Personal information**

The user can insert their personal information when navigating to “My profile” and clicking on the “Edit profile” icon next to their name.

Currently, banner picture cannot be changed. E-mail cannot be changed but can be hidden from the profile. What can be changed are username, first- and last name, age, gender, bio and profile picture. In case the user has not provided their custom profile picture, the default one will change based on their provided gender.

Providing all this information about who the user is will make it easier for people to think of gifts to give them in case they want it to be something that is not in their Wishlist.

All profile information will be requested with one request that gets the full profile – profile where Wishlist and Gifts are already included from the backend.

### 7.2.2 **Interactions with own gifts**

Adding gifts to one's Wishlist requires just clicking the "Add new gift" button on the profile. Each gift can contain information about the name of the item, description, image, url and a partner url in case it is a collaboration with some web shop such as amazon. In case it is, it should be marked as "isPartnered" as well. Another option is "Pin" which will make the gift appear higher up in the Wishlist, giving it more importance.

The only required information is actually the name, but the more information you provide, the more probable it is that people will get you exactly the right thing.

Once the gift is created and present in the Wishlist it is possible to interact with it. Gift's image is shown as a link beside other data in the list.

The options the user sees on their profile are only visible to them. Other user's see different options which are described under "6.3.5 Interactions with other's gifts" paragraph. Buttons that the user sees for their own gifts are "Pin", "Edit", "Archive", "Delete" and "Reorder". Currently "Pin" and "Reorder" have no functionality and will be added with future improvements.

## 7.3 Interactions with other users

### 7.3.1 **User search**

Searching for other users of the site is a good way to find your friends profiles. Search page contains just the input until the search query is made, then it will display the results as well.

Searching is done based on users' full names and is using AppUsersController for requesting the results. It is implemented so that even writing one letter or phrase will give results that contain it. In case there are no results for the keyword provided, the user will be presented with an error message stating that this kind of user could not be found.

Search results are displayed as a list, including the user's profile picture, full name, e-mail, last active date and an icon for navigating to their profile, even though this works by clicking on their name as well.

### 7.3.2 **Inviting friends**

In case the user tries to search for their friend's profile and discovers they are not registered in the site or already knows it then they can easily send an invitation to them.

This is done via navigating to "Friends" in the top navigation and clicking on the "My invited users" icon which reveals a page of people the user has already invited to join. Now they can click the button "Invite a friend", filling in their e-mail, phone number and message to send to them. Once they confirm it, their e-mail client opens where e-mail address, subject and body are already pre-filled and the e-mail could be sent right away. Even if they do not send the email, an entry of this invitation is already created and can be seen, edited or deleted on the "My invited users" page. There is also an indicator of whether the user has already joined the site or not. Once the user registers, it will be checked whether they were invited by an existing user or not based on their e-mail. If they were, "has joined" field will be updated correspondingly.

### 7.3.3 **Friendships**

Other user's can be added as friends when visiting their profile. "Add friend" button is located next to their name and will be changed to "Unfriend" instead after the friend request is sent.

Once the user sends a friend request, their friend will get a notification about it and see the pending request under "Pending friend requests" page. There they can either confirm or decline the request. In case they approve, the existing friendship is updated to change the field "isConfirmed" to true instead of previously set false. Requesting friend will get a notification about the approval as well and can see their new friendship under "Friends" page. However, if they decide to decline the friendship or the requester cancels the request, the relationship will be deleted from the database and could be re-added when either party decides to send a new friend request.

### 7.3.4 **Interactions with other's gifts**

The options that the user has for interacting with gifts are different when the user is on another user's profile, not their own. First the only option to interact with an active status gift is to reserve it, using the "Reserve" button next to corresponding gift. This means the user plans to gift it to the user who's Wishlist it is in. The user knows whether the gift is already reserved by someone else because then there is no "Reserve" button – only text saying "Reserved". Hovering this will show the date when it was reserved.

Once the user reserves a gift, they will see additional options. These options are icons representing "Mark as gifted" and "Cancel reservation". To better keep track of all the gifts the user has reserved and not have to navigate around in different profiles, they can go to "Reserved gifts" page from the top navigation. There is a list of all the gifts the user has reserved and the same previously mentioned interaction options for each one. Clicking on "Mark as gifted" will remove the gift from the reserved status and set it as archived instead. However, it is not fully archived until the receiver confirms it.

Under the receiver user's "Archived gifts" page they will see a list of "Pending received" gifts where they have two options – "Confirm receiving this gift" or "Deny receiving this gift". If they confirm, both users will see this gift under their "Archived gifts" page – more specifically under "Given by me" and "Received" respectively. In case they deny receiving it, the gift will be updated back to active status and be visible in their Wishlist again. The archive entry will not be kept in this case.

Gifts the user has received from other's and confirmed the receival, can be re-activated again. If this is done, archived entry will be kept, but a copy with the same data will be used to create a new active Gift and add it to the Wishlist. In the archive, all confirmed Gifts can be deleted as well – this will be permanent and cannot be undone.

From the front-end perspective, all these interactions are done via GiftController for the sake of simplicity. Front-end should not care if everything is done via one Gifts table or multiple different ones in the backend, this is an implementation detail and only concerns database and business logic. However, in the backend there are three tables – Gifts, ReservedGifts and ArchivedGifts. There is also a table for statuses and Gifts has a foreign key for status id. When Gift is changed to reserved status, new entry for ReservedGifts is created with additional information regarding reservation. This is to not bloat Gifts table with irrelevant information when not needed. The same is done when gift is changed to archived status, ArchivedGifts entry will be made with additional info – whether it is confirmed or not, date of archival etc. In case reservation is cancelled or archivedgift denied, these entries will be removed from corresponding tables and in Gifts table the entry will just have an updated status – back to active.

## 7.4 Campaigns and donates

Campaigns are so-called gifting events where users can donate gifts to those in need. Campaigns can only be created and edited by users with Campaign Manager role but seen by everyone. Currently, only the seeded admin user has this role. Registered users do not have it by default. It is something that can only be manually changed, on purpose, because being one is more of an exception than a rule and the site owner has to carefully decide who can have this role to avoid supporting scams as well.

Under “Campaigns” page, all users can see the list of existing Campaigns. There is also a button for “My campaigns”, clicking on it will direct to a page with personal campaigns, created by the user. In case they are not a Campaign Manager, it will be stated there, along with a back button. In case they are a Campaign Manager, it will be stated and there will be a button to “Create new’ and a list of existing ones.

Each Campaign can contain information about its name, description, image for the advertisement, institution connected to it and from-to dates for when it should be active. The only required data is name and dates, but as usual – the more information is provided, the better, improving the chances of users participating in it.

Related Donatees can be added for each Campaign – again, only by Campaign Manager. But everyone can see them and interact with them – similarly to interacting with friend’s gifts – marking it reserved, then gifted etc. Each Donatee is basically a mix of user profile information and gift information, together in one table.

There might not be any active Campaigns or there might be multiple. Probability for active Campaigns is probably the highest during holidays such as Christmas.

## 7.5 Friendship notifications

The user gets notification every time someone sends them a friend request or approves a friend request that they sent themselves.

Every time someone sends a friend request or approves one, backend will create a new entry in UserNotifications table, marking it as ‘active’ and including the friend’s id.

On site activation a request for getting all new personal notifications will be sent to backend, which will respond with a list of notifications in case there are any in ‘active’ status. Once the user has clicked on a certain notification, it will be marked as inactive and will not be sent as a response with the next get request.

Notification icon is in the top navigation panel, right side. It is an empty bell icon in case there are no new notifications and a filled-in bell with small lines on the sides when there are new ones. Clicking on it will reveal a dropdown with all the new notifications. Clicking on each of them will result in redirection to corresponding page – for new friend request the user will be taken to “pending friend requests” page etc.

All other notifications could be done analogously.

# 8 Future improvements

## 8.1 General refactoring

Code written is never perfect and there will always be room for improvement, especially when the time to finish the project is very limited. What needs to be kept in mind is for instance that there should be no unnecessary duplications of code. Requests from front end should make sense, no unnecessary multiple requests should be sent when the data could be sent via one request from the backend instead. DTOs should be overviewed and changed to always only include the data that is actually needed and be more consistent. Architecture of layers should be overviewed as well as there might be some logic that would better fit to for example repository rather than business logic layer or the other way around. The same is with API controllers – they should include only necessary data validation checks and business method calls. Business logic should be more consistent across all services – if something is done in one way in one place, the same approach should be used everywhere or even reused if possible. Should be looked into if services could use each other in some reasonable way to make features such as notification adding more convenient and avoid duplications even further.

