

Социальный сервис для построения генеалогического древа семьи на основе микросервисной архитектуры

Зацепин В.В.

ФКН ВГУ

<https://gitlab.com/ofamily/ofamily>

Воронеж 2017

**Оглавление**

[Введение 2](#_Toc499300328)

[Постановка задачи 2](#_Toc499300329)

[Анализ задачи 3](#_Toc499300330)

[Основные объекты системы 3](#_Toc499300331)

[Средства реализации 4](#_Toc499300332)

[Реализация 5](#_Toc499300333)

[Общее описание 5](#_Toc499300334)

[Запуск приложения 8](#_Toc499300335)

[Клиентское приложение 9](#_Toc499300336)

[Заключение 12](#_Toc499300337)

# Введение

Жизнь человека неразрывно связана с таким понятием как время: прошлое, настоящее и будущее. Прошлое определено для всего в нашем мире, и мы называем это историей. Есть история мира, отдельных народов, государств и т.д. История есть и у каждой семьи. Семья – это базовая ячейка общества. Каждый человек с момента рождения и до конца жизни является членом определенной семьи. История семьи – переплетение историй наших предков. Знание такой истории помогает человеку, а особенно детям, в сложном процессе самоопределения, и поиске ответов на такие вопросы, как «Откуда я родом?», «Почему я здесь?», «Что будет дальше?».

Один из способов наглядного представления истории семьи – это генеалогическое древо. **Генеалогическое древо** – это больше, чем просто набор из нескольких семейных линий, протянутых через временную шкалу. На самом деле, это история в персональном масштабе, путешествие через многие жизни, сплетенные вместе прошлым, настоящим и будущим.

Таким образом, сервис, позволяющий в определенном формате создавать генеалогические древа и сохранять информацию о предках, помог бы систематизировать накопленные знания и предоставить к ним доступ всем членам семьи в максимально удобном виде.

# Постановка задачи

Целью проекта является предоставление конечному пользователю Web сервиса (сайта) для построения генеалогического древа семьи с возможностью указания некоторой информации о членах семьи.

В рамках данного сервиса необходимо реализовать следующе требования:

* Возможность явного отображения древа в виде графа, где узлами являются члены семьи, а ребра – родственные связи между ними;
* Возможность редактировать граф:
  + Создавать и удалять узлы;
  + Создавать и удалять связи;
* Возможность указать для каждого члена семьи:
  + ФИО;
  + Дату рождения;
  + Фотографию;
  + Биографию;
* Возможность предоставления информации о генеалогическом дереве другим пользователям сервиса

# Анализ задачи

## Основные объекты системы

Для реализации системы на всех уровнях: от базы данных до клиентского приложения необходимо определить основные объекты системы, их свойства и требования к ним.

* Пользователь
  + Доступ к системе имеет только зарегистрированный пользователь
  + Для пользователя определяются
    - Имя
    - Фамилия
    - Пол
    - Адрес электронной почты
    - Пароль
  + Генеалогическое древо (далее просто дерево)
    - Для дерева определяется название
    - Включает одного или более пользователей сервиса
    - Включает ни одного или более членов семьи
  + Член семьи
    - Является узлом дерева
    - Явно не связан с ни с каким пользователем системы
    - Для члена семьи определяется:
      * Имя и фамилия
      * Пол
      * Дата рождения
      * Фотография
      * Дополнительная фотография
      * Биография
      * Родители



Рисунок 1 - ER диаграмма приложения

Определив основные объекты системы возможно построение диаграммы использования приложения.



Рисунок 2 – диаграмма вариантов использования

## Средства реализации

Для реализации приложения была выбрана микросервисная архитектура. В качестве основного серверного языка программирования был выбран Golang (GO) и фреймворк Gin, а в качестве дополнительного – Kotlin и фреймворк Spring WebFlux, основанного на реактивном программирование и неблокирующем сервере Netty (non-blocking IO). В качестве организации доступа к данным был выбран REST подход. Для хранения данных была использована база MongoDB, а так же сетевое журналирующее хранилище данных типа «ключ-значение» Redis. Задача взаимодействия микросервисом между собой была решена при помощи HTTP и брокера сообщений RabbitMQ. Клиентская часть приложения (webapp) реализована на ECMAScript 6 и фреймворке ReactJS c использованием модели состония Redux, а так же библиотеки RxJS для использования реактивного подхода в программировании. Для запуска и установки всего приложения использовались сервисы GitlabCI, DigitalOcean (PaaS) и технология виртуализации Docker. Полный список технологий:

