

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

К ЗАЩИТЕ ДОПУСТИТЬ

_____ *А. М. Ковальчук*

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

ИНФОРМАЦИОННАЯ СИСТЕМА ГОРОДСКОГО
АВТОТРАНСПОРТА

БГУИР КП 1-40 02 01 306 ПЗ

Студент гр. 150503

А. И. Ефимчик

Руководитель

А. В. Марзалюк

Минск 2022

СОДЕРЖАНИЕ

Перечень используемых сокращений и определений	6
Введение.....	7
1 Постановка задачи.....	8
2 Обзор литературы	9
3 Структура входных и выходных данных	10
3.1 Пример файла с информацией об останочных пунктах.....	11
3.2 Структура выходных данных	11
4 Структурное проектирование	12
4.1 Реализация класса bus_stop – автобусной остановки.....	12
4.2 Реализация core – ядра программы.....	13
4.3 Реализация класса person – персоны.....	13
4.4 Реализация класса passenger – пассажира.....	14
4.5 Реализация класса driver – водителя.....	14
4.6 Реализация класса drivers_manager – менеджера водителей	15
4.7 Реализация класса route – маршрута.....	15
4.8 Реализация класса routes_manager – менеджера маршрутов	16
4.9 Реализация класса travel_card – проездного.....	17
4.10 Реализация класса travel_cards_manager – менеджера про- ездных	17
4.11 Реализация класса uid_generator – генератора UID-ов	18
4.12 Реализация класса user – пользователя	18
4.13 Реализация класса user_admin – администратора.....	18
4.14 Реализация класса user_passenger – пользователя пассажира....	19
4.15 Реализация класса users_manager – менеджера пользователей..	20
4.16 Реализация класса vehicle – транспортного средства.....	20
4.17 Реализация класса tram – трамвая	21
4.18 Реализация класса bus – автобуса	21
4.19 Реализация класса e_bus – электробуса.....	22
4.20 Реализация класса vehicle_manager – менеджера ТС	22
4.21 Реализация класса controller – управления ИС из терминала.....	23
4.22 Реализация класса application – основного класса приложения..	24
4.23 Реализация класса renderer – вывод сообщений	24

4.24 Реализация класса manager – обязательные к реализации функции менеджера	25
4.25 Реализация класса application – основного класса приложения..	25
5 Функциональное проектирование	26
5.1 Схемы алгоритмов	26
5.2 Алгоритмы по шагам.....	26
6 Руководство пользователя	28
Заключение.....	32
Список использованных источников.....	33
Приложение А (обязательное) Листинг программы.....	34
Приложение Б (обязательное) Диаграмма классов приложения.....	85
Приложение В (обязательное) Блок-схемы алгоритмов методов.....	87
Приложение Г (обязательное) Ведомость документов	89

ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ И ОПРЕДЕЛЕНИЙ

В пояснительной записке будут использоваться следующие сокращения и определения:

Определение 1. *Сериализация – процесс перевода структуры данных в битовую последовательность*

Определение 2. *Десериализация – создание структуры данных из битовой последовательности*

1. ИС – информационная система
2. ТС – транспортное средство
3. ОП – остановочный пункт
4. БД – база данных
5. STL – Standard Template Library
6. UI – User Interface
7. UID – Unique Identifier
8. UML – Unified Modeling Language
9. API – Application Programming Interface

ВВЕДЕНИЕ

C++ — компилируемый строго типизированный язык программирования общего назначения. Наибольшее внимание уделено поддержке объектно-ориентированного программирования. Почему объектно-ориентированный подход к программированию стал приоритетным при разработке большинства программных проектов? ООП предлагает новый мощный способ решения проблемы сложности программ. Вместо чтобы рассматривать программу как набор последовательно выполняемых инструкций, в ООП программа представляется в виде совокупности объектов, обладающих сходными свойствами и набором действий, которые можно с ними производить.