Front-end especially needs refactoring since it was done more in a hurry due to the subject being more about backend functionality – front-end is done just to demonstrate that the backend works and is usable in a sensible way. Front-end itself should be refactored, duplications of code and unnecessary files removed, every "!important" keyword that was added to quickly overwrite existing CSS should be removed as it is not a good practise. Design should be responsive across the whole site.

## 8.2 Profile picture and banner changing

Currently there are hard-coded profile pictures, dependant on the choice of gender that the user has made. Profile banner photo is the same default one for everyone but in the future both of these can of course be changed by the user to fit their personal preference. Right now it was not done due to time limitations and some concerns over data storage capacity as images generally require a lot of space, therefore requiring some additional decompression features and validation to be added. Since this is not directly connected with learning how to create a layered C# Web API, it did not seem like a priority and was left out as a future possibility.

## 8.3 Bugfixes and performance enhancement

More thorough testing and debugging needed to find places where things could be improved. Need to check that duplicate database entries could not be created anywhere where it should not happen. Need to enhance queries speed and

## 8.4 Campaign ads logic

There could be implemented some logic based on which some Campaigns will be shown on the front page more frequently than others. Or that only the latest added Campaign will be shown or the one with the most/least interaction. This is a business decision that needs to be thought through.

## 8.5 Messaging feature within the app

Personal messaging is a standard feature present in all modern web applications. But this itself is like another full application within current application, requiring a lot of additional work as well as storage space. Security of the messages contents needs to be kept in mind as well. One way it could be done is with encryption. Aforementioned reasons are why it was not currently done and only possibility to send e-mails to friends is present. Of course, it could be done in the future and would be even easier if existing libraries could be reused to not “reinvent the wheel”, but this would require some research.

## 8.6 Additional and more informative notifications

Currently the user gets notification when someone adds them as a friend or accepts their friend request. There should definitely be notifications about more interactions than that – for example when someone reserves your gift (without showing who it was or what gift), claims they have gifted you something (seeing what it was and from who), when a friend the user invited joined the site, when new Campaign was added and so on. However, existing notifications already prove that the made solution works and additional

## 8.7 Additional gift related features

Currently gifts can be added, edited, reserved, archived and deleted. But there is a plan to make them reorderable as well and add a possibility to have ‘pinned gifts’ section which will always be shown before other gifts. However, right now this is not done because it would require implementing some approach of saving the order of the gifts to database and requesting them based on that order. This is probably not that difficult but would still require researching how to do it properly – what is the common best approach in such circumstances. It would also require implementing drag-and-drop in the front-end which would send PUT requests for ordering with each change. This also needs research as there are probably existing drag-and-drop libraries that could be utilized.

## 8.8 Private profile setting

Currently everyone can see everyone’s profile if they are logged in. They can also reserve gifts of users that are not friends. This should be improved by allowing the user to set their profile as private (seen only to friends) or even further, setting that everyone can see their profile, but only friends can interact with their gifts.

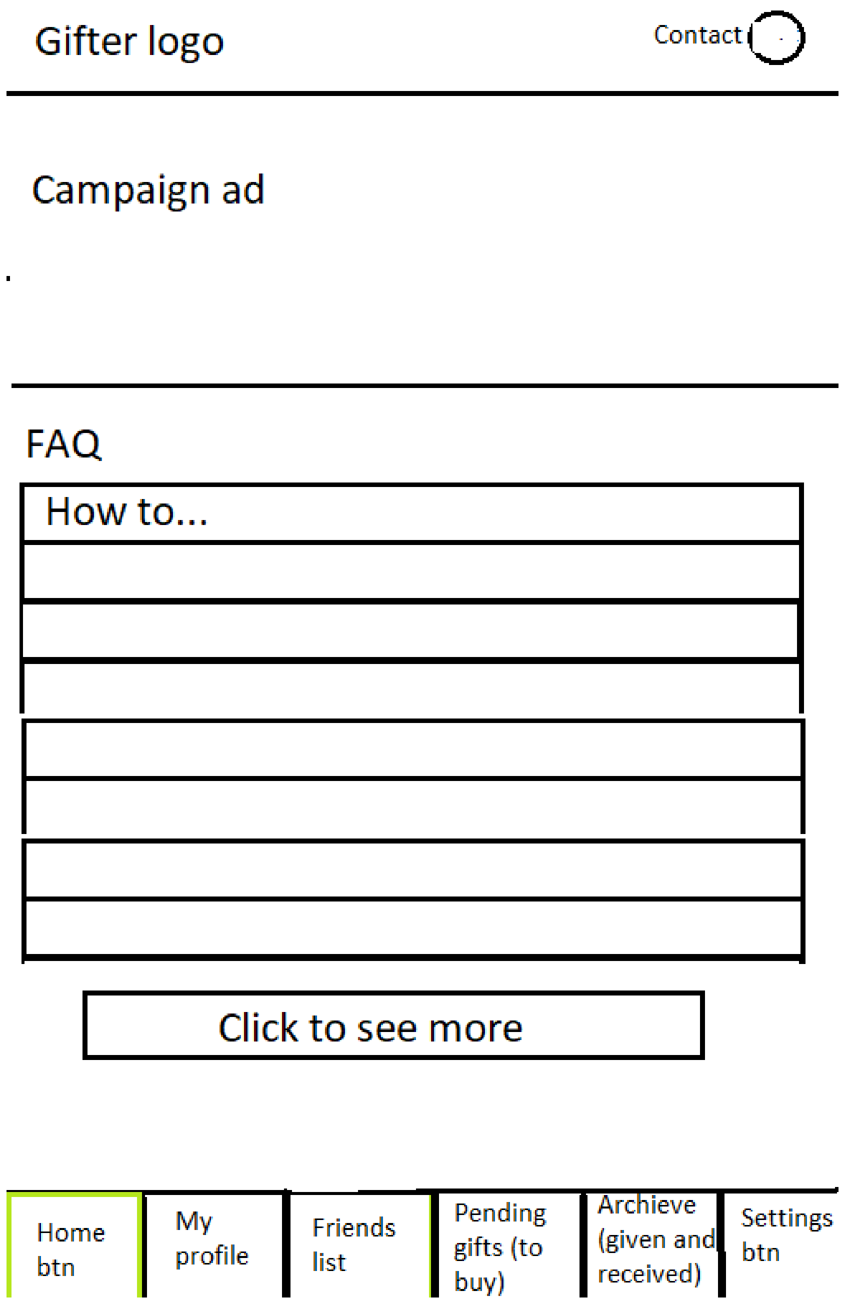
# 9 User interface designs

Proof-of-concept (POC) sketches for the user interface (UI) of the application were at first done for the mobile-view only because the approach of designing the UI of the application was presumably going to be mobile-first as it is lately the more recommended approach. However, this was not the case and Desktop view got the higher importance due to it being a Desktop application rather than a mobile app. This does not mean responsive design is not important as it will certainly be added later.