* Go
* Go Gin
* Kotlin
* Spring Boot
* Spring WebFlux
* Maven
* MongoDB
* Redis
* RabbitMQ
* ECMAScript 6
* ReactJS
* Redux
* RxJs
* MaterialUI
* Vis.js
* Webpack
* Nginx
* Docker
* Docker compose

# Реализация

## Общее описание

На начальном этапе реализации были выделены основные категории компонентов:

* Общие компоненты системы
  + Common-go – библиотека на языке программирования Golang для реализации основных инфраструктурных требований ко всем микросервисам системы
  + Common-env – общие сторонние middleware компоненты (RabbitMQ)
* Инфраструктурные микросервисы
  + Service-discovery (Golang)– микросервис, реализующий паттерн service registry
  + Service-gateway (Golang) – прокси сервер, предоставляющий единую HTTP точку доступа ко всей распределенной системе
* Микросервисы, реализующие бизнесс логику
  + Service-account – микросервис отвечающий за регистрацию пользователей (signup) и их вход в приложение (login)
  + Service-tree – микросервис для работы с сущностями генеалогического древа и членами семьи
  + Service-static – микросервис для загрузки и получения статичных файлов (фотографии)
  + Service-webapp – клиентское приложение



Рисунок 3 – общая схема взаимодействия компонентов

Центральное место во всей системе занимает service-discovery, который, используя хранилище Redis, предоставляет точку доступа (/endpoints) для сохранения и получения информации о всех других компонентах системы. Каждый микросервис (tree, account, static) отправляет запрос на сохранение информации о себе в формате JSON каждые 10 секунд. Структура сообщения состоит из названия сервиса и его IP адреса, по которому доступен микросервис. Service-discovery в свою очередь сохраняет такие сообщения в Redis используя TTL (time to live) в 60 секунд. Так же, service-discovery предоставляет возможность получить все зарегистрированные в нем микросервисы.

Такой механизм необходим в рамках распределенной системы, чтобы поддержать взаимодействие компонентов между собой по протоколу HTTP. В рамках библиотеки common-go были разработаны общие модули для поддержки функциональности таких механизмов, как Client-side Discovery Registry и Client-side Load Balancer. Оба эти механизма использует service-gateway, для того, чтобы обеспечивать перенаправление всех запросов на соответствующие микросервисы. Service-gateway является единой точкой доступа ко всему приложение для service-webapp.



Рисунок 4 – диаграмма последовательности работы компонентов при обслуживании HTTP запроса от service-webapp.

Для наблюдения за системой и ее поведением в целом, была разработана web-страница service-gateway, которая предоставляет информацию о всей топологии микросервисов и задержки доступа к ним.

