**API-Service выполняет роль API-Gateway , обеспечивая маршрутизацию запросов пользователя на соответствующие микросервисы. Сервис также отвечает за авторизацию, добавление заголовков и предварительную обработку запросов.  
Технологический стек  
 • Gin/REST API/gRPC : Используется фреймворк Gin для обработки HTTP-запросов, а также поддержка gRPC для взаимодействия с сервисом SessionManagement.  
 • Middleware : Реализованы промежуточные обработчики (middlewares) для выполнения различных задач:  
 ◦ Logging : Добавление уникального traceID и таймаута в контекст запроса для отслеживания и мониторинга.  
 ◦ Rate-Limiter : Ограничение количества запросов с одного IP-адреса с использованием механизма очистки неактивных записей через фоновую горутину.  
 ◦ Authorization : Проверка валидности текущей сессии пользователя посредством gRPC-запроса к микросервису SessionManagement . Реализована retry-логика для повторных попыток в случае временных ошибок сервиса. Ошибки разделены на клиентские (например, невалидный токен) и серверные (например, недоступность сервиса). В случае успеха добавляются данные userID и sessionID в контекст запроса для последующего использования.  
 • Timeout Handling : Проверка истечения таймаута контекста. Если таймаут истекает, возвращается ошибка 500 Internal Server Error с соответствующим сообщением клиенту.  
 • Используется log из стандартной библиотеки для логирования  
 • Файлы конфигурации загружаются из config.yml с помощью библиотеки viper  
 • Проект разделен на пакеты и представляет отдельный модуль  
Проксирование HTTP-запросов  
Реализован механизм проксирования HTTP-запросов на целевые микросервисы с добавлением необходимых заголовков и данных из контекста. Для каждого эндпоинта определена своя логика:  
 1. Запрос на /api/reg :  
 ◦ Проксирование запроса пользователя на микросервис UserManagement (/reg).  
 ◦ Добавление traceID из контекста в заголовки запроса.  
 ◦ Возврат ответа от микросервиса клиенту.  
 2. Запрос на /api/auth :  
 ◦ Проксирование запроса пользователя на микросервис UserManagement (/auth).  
 ◦ Добавление traceID из контекста в заголовки запроса.  
 ◦ Возврат ответа от микросервиса клиенту.  
 3. Запрос на /api/logout :  
 ◦ Проксирование запроса пользователя на микросервис UserManagement (/logout).  
 ◦ Добавление traceID, userID и sessionID из контекста в заголовки запроса.  
 ◦ Возврат ответа от микросервиса клиенту.  
 4. Запрос на /api/delete :  
 ◦ Проксирование запроса пользователя на микросервис UserManagement (/delete).  
 ◦ Добавление traceID, userID и sessionID из контекста в заголовки запроса.  
 ◦ Возврат ответа от микросервиса клиенту.  
Примечание: Для эндпоинтов, доступных только зарегистрированным пользователям (/api/logout, /api/delete), данные userID и sessionID добавляются в контекст после успешной авторизации. Для публичных эндпоинтов (/api/reg, /api/auth) эти данные не добавляются, так как они не требуются.   
Graceful Shutdown  
Реализована поддержка Graceful Shutdown для корректного завершения работы сервера и фоновых горутин:  
 • При получении сигнала завершения (SIGTERM или SIGINT) сервер завершает обработку текущих запросов и останавливает все активные горутины.  
 • Это гарантирует отсутствие потери данных и корректное завершение всех операций.  
Документация  
Реализована документация API с использованием Swagger для описания всех доступных эндпоинтов, их параметров и возможных ответов. Это упрощает интеграцию и тестирование сервиса.  
  
  
  
UserManagement отвечает за управление пользователями, включая регистрацию, аутентификацию, выход из аккаунта и удаление аккаунта. Сервис обеспечивает взаимодействие с базой данных PostgreSQL и микросервисом SessionManagement через gRPC, а также предоставляет REST API для клиентов.  
Технологический стек  
 • REST API/gRPC : Используется фреймворк Gin для обработки HTTP-запросов, а также поддержка gRPC для взаимодействия с микросервисом SessionManagement .  
 • Middleware :  
 ◦ Logging : Извлечение уникального traceID из заголовков запроса и добавление его в контекст для логирования. Создание таймаута для мониторинга выполнения запроса.  