## 9.1 Initial POC design (mobile view)

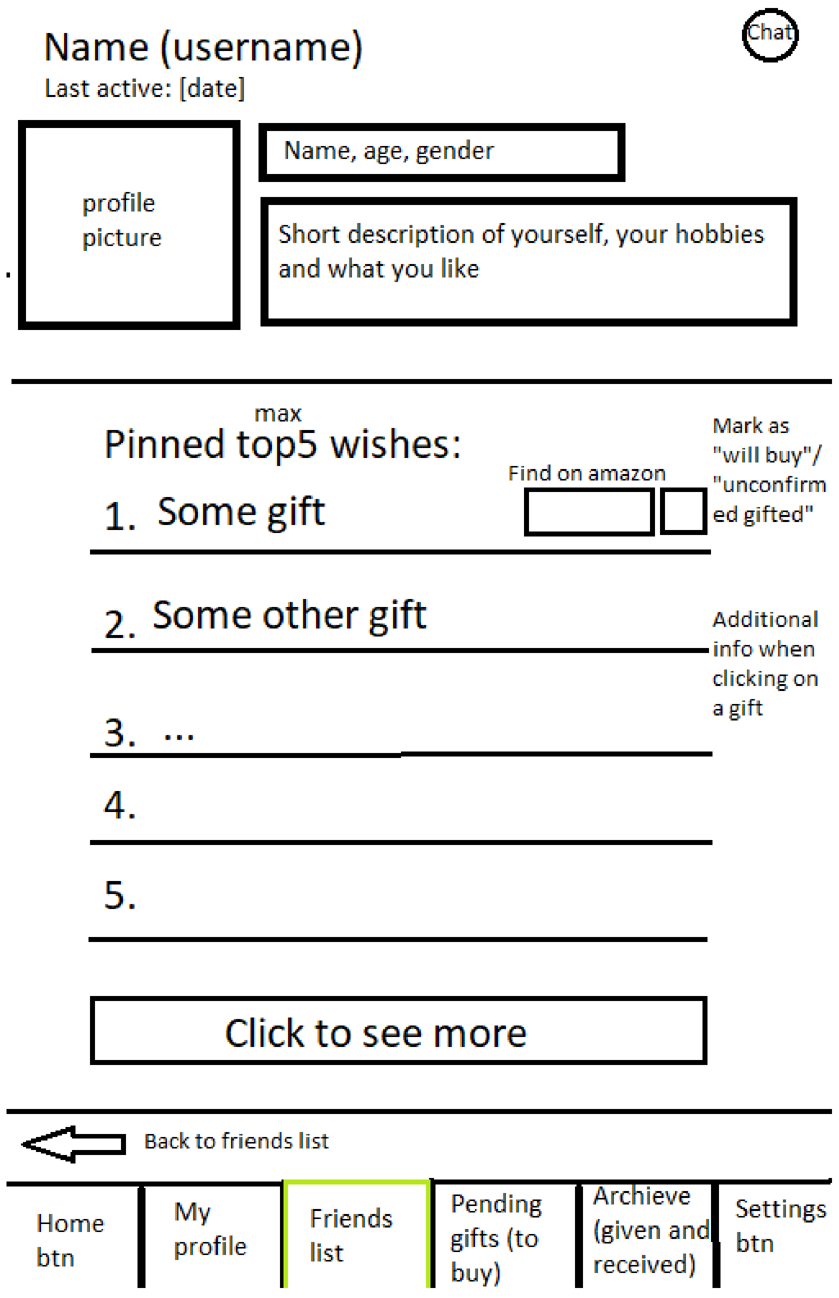
The initial idea just to visualize the goal better.