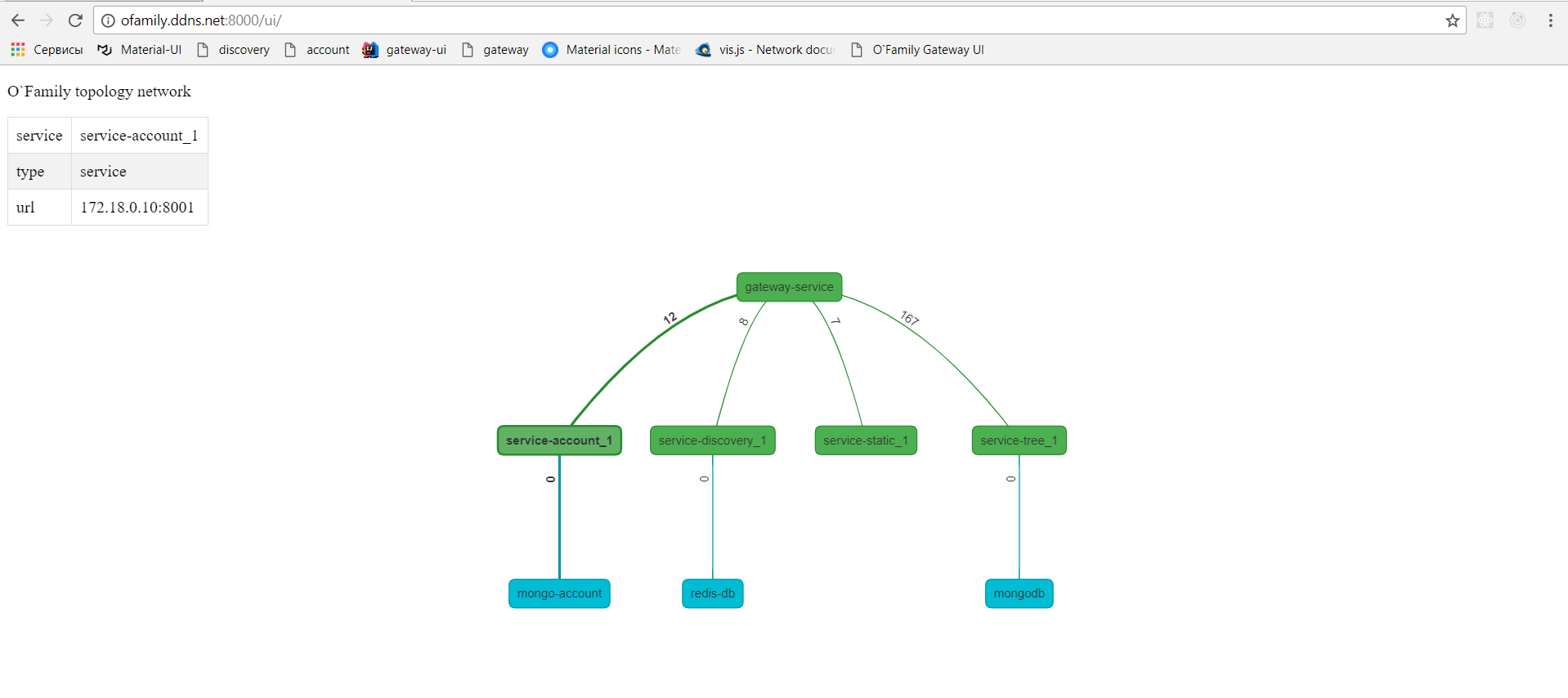


Рисунок 5 – web-страницы service-gateway

RabbitMQ используется для того, чтобы отправить оповещение из service-account в service-tree о том, что новый пользователь был зарегистрирован в системе, который в свою очередь создает дерево и начальный узел в этом дереве, соответствующий зарегистрированному аккаунту.