 ◦ Authorization : Извлечение данных userID и sessionID из заголовков запроса для эндпоинтов /logout и /delete. Если данные отсутствуют или невалидны, возвращается ошибка 400 Bad Request. Попытка передачи этих данных в эндпоинты /reg или /auth также приводит к ошибке 400 Bad Request.  
 • Timeout Handling : Проверка истечения таймаута контекста. В случае истечения возвращается ошибка 500 InternalServerError.  
 • Используется log из стандартной библиотеки для логирования  
 • Файлы конфигурации загружаются из config.yml с помощью библиотеки viper  
 • Проект разделен на пакеты и представляет отдельный модуль  
Работа эндпоинтов  
 1. POST /reg (Регистрация) :  
 ◦ Извлекает traceID из контекста для логирования.  
 ◦ Проверяет метод запроса (POST). Если метод некорректный, возвращается ошибка 400 BadRequest.  
 ◦ Получает JSON-данные клиента (email, имя, пароль) и выполняет их анмаршаллинг в структуру пользователя.  
 ◦ Передает данные в бизнес-логику RegistrateAndLogin .  
 ◦ Возвращает ответ клиенту и добавляет данные сессии в cookie.  
 2. POST /auth (Аутентификация) :  
 ◦ Извлекает traceID из контекста для логирования.  
 ◦ Проверяет метод запроса (POST). Если метод некорректный, возвращается ошибка 400 BadRequest.  
 ◦ Получает JSON-данные клиента (email, пароль) и выполняет их анмаршаллинг в структуру пользователя.  
 ◦ Передает данные в бизнес-логику AuthenticateAndLogin .  
 ◦ Возвращает ответ клиенту и добавляет данные сессии в cookie.  
 3. DELETE /logout (Выход из аккаунта) :  
 ◦ Извлекает traceID, userID и sessionID из контекста для логирования.  
 ◦ Проверяет метод запроса (DELETE). Если метод некорректный, возвращается ошибка 400 BadRequest.  
 ◦ Передает данные в бизнес-логику Logout .  
 ◦ Возвращает ответ клиенту и удаляет данные сессии из cookie.  
 4. DELETE /delete (Удаление аккаунта) :  
 ◦ Извлекает traceID, userID и sessionID из контекста для логирования.  
 ◦ Проверяет метод запроса (DELETE). Если метод некорректный, возвращается ошибка 400 BadRequest.  
 ◦ Получает JSON-данные пароля от клиента и выполняет их анмаршаллинг.  
 ◦ Передает данные в бизнес-логику DeleteAccount .  
 ◦ Возвращает ответ клиенту и удаляет данные сессии из cookie.  
  
Уровень бизнес-логики  
 1. RegistrateAndLogin :  
 ◦ Получает данные пользователя (email, имя, пароль) из параметров и traceID из контекста.  
 ◦ Проводит валидацию данных: проверка формата email, минимальной длины имени и пароля.  
 ◦ Начинает транзакцию в базе данных с использованием механизма defer для rollback в случае ошибки.  
 ◦ Хэширует пароль с помощью библиотеки bcrypt и генерирует уникальный UUID для пользователя.  
 ◦ Сохраняет данные пользователя в базе данных методом CreateUser с использованием retry-логики:  
 ▪ Разделение ошибок на клиентские (например, дублирование email) и серверные (например, недоступность БД).  
 ▪ Проверка истечения таймаута контекста.  
 ◦ Создает сессию с помощью gRPC-запроса к микросервису SessionManagement , используя retry-логику.  
 ◦ В случае успеха выполняет commit транзакции и возвращает данные (expiryTime, userID, sessionID) в эндпоинт /reg.  
 ◦ В случае ошибки вызывает rollback транзакции и возвращает тип ошибки и её описание в эндпоинт /reg.  
 2. AuthenticateAndLogin :  
 ◦ Получает данные пользователя (email, пароль) из параметров и traceID из контекста.  
 ◦ Проводит валидацию данных: проверка формата email и минимальной длины пароля.  
 ◦ Получает данные пользователя из базы данных методом GetUser с использованием retry-логики:  
 ▪ Разделение ошибок на клиентские (например, пользователь не найден) и серверные (например, недоступность БД).  
 ▪ Проверка истечения таймаута контекста.  
 ◦ Создает сессию с помощью gRPC-запроса к микросервису SessionManagement , используя retry-логику.  