### 9.1.1 **Homepage**

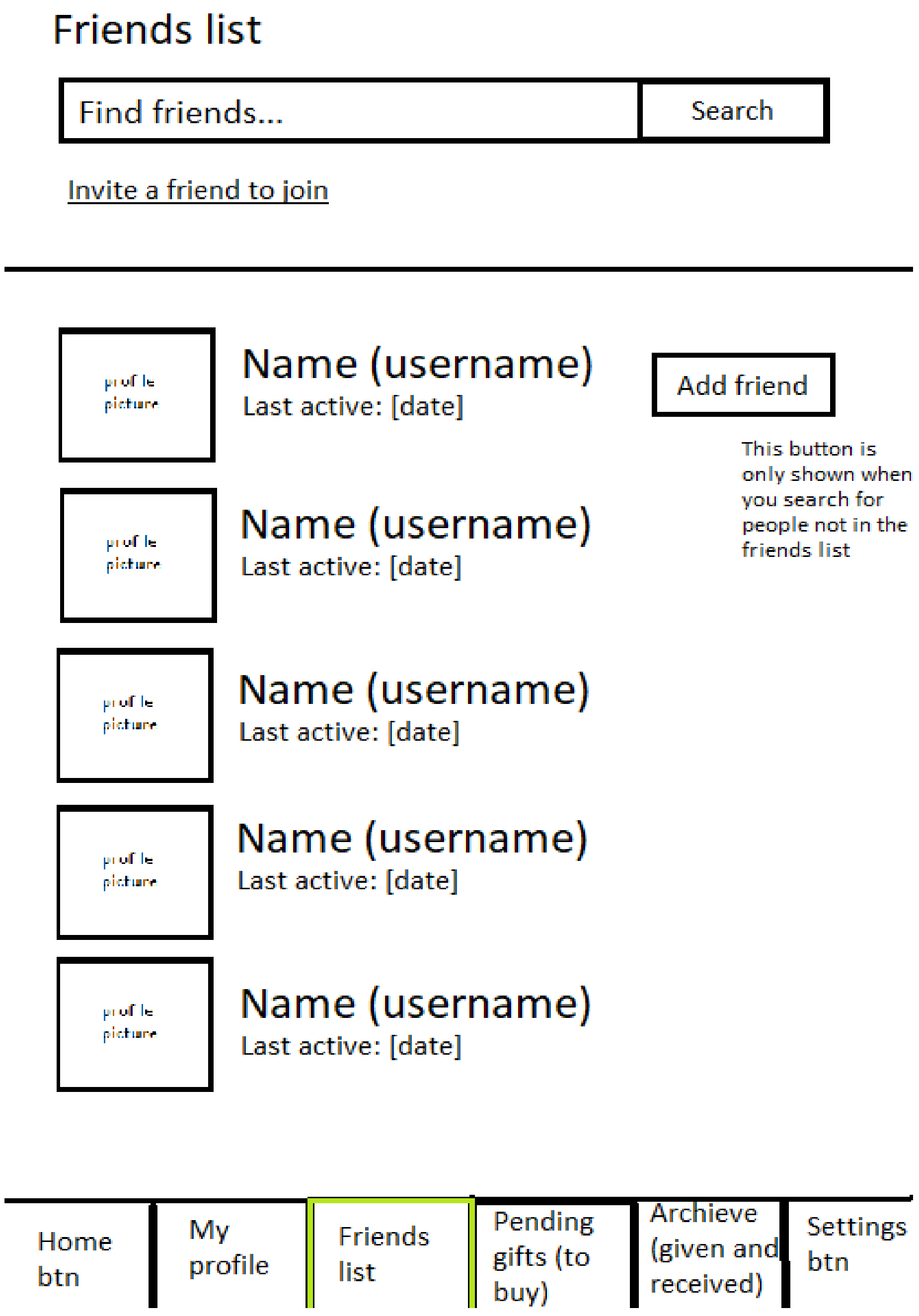


*Personal notifications might appear under ad before FAQ or before the ad.*

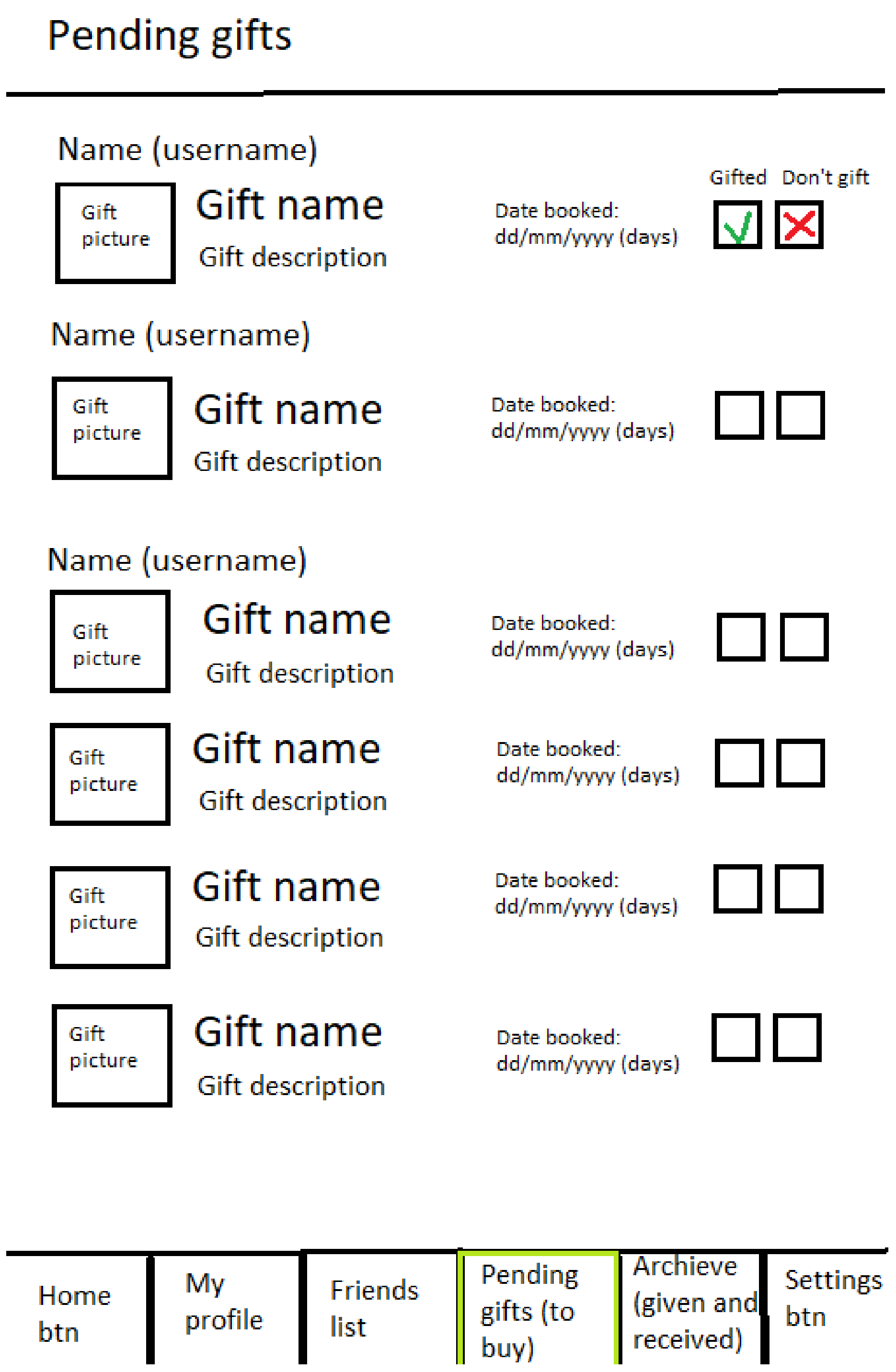
### 9.1.2 **Profile**


Name (username)
Last active: [date]
Chat
profile picture
Name, age, gender
Short description of yourself, your hobbies and what you like
max
Pinned top5 wishes:
Find on amazon
Mark as "will buy"/ "unconfirmed gifted"
1. Some gift
2. Some other gift
Additional info when clicking on a gift
3. …
4.
5.
Click to see more
Back to friends list
Home btn
My profile
Friends list
Pending gifts (to buy)
Archieve (given and received)
Settings btn

### 9.1.3 **Friends list / search**



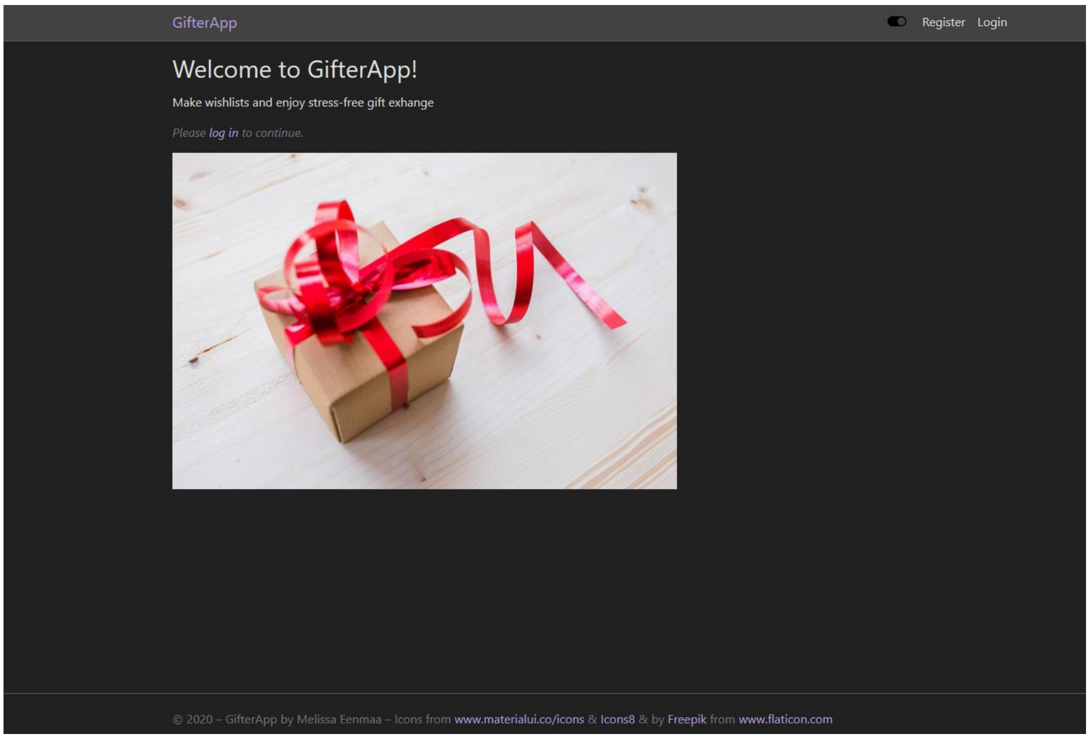
9.1.4 **Reserved gifts / archived gifts**



## 9.2 Actual resulting design (desktop view)

These are screenshots of the web page that resulted from the project. Mostly dark theme is used for the examples.

### 9.2.1 **Homepage**



GifterApp

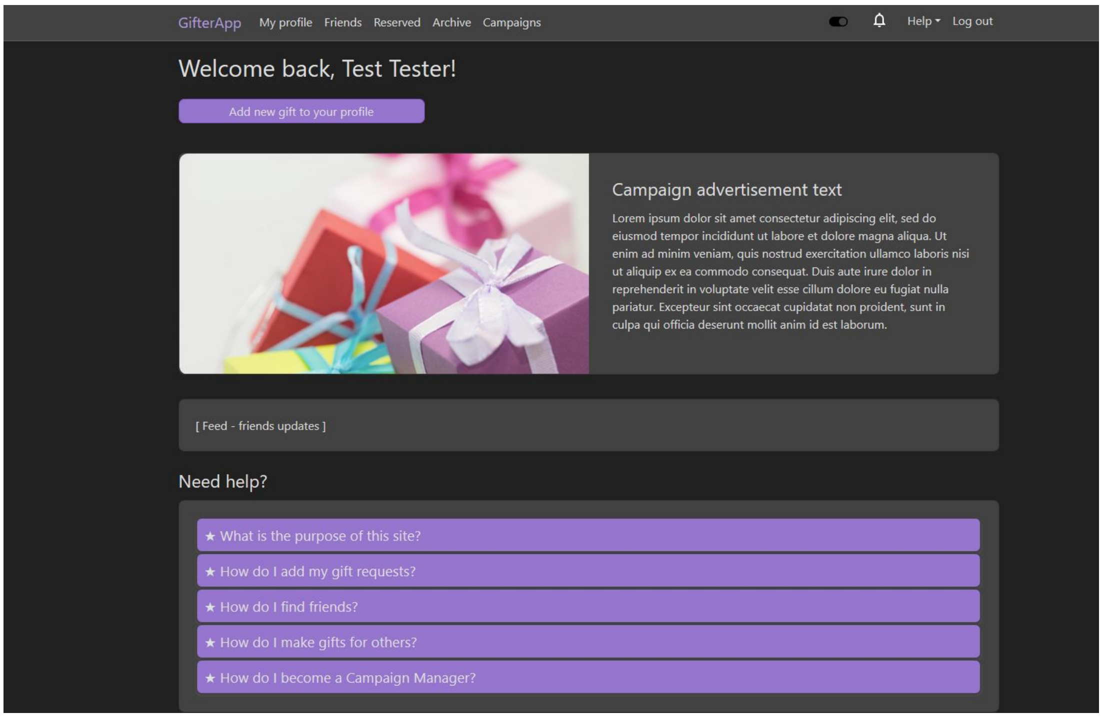
Register Login

Welcome to GifterApp!

Make wishlists and enjoy stress-free gift exhange

*Please log in to continue.*

© 2020 – GifterApp by Melissa Eenmaa – Icons from www.materialui.co/icons & Icons8 & by Freepik from www.flaticon.com



GifterApp My profile Friends Reserved Archive Campaigns

Help Log out

Welcome back, Test Tester!