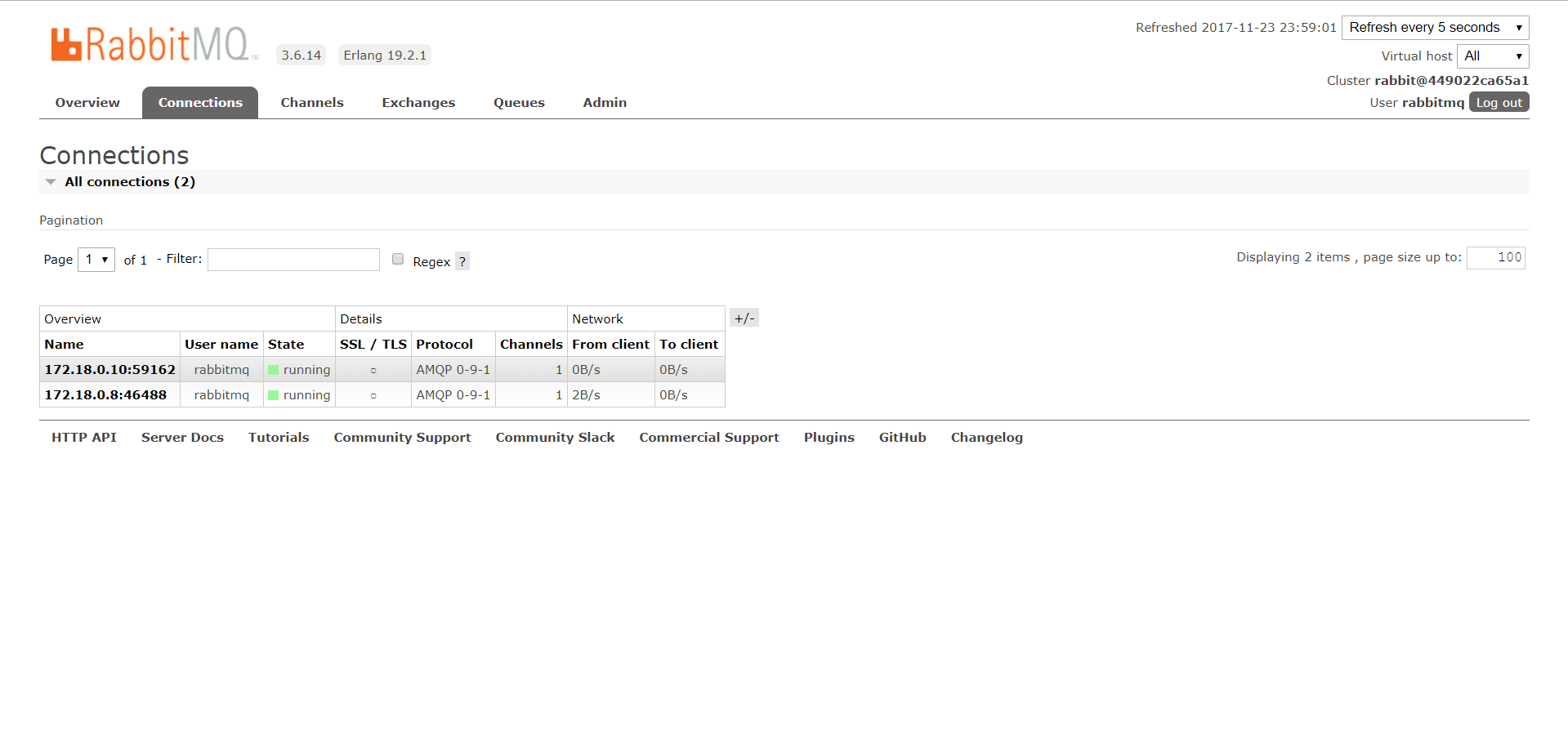


Рисунок 6 – web-страница менеджмента RabbitMQ

Объектная модель приложения строится на документах, хранящихся в двух базах данных MongoDB.

## Запуск приложения

Запуск каждого из микросервисов возможен в двух основных вариантах: прямой запуск приложения (exe, jar, nodejs) и запуск Docker Image. Первый способ используется для локальной разработке и доступу к процессу откладки, а второй - для разворачивании всей системы на удаленном сервере при помощи технологии docker-compsoe. Такой подход позволяет избежать необходимости установки установки на сервере дополнительных объектов системы, составляющие рабочее окружение приожения (MongoDB, Radis, RabbitMQ).



Рисунок 7 – CI процесс

Для связи приложения в одну сеть используется технология docker-compose. Применение такого понятия, как «COMPOSE\_PROJECT\_NAME», позволило достичь независимости разворачивания компонентов между собой и разделить сборку всего приложения на отдельные процессы сборок каждого из микросервисов. Проще говоря, например, для перезапуска service-webapp на удаленном сервере не требуется, перезапуск других микросервисов.



Рисунок 8 – Зависимость компонентов между собой в контексте разделения их на разные файлы конфигураций docker-compose.yaml

## Клиентское приложение

Клиентское приложение (web страница) была разработана в рамках микросервиса service-webapp при помощи фреймворка ReactJS и визуальных компонентов Material Design. Для локальной разработки использовался сервер NodeJS Express, а для установки на удаленном сервере – nginx с настроенным перенаправлением (proxy server) на service-gateway.