 ◦ В случае успеха возвращает данные (expiryTime, userID, sessionID) в эндпоинт /auth.  
 ◦ В случае ошибки возвращает тип ошибки и её описание в эндпоинт /auth.  
 3. Logout :  
 ◦ Получает sessionID из параметров и traceID из контекста.  
 ◦ Удаляет сессию с помощью gRPC-запроса к микросервису SessionManagement , используя retry-логику:  
 ▪ Разделение ошибок на клиентские (например, сессия не найдена) и серверные (например, недоступность сервиса).  
 ▪ Проверка истечения таймаута контекста.  
 ◦ В случае успеха возвращает управление в эндпоинт /logout.  
 ◦ В случае ошибки возвращает тип ошибки и её описание в эндпоинт /logout.  
 4. DeleteAccount :  
 ◦ Получает данные пользователя (userID, пароль, sessionID) из параметров и traceID из контекста.  
 ◦ Начинает транзакцию в базе данных с использованием механизма defer для rollback в случае ошибки.  
 ◦ Удаляет пользователя из базы данных методом DeleteUser с использованием retry-логики:  
 ▪ Разделение ошибок на клиентские (например, пользователь не найден) и серверные (например, недоступность БД).  
 ▪ Проверка истечения таймаута контекста.  
 ◦ Удаляет сессию с помощью gRPC-запроса к микросервису SessionManagement , используя retry-логику.  
 ◦ В случае успеха выполняет commit транзакции и возвращает управление в эндпоинт /delete.  
 ◦ В случае ошибки вызывает rollback транзакции и возвращает тип ошибки и её описание в эндпоинт /delete.  
Уровень базы данных  
 1. CreateUser :  
 ◦ Проверяет наличие данных пользователя в базе данных.  
 ◦ Возвращает клиентскую ошибку, если данные уже зарегистрированы.  
 ◦ В случае успеха сохраняет данные в базе данных и передает управление в бизнес-логику RegistrateAndLogin .  
 ◦ В случае ошибки возвращает серверную ошибку в бизнес-логику.  
 2. GetUser :  
 ◦ Проверяет наличие данных пользователя в базе данных.  
 ◦ Возвращает клиентскую ошибку, если данные не найдены.  
 ◦ В случае успеха возвращает userID в бизнес-логику AuthenticateAndLogin .  
 ◦ В случае ошибки возвращает серверную ошибку в бизнес-логику.  
 3. DeleteUser :  
 ◦ Проверяет наличие данных пользователя в базе данных.  
 ◦ Возвращает клиентскую ошибку, если данные не найдены.  
 ◦ В случае успеха удаляет данные из базы данных и передает управление в бизнес-логику DeleteAccount .  
 ◦ В случае ошибки возвращает серверную ошибку в бизнес-логику.  
Graceful Shutdown  
Реализована поддержка Graceful Shutdown :  
 • При получении сигнала завершения (SIGTERM или SIGINT) сервер завершает обработку текущих запросов и останавливает все активные горутины.  
 • Это гарантирует корректное завершение всех операций без потери данных.**

**SessionManagement отвечает за управление сессиями пользователей, включая их создание, валидацию и удаление. Сервис использует Redis для хранения данных сессий и предоставляет gRPC-интерфейс для взаимодействия с другими микросервисами, такими как API-Service и UserManagement .  
Технологический стек  
 • gRPC : Используется для обработки запросов от API-Service и UserManagement .  
 • Redis : Хранилище данных сессий, поддерживающее автоматическое удаление сессий после истечения времени жизни (TTL).  
 • Timeout Handling : Проверка истечения таймаута контекста. В случае истечения возвращается ошибка: codes.Internal.  
 • Logging : Используется библиотека Zap Logger для логирования событий с уникальным traceID для трассировки запросов.  
 • Конфигурация : Файлы конфигурации загружаются из config.yml с помощью библиотеки Viper .  
 • Модульность : Проект разделен на пакеты, что обеспечивает чистую архитектуру и удобство поддержки.  
Работа gRPC-эндпоинтов  
 1. CreateSession :  
 ◦ Извлекает traceID из метаданных запроса и добавляет его в контекст для логирования.  
 ◦ Передает управление в бизнес-логику CreateSession .  
 ◦ Возвращает ответ вызывающему обработчику (например, UserManagement ).  
