Клиент-серверное приложение

“Корпоративный планировщик задач сотрудников”.

Назначение:

Постановка и контроль выполнения задач компании с возможностью доступа к ним через внутренний сервер компании.

Состав приложения:  
1. Клиентская часть с GUI интерфейсом и возможностью подключения к серверу.

2. Серверная часть с подключенной базой данных и возможностью обработки запросов от клиентской части.

3. Консольное приложение для создания базы данных и занесения в неё начальных значений.

**Подробнее о клиентской части**:

Интерфейс пользователя создается с применением фреймворка Qt.

Приложение поддерживает три типа пользователей:

1. Администратор.
2. Оператор базы данных (db\_operator).
3. Обычный сотрудник (user, пользователь).

Возможности администратора:

1. Запрос от сервера списка сотрудников.
2. Создание нового сотрудника.
3. Изменение данных сотрудника: ФИО, тип пользователя, пароль.
4. Просмотр всех задач и на кого они назначены.
5. Отправка на сервер команды на завершение работы серверной части.

Возможности оператора базы данных:

1. Создание новой задачи с назначением на неё любого сотрудника.
2. Просмотр всех задач и на кого они назначены.
3. Изменение статуса любой задачи.
4. Возможность сдвига времени выполнения (дедлайна).

Возможности сотрудника:

1. Запрос от сервера и просмотр списка только своих задач.
2. Возможность запросить у сервера список всех задач.
3. Возможность взять на себя чужую задачу.
4. Создание новой задачи с автоматическим назначением на себя.
5. Изменение статуса своей задачи.

Алгоритм работы клиентской части:

1. Приложение запускается с указанием ip адреса сервера и номера порта.
2. Для каждого из трёх типов пользователей у приложения подготовлен свой интерфейс.
3. В случае отсутствия связи с сервером, приложение завершается с ошибкой.
4. Пользователь видит окно для ввода логина и пароля.
5. Введённая комбинация логина и пароля отправляется на сервер.
6. Если данные введены корректно, пользователю становится доступен интерфейс, настроенный согласно его типу.
7. Выбранное пользователем действие в виде json запроса отправляется на сервер.
8. В случае успеха, интерфейс отображает применённые изменения, а в случае ошибки, отображается дочернее окно с описанием ошибки.

**Подробнее о серверной части**:

Графической части нет.

После включения работает в режиме асинхронной обработки запросов от клиента, создавая для каждого клиента свою сессию. Количество сессий ограничено для защиты от слишком большого количества запросов. Асинхронная работа по сети реализована при помощи библиотеки boost.

Управляет базой данных, в которой хранятся данные о сотрудниках и назначенных задачах. База данных является реляционной и создана на основе sqlite. Серверная часть может изменять содержимое базы данных, но не может создавать базу.

База данных содержит всего пять таблиц, две из которых имеют начальные неизменяемые значения, а остальные три изменяются по мере отработки запросов от клиентской части. Доступ к таблицам защищен / синхронизирован мьютексом.

Неизменяемые таблицы:

1. UserTypes – содержит возможные типы пользователей.
2. TaskStatuses – содержит возможные статусы задач.

Изменяемые таблицы:

1. Users – таблица с пользователями.
2. Security – реализует связь между пользователем, его типом и паролем.
3. Tasks – содержит задачи, статусы задач и связь с пользователями.

**Подробнее о таблицах**:

Номер столбца, наименование, тип данных, описание.

*UserTypes*

1, user\_type\_number, TINYINT, уникальное число-ключ для типа пользователя.  
2, user\_type\_description, CHAR(16), текстовое название типа пользователя.

*TaskStatuses*

1, status\_type\_number, TINYINT, уникальное число - ключ для статуса задачи.  
2, status\_type\_description, CHAR(16), текстовое название статуса задачи.

*Users*

1, user\_unique\_id, SMALLINT, уникальное число – идентификатор пользователя.  
2, user\_unique\_login, CHAR(16), уникальный логин пользователя.  
3, first\_name, CHAR(24), фамилия.  
4, second\_name, CHAR(24), имя.  
5, patronymic, CHAR(24), отчество.

*Security*

1, user\_unique\_id, SMALLINT, уникальное число – идентификатор пользователя.  
2, user\_type\_number, TINYINT, число – тип пользователя.  
3, password, CHAR(32), пароль.