Add new gift to your profile

Campaign advertisement text

Lorem ipsum dolor sit amet consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

[ Feed - friends updates ]

Need help?

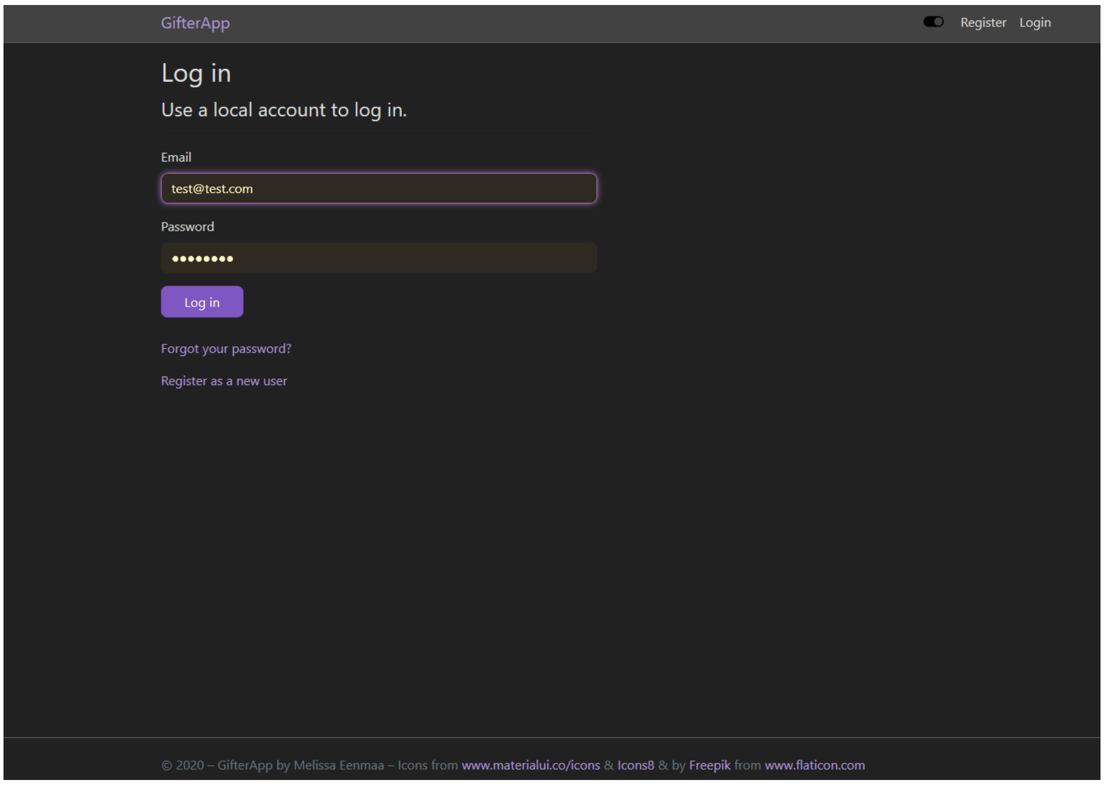
★ What is the purpose of this site?

★ How do I add my gift requests?

★ How do I find friends?

★ How do I make gifts for others?

★ How do I become a Campaign Manager?



GifterApp

Register Login

Log in

Use a local account to log in.

Email

test@test.com

Password

••••••••

Log in

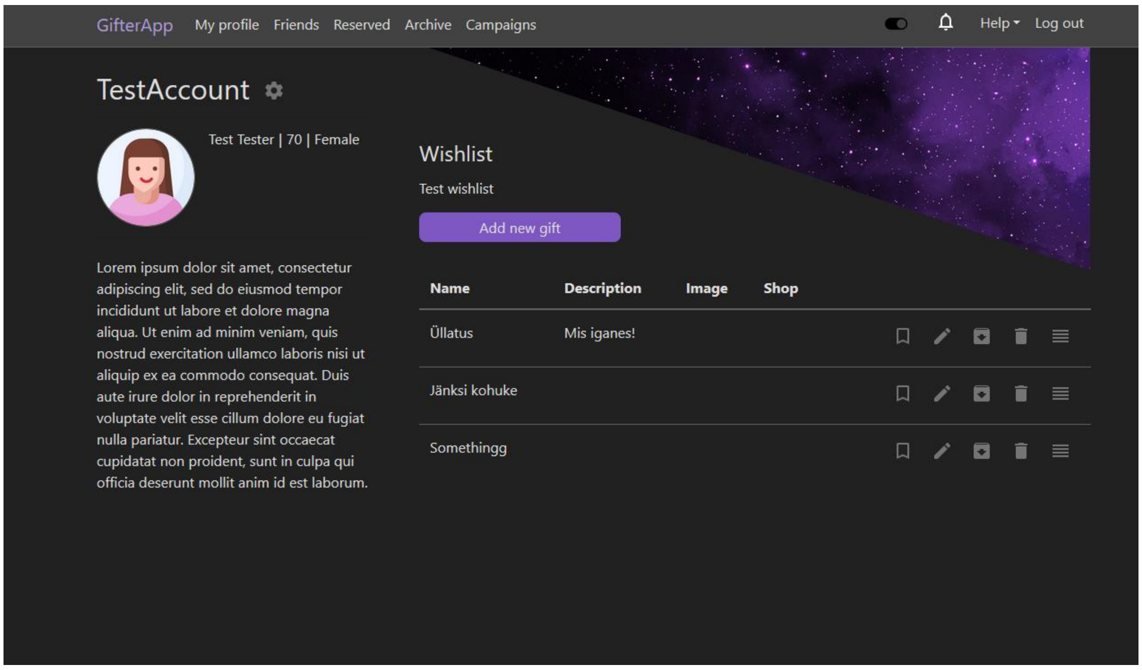
Forgot your password?

Register as a new user

© 2020 – GifterApp by Melissa Eenmaa – Icons from www.materialui.co/icons & Icons8 & by Freepik from www.flaticon.com

### 9.2.2 **Profile**

Own profile:



TestAccount

Test Tester | 70 | Female

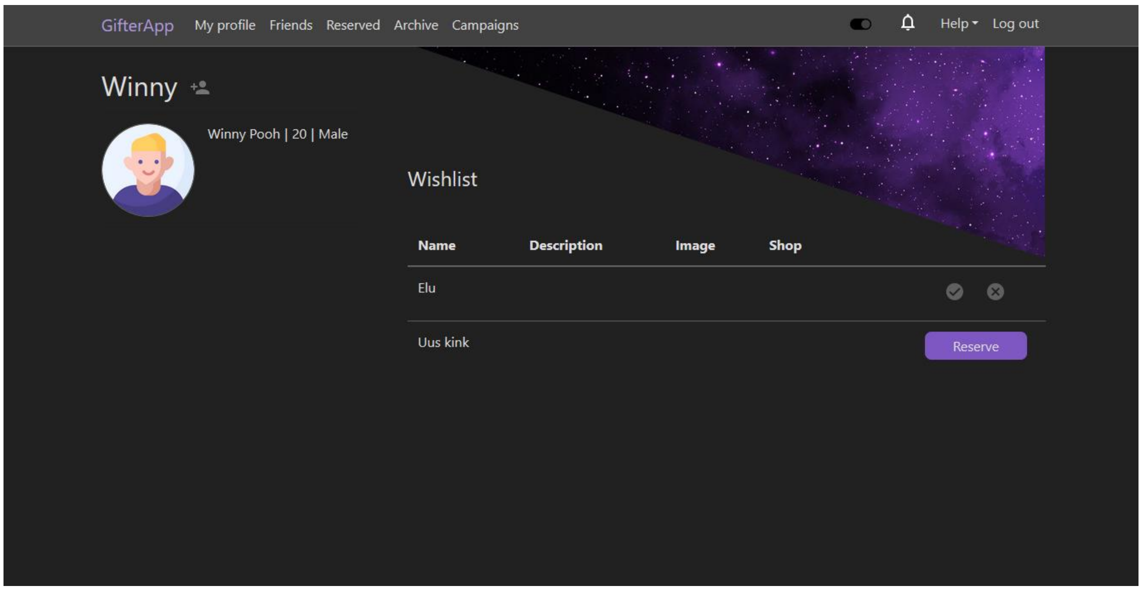
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Wishlist