 2. ValidateSession :  
 ◦ Извлекает traceID из метаданных запроса и добавляет его в контекст для логирования.  
 ◦ Передает управление в бизнес-логику ValidateSession .  
 ◦ Возвращает ответ вызывающему обработчику.  
 3. DeleteSession :  
 ◦ Извлекает traceID из метаданных запроса и добавляет его в контекст для логирования.  
 ◦ Передает управление в бизнес-логику DeleteSession .  
 ◦ Возвращает ответ вызывающему обработчику.  
Уровень бизнес-логики  
 1. CreateSession :  
 ◦ Проверяет истечение таймаута контекста. В случае истечения возвращается ошибка codes.Internal.  
 ◦ Получает userID из параметров запроса и traceID из контекста.  
 ◦ Проводит валидацию userID (проверка формата UUID). В случае некорректного формата возвращается ошибка codes.InvalidArgument.  
 ◦ Генерирует уникальный sessionID и сохраняет данные сессии (userID, expirationTime) в Redis через метод SetSession .  
 ▪ В случае ошибки Redis возвращается ошибка codes.Internal.  
 ◦ Возвращает успешный ответ с данными сессии в gRPC-эндпоинт.  
 2. ValidateSession :  
 ◦ Проверяет истечение таймаута контекста. В случае истечения возвращается ошибка codes.Internal.  
 ◦ Получает sessionID из параметров запроса и traceID из контекста.  
 ◦ Проводит валидацию sessionID (проверка формата UUID). В случае некорректного формата возвращается ошибка codes.InvalidArgument.  
 ◦ Вызывает метод GetSession для получения данных сессии из Redis.  
 ▪ Если сессия не найдена, возвращается ошибка codes.InvalidArgument.  
 ▪ В случае успеха возвращаются данные (userID, expirationTime) в gRPC-эндпоинт.  
 ▪ В случае ошибки Redis возвращается ошибка codes.Internal.  
 3. DeleteSession :  
 ◦ Проверяет истечение таймаута контекста. В случае истечения возвращается ошибка codes.Internal.  
 ◦ Получает sessionID из параметров запроса и traceID из контекста.  
 ◦ Проводит валидацию sessionID (проверка формата UUID). В случае некорректного формата возвращается ошибка codes.InvalidArgument.  
 ◦ Вызывает метод DeleteSession для удаления сессии из Redis.  
 ▪ Если сессия не найдена, возвращается ошибка codes.InvalidArgument.  
 ▪ В случае успеха возвращается подтверждение удаления сессии.  
 ▪ В случае ошибки Redis возвращается ошибка codes.Internal.  
Уровень базы данных  
 1. SetSession :  
 ◦ Проверяет истечение таймаута контекста. В случае истечения возвращается ошибка codes.Internal.  
 ◦ Сохраняет данные сессии (userID, expirationTime) в Redis с использованием sessionID в качестве ключа.  
 ◦ Устанавливает время жизни (TTL) для сессии, после которого она автоматически удаляется из Redis.  
 ◦ В случае ошибки Redis возвращается ошибка codes.Internal.  
 2. GetSession :  
 ◦ Проверяет истечение таймаута контекста. В случае истечения возвращается ошибка codes.Internal.  
 ◦ Извлекает данные сессии (userID, expirationTime) из Redis по ключу sessionID.  
 ▪ Если сессия не найдена, возвращается ошибка codes.InvalidArgument.  
 ▪ В случае успеха возвращаются данные сессии.  
 ▪ В случае ошибки Redis возвращается ошибка codes.Internal.  
 3. DeleteSession :  
 ◦ Проверяет истечение таймаута контекста. В случае истечения возвращается ошибка codes.Internal.  
 ◦ Удаляет сессию из Redis по ключу sessionID.  
 ▪ Если сессия не найдена, возвращается ошибка codes.InvalidArgument.  
 ▪ В случае успеха возвращается подтверждение удаления.  
 ▪ В случае ошибки Redis возвращается ошибка codes.Internal.  
Graceful Shutdown  
Реализована поддержка Graceful Shutdown :  
 • При получении сигнала завершения (SIGTERM или SIGINT) сервис завершает обработку текущих запросов и освобождает ресурсы.  
 • Это гарантирует корректное завершение всех операций без потери данных.**

**Система состоит из трех микросервисов:**