*Tasks*

1, task\_unique\_id, INTEGER, уникальное число – идентификатор задачи.  
2, user\_unique\_id, SMALLINT, число – идентификатор пользователя.  
3, status\_type\_number, TINYINT, число – статус задачи.  
4, deadline, DATE, крайний срок выполнения.  
5, task\_name, CHAR(64), название задачи.  
6, task\_description, VARCHAR(1000), описание, что нужно сделать.

**Подробнее о консольном приложении для создания базы.**

Графической части нет.

При запуске требует указания имени файла для новой базы данных.

Если в переданном имени отсутствует расширение “.sqlite”, добавляет автоматически.

Если файл с таким именем уже существует, программа завершается с ошибкой, т.к. можно создавать новую базу, но нельзя перезаписывать старую.

Программа создает новый файл с переданным именем и в нём новую базу данных.

Далее программа использует заранее подготовленные sql скрипты для следующих действий:

1. Создаются пять таблиц, описанные выше.
2. Таблицы UserTypes и TaskStatuses заполняются константными данными.
3. Таблицы Users, Security и Tasks заполняются начальными данными.
4. Производится вывод в консоль содержимого всех таблиц.

На этом работа консольного приложения завершается.

**Виды сетевых запросов от клиентской части к серверу.**

1. Проверка пары логин/пароль. Ответ сервера либо содержит ответ “false”, означающий, что с такой комбинацией авторизация не возможна, либо содержит ответ “true” и поле с номером типа пользователя user\_type\_number.
2. Запрос описания типа пользователя. Отправляем user\_type\_number и получаем строку с описанием типа: administrator, db\_operator, user.
3. Получение списка всех пользователей. Запрос массива со всеми user\_unique\_id пользователей.
4. Получение данных пользователя по его id. Отправить user\_unique\_id пользователя и получить значения полей user\_unique\_login, first\_name, second\_name и patronymic.
5. Изменение пароля пользователя. Отправляем user\_unique\_id и новый пароль. В ответе может быть либо true, либо false, если по какой-то причине не удалось сохранить в базе новый пароль.
6. Изменение типа пользователя. Отправляем user\_unique\_id и новый user\_type\_number. В ответе может быть либо true, либо false, если по какой-то причине не удалось записать в базу новый тип пользователя.
7. Создание нового пользователя. Отправляем логин, ФИО, тип пользователя и пароль. Новый user\_unique\_id формируется на сервере автоматически. В ответе от сервера получаем false, если произошла ошибка, например пользователь с таким логином уже есть, либо получаем true и строку с новым user\_unique\_id.
8. Получение списка всех задач. Запрос массива со всеми task\_unique\_id задач.
9. Запрос данных задачи. Отправляем task\_unique\_id. Получаем значения полей user\_unique\_id, status\_type\_number, deadline, task\_name, task\_description.
10. Получение описания статуса задачи. Отправляем status\_type\_number, а получаем строку со значением поля status\_type\_description.
11. Получение списка задач, назначенных на конкретного пользователя. Отправляем id пользователя user\_unique\_id, а получаем массив с task\_unique\_id, привязанных именно к нему.
12. Изменение статуса задачи. Отправляем task\_unique\_id и новый status\_type\_number. Получаем либо true, либо false в случае ошибки.
13. Изменение крайнего срока выполнения задачи. Отправляем task\_unique\_id и новый deadline. Получаем либо true, либо false в случае ошибки.
14. Назначение нового пользователя на выполнение задачи. Отправляем task\_unique\_id и новый user\_unique\_id. Получаем либо true, либо false в случае ошибки.
15. Создание новой задачи. Отправляем user\_unique\_id, deadline, task\_name и task\_description. Сервер сформирует новый task\_unique\_id и назначит задаче статус appointed. В ответе от сервера получаем false, если произошла ошибка при добавлении задачи в базу, либо получаем true и строку с новым task\_unique\_id.
16. Получение списка всех задач, имеющих определённый статус. Отправляем status\_type\_number, а получаем массив с соответствующими task\_unique\_id.
17. Получение списка всех задач, которые необходимо выполнить до определённой даты. Отправляем deadline, а получаем массив с соответствующими task\_unique\_id.
18. Получение списка всех задач, крайний срок исполнения для которых вышел. Отправляем deadline, а получаем массив с соответствующими task\_unique\_id.
19. Проверка, имеет ли пользователь доступ к задаче. Отправляем user\_unique\_id и task\_unique\_id. Получаем либо true, либо false, если пользователь с задачей не связан. Может понадобиться для сохранения результата редактирования задачи.