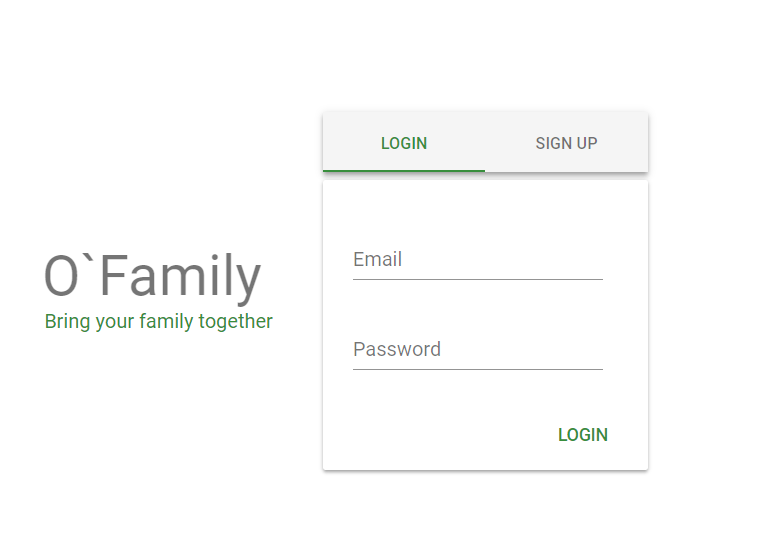


Рисунок 9 – Страница входа в приложения

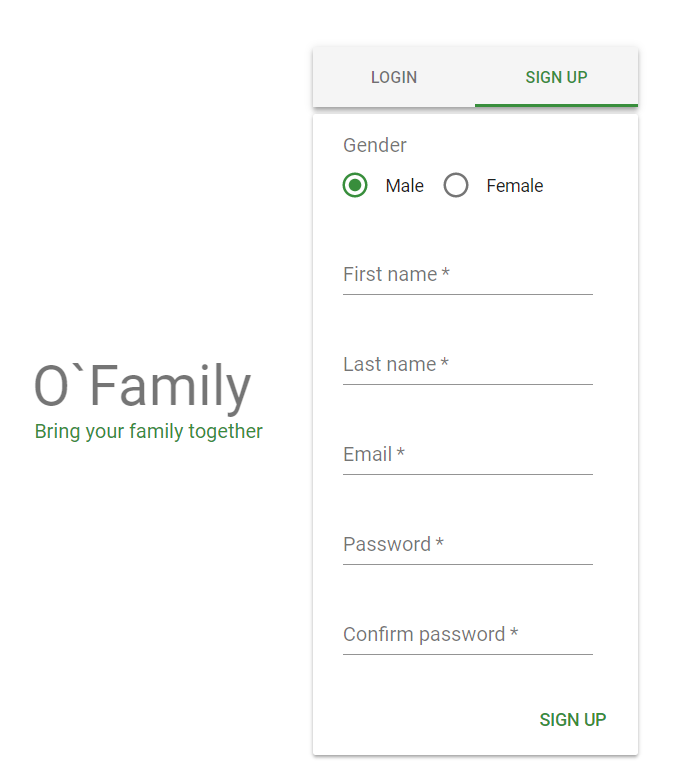


Рисунок 10 – Страница регистрации

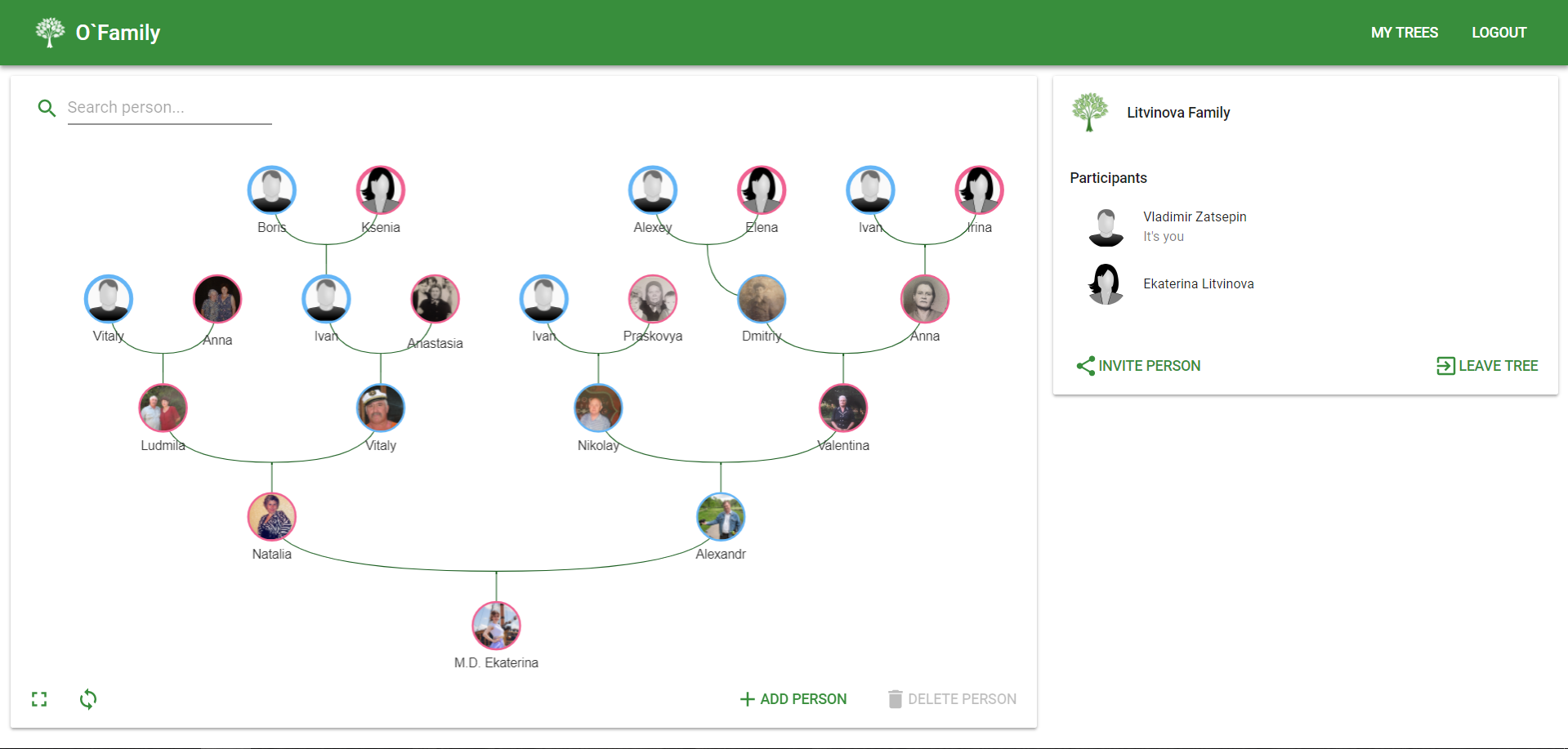


Рисунок 11 – Главная страница

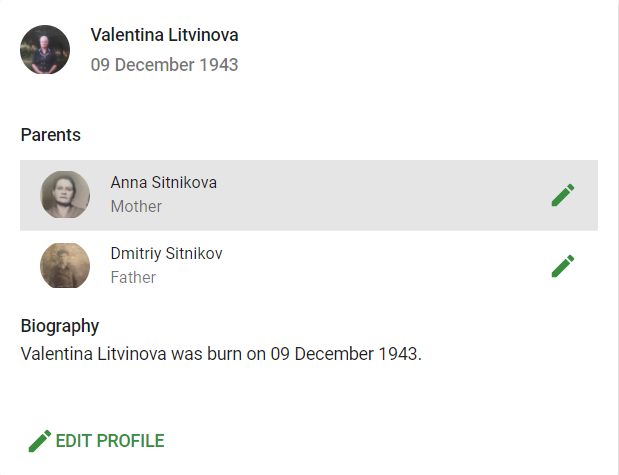


Рисунок 12 – Информация о члене семьи