Test wishlist

| Name | Description | Image | Shop |
|---|---|---|---|
| Üllatus | Mis iganes! | | |
| Jänksi kohuke | | | |
| Somethingg | | | |

Friend’s profile:



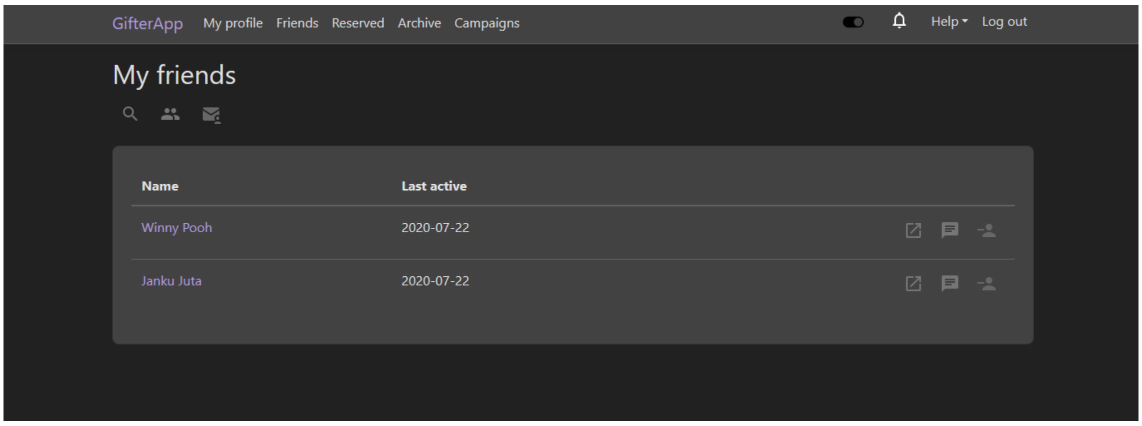
Winny

Winny Pooh | 20 | Male

Wishlist

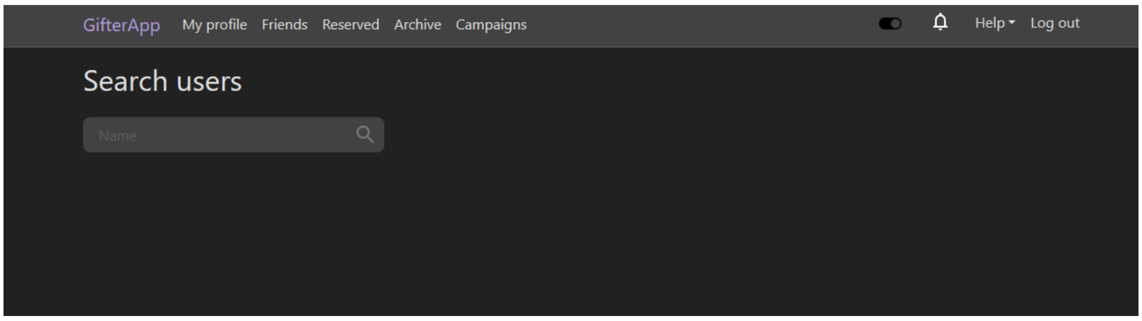
| Name | Description | Image | Shop |
|---|---|---|---|
| Elu | | | |
| Uus kink | | | |

## 9.2.3 Friends list / search

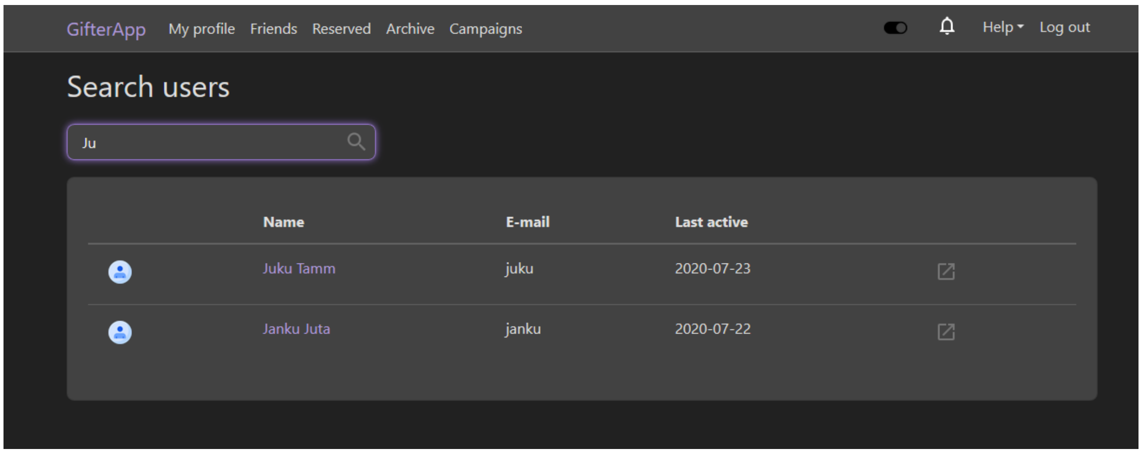


My friends

| Name | Last active |
|---|---|
| Winny Pooh | 2020-07-22 |
| Janku Juta | 2020-07-22 |



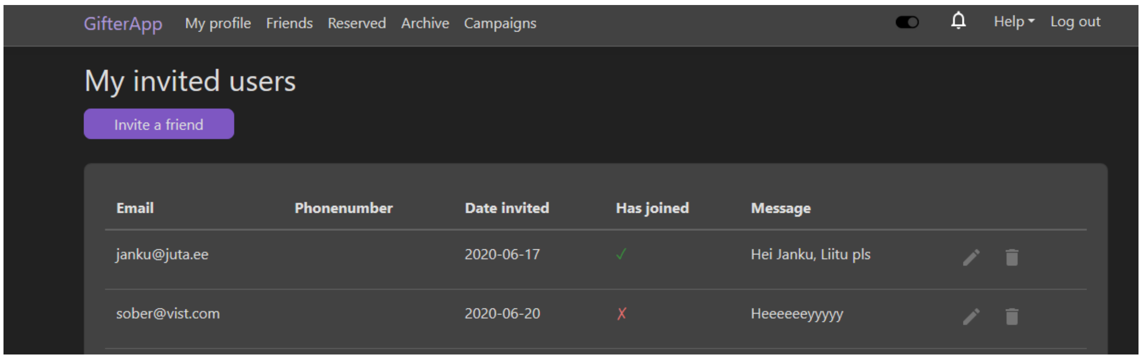
Search users



Search users

Ju

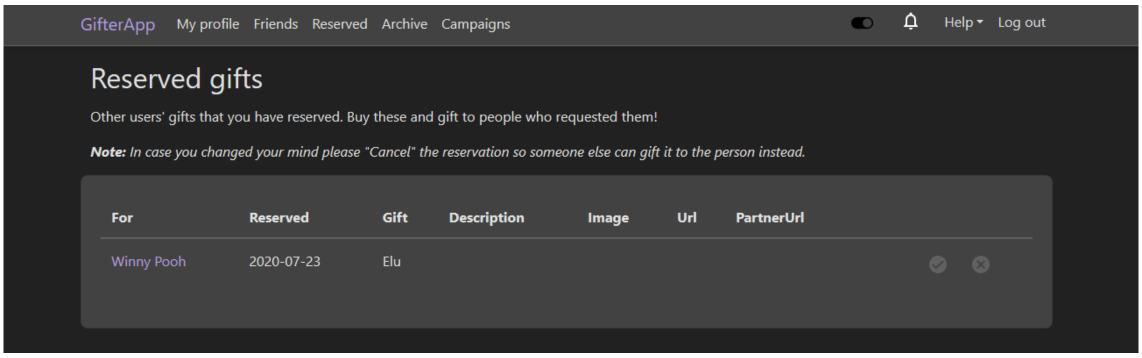
| Name | E-mail | Last active |
|---|---|---|
| Juku Tamm | juku | 2020-07-23 |
| Janku Juta | janku | 2020-07-22 |



My invited users

| Email | Phonenumber | Date invited | Has joined | Message |
|---|---|---|---|---|
| janku@juta.ee | | 2020-06-17 | ✓ | Hei Janku, Liitu pls |
| sober@vist.com | | 2020-06-20 | ✗ | Heeeeeeyyyyy |

## 9.2.4 **Reserved gifts / archived gifts**

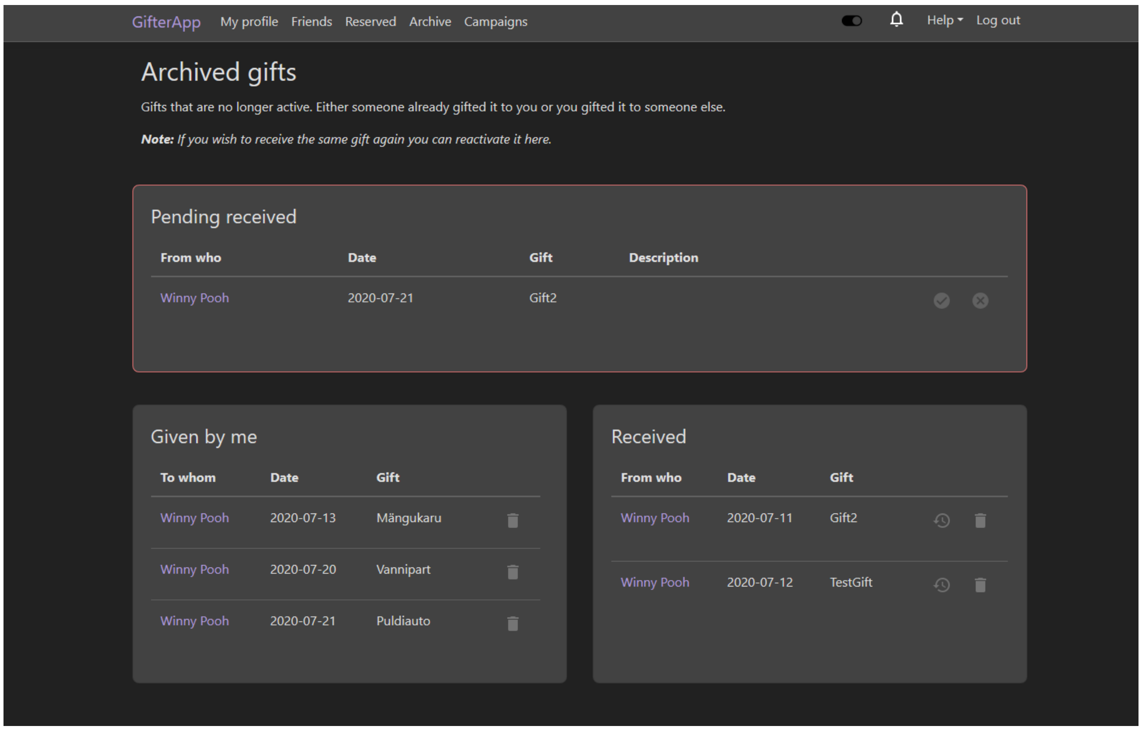


Reserved gifts

Other users' gifts that you have reserved. Buy these and gift to people who requested them!

***Note:*** *In case you changed your mind please "Cancel" the reservation so someone else can gift it to the person instead.*

| For | Reserved | Gift | Description | Image | Url | PartnerUrl |
|---|---|---|---|---|---|---|
| Winny Pooh | 2020-07-23 | Elu | | | | |



Archived gifts

Gifts that are no longer active. Either someone already gifted it to you or you gifted it to someone else.

***Note:*** *If you wish to receive the same gift again you can reactivate it here.*

Pending received

| From who | Date | Gift | Description |
|---|---|---|---|
| Winny Pooh | 2020-07-21 | Gift2 | |

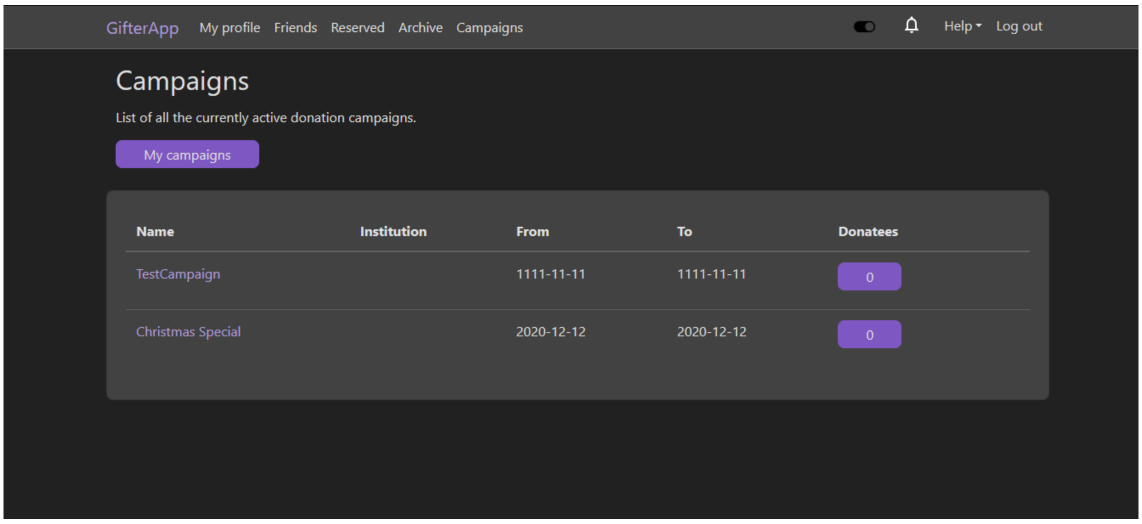
Given by me

| To whom | Date | Gift |
|---|---|---|
| Winny Pooh | 2020-07-13 | Mängukaru |
| Winny Pooh | 2020-07-20 | Vannipart |
| Winny Pooh | 2020-07-21 | Puldiauto |

Received

| From who | Date | Gift |
|---|---|---|
| Winny Pooh | 2020-07-11 | Gift2 |
| Winny Pooh | 2020-07-12 | TestGift |

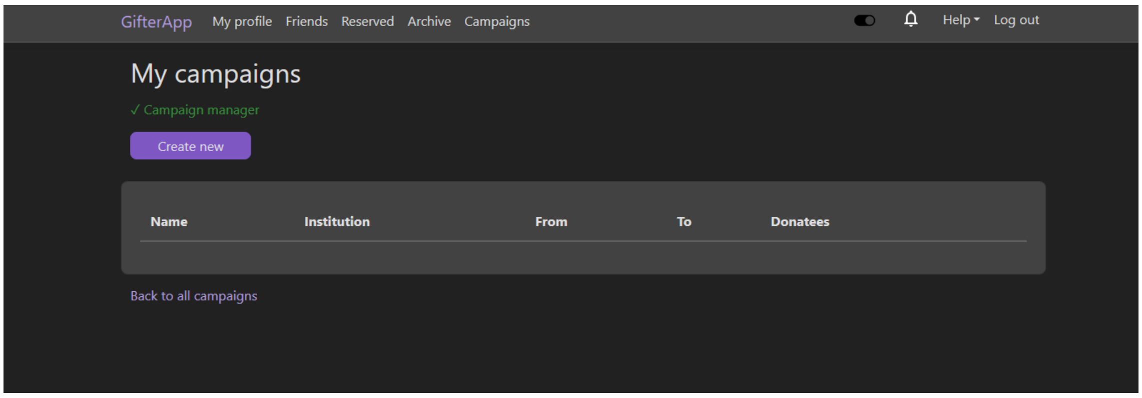
### 9.2.5 **Campaigns / donatees**



Campaigns

List of all the currently active donation campaigns.

| Name | Institution | From | To | Donatees |
|---|---|---|---|---|
| TestCampaign | | 1111-11-11 | 1111-11-11 | 0 |
| Christmas Special | | 2020-12-12 | 2020-12-12 | 0 |