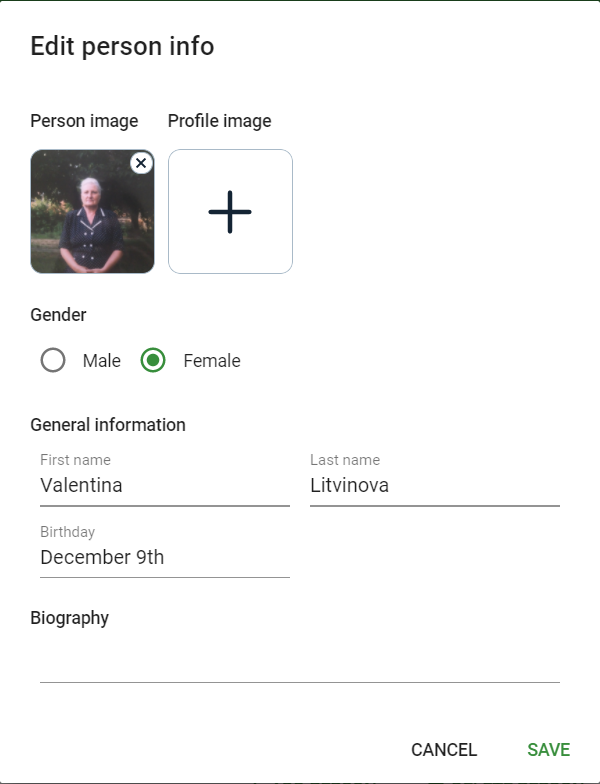


Рисунок 13 – Элемент редактирования информации о члене семьи

# Заключение

В рамках данной работы был разработан сервис для построения генеалогического древа семьи с возможностью указания некоторой информации о членах семьи на основе микросервисной архитектуры.