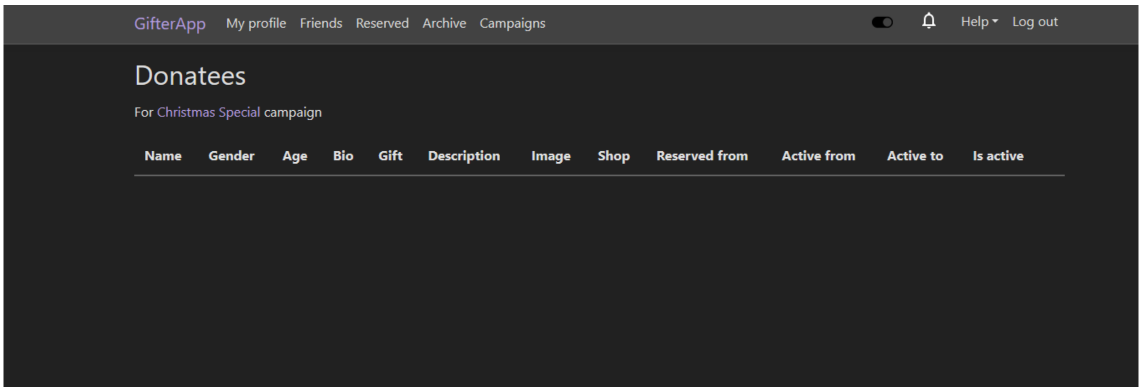


My campaigns

✓ Campaign manager

| Name | Institution | From | To | Donatees |
|---|---|---|---|---|

**TODO:** Fix My Campaigns (some bug with get request, showing 0 results)



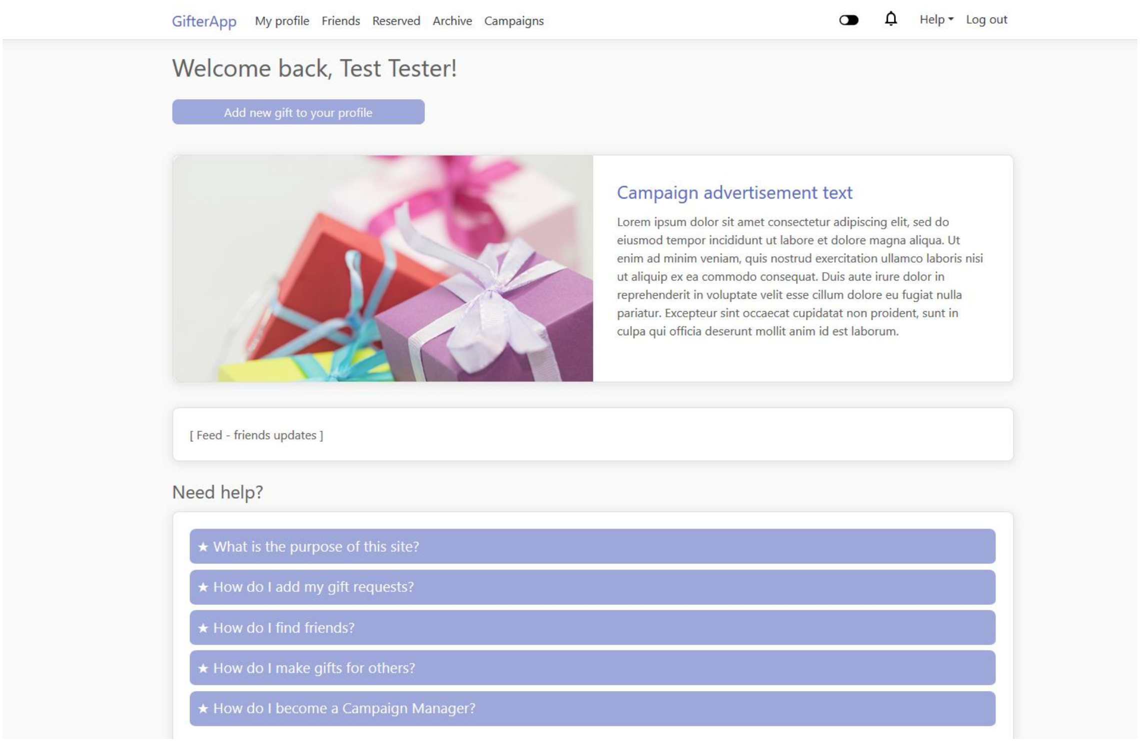
Donatees

For Christmas Special campaign

| Name | Gender | Age | Bio | Gift | Description | Image | Shop | Reserved from | Active from | Active to | Is active |
|---|---|---|---|---|---|---|---|---|---|---|---|

**TODO:** Fix Donatees (some bug with adding them, no button etc)

### 9.2.6 **Light theme example**



GifterApp My profile Friends Reserved Archive Campaigns Help Log out

Welcome back, Test Tester!

Add new gift to your profile

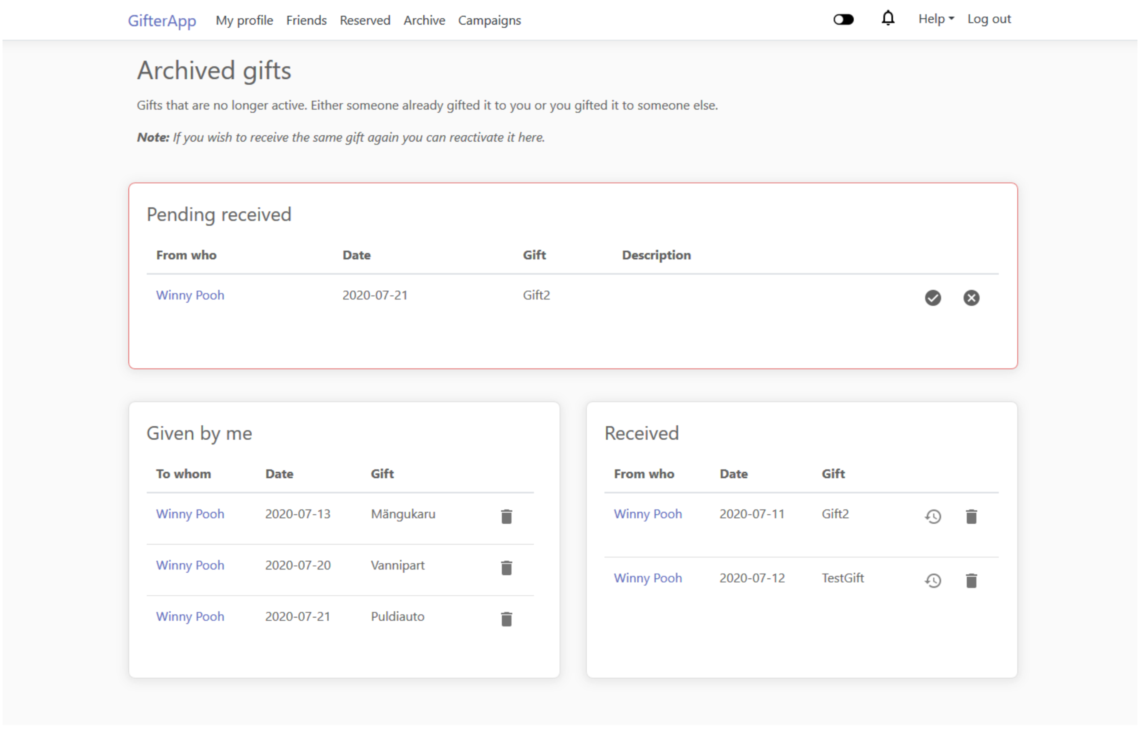
Campaign advertisement text

Lorem ipsum dolor sit amet consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

[ Feed - friends updates ]

Need help?

★ What is the purpose of this site?
★ How do I add my gift requests?
★ How do I find friends?
★ How do I make gifts for others?
★ How do I become a Campaign Manager?



GifterApp My profile Friends Reserved Archive Campaigns Help Log out

Archived gifts

Gifts that are no longer active. Either someone already gifted it to you or you gifted it to someone else.

***Note:*** *If you wish to receive the same gift again you can reactivate it here.*

Pending received

| From who | Date | Gift | Description |
|---|---|---|---|
| Winny Pooh | 2020-07-21 | Gift2 | |

Given by me

| To whom | Date | Gift |
|---|---|---|
| Winny Pooh | 2020-07-13 | Mängukaru |
| Winny Pooh | 2020-07-20 | Vannipart |
| Winny Pooh | 2020-07-21 | Puldiauto |

Received

| From who | Date | Gift |
|---|---|---|
| Winny Pooh | 2020-07-11 | Gift2 |
| Winny Pooh | 2020-07-12 | TestGift |