Синтаксис C++ унаследован от языка C[1]. Одним из принципов разработки было сохранение совместимости с C. Тем не менее, C++ не является в строгом смысле надмножеством C. Множество программ, которые могут одинаково успешно транслироваться как компиляторами C, так и компиляторами C++, довольно велико, но не включает все возможные программы на C. Нововведениями C++ в сравнении с C являются:

- поддержка объектно-ориентированного программирования через классы. C++ предоставляет все четыре возможности ООП — абстракцию, инкапсуляцию, наследование (в том числе и множественное) и полиморфизм[2]

- поддержка обобщённого программирования через шаблоны
- стандартная библиотека C++ состоит из стандартной библиотеки C и библиотеки шаблонов (Standard Template Library, STL), которая предоставляет обширный набор обобщенных контейнеров и алгоритмов

- дополнительные типы данных
- обработка исключений
- виртуальные функции
- перегрузка (overloading) операторов
- перегрузка имён функций
- ссылки и операторы управления свободно распределяемой памятью

Данная программа предназначена для информатизации в сфере пассажирских перевозок. При бурном развитии общественного транспорта и глобальной информатизации мира подобный тип программ становится все более актуален.

1 ПОСТАНОВКА ЗАДАЧИ

Программа должна иметь текстовый пользовательский интерфейс (*TUI*) с необходимыми пунктами меню. Информация, используемая системой должна храниться в различных файлах (бинарных или текстовых), связанных определенным образом, и включать: информацию о пользователях, маршрутах, транспорте, остановках и проездных билетах. По запросу выдавать информацию о расписании, остановочных пунктах, оставшихся на проездном поездках, а так же статистику использования маршрута. Также необходимо создать возможность заполнять базу имеющихся остановок, маршрутах, водителях, пассажирах и транспорте.

Разработать иерархию классов с использованием наследования. Разработать и использовать в программе классы контейнеров и итераторов (с использованием STL). Производить обработку исключительных ситуаций.

В архитектуре ИС заложить архитектурную возможность для добавления функционала, новых видов транспорта. Предусмотреть два вида пользователей – пассажир и администратор, а так же возможность создания их учетных записей. Функционал администратора:

1. Создавать нового водителя
2. Просмотр имеющихся водителей
3. Редактирование водителя
4. Создание нового транспортного средства *ТС*
5. Просмотр имеющихся транспортных средств
6. Создание остановочного пункта
7. Просмотр имеющихся остановочных пунктов
8. Редактирование остановочного пункта
9. Создание нового маршрута
10. Добавление остановочного пункта в маршрут
11. Просмотр статистики маршрута
12. Просмотр информации о маршруте

Функционал пассажира:

1. Покупка проездных билетов
2. Просмотр количества оставшихся поездок
3. Входить в транспортное средство
4. Просмотр времени прибытия *ТС* на остановочном пункте

2 ОБЗОР ЛИТЕРАТУРЫ

Для дальнейшего понимания работы информационной системы необходимо пояснить некоторые термины, которые будут использоваться.

Определение 3. *Информационная система (ИС) — система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые и т. д.), которые обеспечивают и распространяют информацию.*

Определение 4. *Общественный транспорт — разновидность пассажирского транспорта как отрасли, предоставляющей услуги по перевозке людей по маршрутам, которые перевозчик заранее устанавливает, доводя до общего сведения способ доставки (транспортное средство), размер и форму оплаты, гарантируя регулярность (повторяемость движения по завершении производственного цикла перевозки), а также неизменяемость маршрута по требованию пассажиров.*

Определение 5. *Маршрут — путь следования объекта, учитывающий направление движения относительно географических ориентиров или координат, с указанием начальной, конечной и промежуточных точек, в случае их наличия.*

Определение 6. *Билет — документ, удостоверяющий наличие некоего права у какого-либо определённого лица или у предъявителя билета. Действие билета может распространяться на конкретное время или не иметь сроков.*

Определение 7. *Учётная запись — хранимая в компьютерной системе совокупность данных о пользователе, необходимая для его опознавания (аутентификации) и предоставления доступа к его личным данным и настройкам.*

Определение 8. *Аутентификация — процедура проверки подлинности пользователя путём сравнения введённого им пароля (для указанного логина) с паролем, сохранённым в базе данных пользовательских логинов.*

3 СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ

В качестве входных файлов в программе используются файлы сохраненных списков объектов. Такими объектами, подлежащими сохранению являются:

- Проездные билеты
- Транспортные средства
- Остановочные пункты
- Водители
- Маршруты
- Учетные записи пользователей

3.0.1 Файл с проездными билетамиВнутри себя хранит следующую информацию:

- Уникальный идентификатор (*UID*)
- Количество поездок

3.0.2 Файл с транспортными средствамиВнутри себя хранит следующую информацию:

- UID
- Регистрационный знак
- Тип ТС
- Вместительность

3.0.3 Файл с остановочными пунктамиВнутри себя хранит следующую информацию:

- UID
- Название
- Координаты

3.0.4 Файл с водителямиВнутри себя хранит следующую информацию:

- UID
- ФИО
- Дату рождения
- Тип водительской лицензии
- Дату истечения лицензии

3.0.5 Файл с маршрутамиВнутри себя хранит следующую информацию:

- UID
- Список остановочных пунктов
- Точку отправления и прибытия
- UID водителя, выполняющего рейс
- UID ТС, выполняющего рейс
-

3.0.6 Файл с учетными записямиВнутри себя хранит следующую информацию:

- UID
- Логин
- Пароль
- UID проездного билета
- ФИО
- Дату рождения

3.1 Пример файла с информацией об остановочных пунктах

```
22 serialization::archive 19 0 0 3 0 0 0 4 6
Vostok 1.00e+00 1.00e+00 0 0 0 0
254 6 Gikalo 3.00e+00 2.00e+00 0 0 162 6
Urusca 3.00e+00 2.00e+00 0 0
```

3.2 Структура выходных данных

В качестве выходных данных используются те же файлы, что и во входных данных, однако при выходе из программы они перезаписываются, тем самым сохраняя внесенные пользователем изменения.

4 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ

Прежде чем описывать каждый класс в отдельности, стоит прояснить архитектуру программы в целом. При проектировании программы одной из основных задач была задача построить максимально гибкую и удобную для расширения систему.

Для реализации задача был выбран подход, использующий *менеджеров* для управления множеством одинаковых объектов[3]. Такой *менеджер* предусмотрен для каждой значимой единицы и имеет соответствующие методы для управления этими объектами. Также менеджеры имеют обязательные для реализации функции сохранения и загрузки данных из базы данных (БД).

Все менеджеры агрегированы в ядро программного продукта *core*, который предоставляет доступ к функционалу менеджеров из вне.

Использование команд позволяет не только управлять изменением модели в обе стороны, но и при достаточной грануляции команд позволяет комбинировать их различным образом, что дает возможность реализовать карты способностей с различными поведением а также преимущество повторного использования кода[4]. UML-диаграмма информационной системы представлена в Приложении Б (ГУИР.400201.002 РР). Листинг ИС приведен в приложении А.

4.1 Реализация класса `bus_stop` – автобусной остановки

```
class bus_stop {
std::string get_options_string();
    - сериализация удобств остановки
bus_stop() = default;
    - конструктор по-умолчанию
bus_stop(const std::string&, double, double);
    - конструктор с параметрами
UID get_uid() const;
    - получение значения UID
void set_uid(UID u);
    - установка UID
const std::string& get_name() const;
    - получение названия
```

```

void set_name(const std::string& n);
    - установка названия
std::string get_coords() const;
    - получение координат
void set_coords(double, double);
    - установка координат
void set_options(std::list<stop_options> stop_options);
    - установка удобств остановки
std::string view_existing_options();
    - получение всех существующих удобств
std::string serialize_ui();
    - сериализация остановки для интерфейса
};

```

4.2 Реализация core – ядра программы

```

class core {
static travel_cards_manager tcm;
    - объект менеджера проездных
static routes_manager rm;
    - объект менеджера маршрутов
static bus_stops_manager bsm;
    - объект менеджера остановок
static drivers_manager dm;
    - объект менеджера водителей
static vehicles_manager vm;
    - объект менеджера ТС
static void load();
    - метод загрузки данных из БД
static void save();
    - метод загрузки данных в БД
};

```

4.3 Реализация класса person – персоны

```

class person {
person(std::string first_name, std::string last_name
, std::string dob);

```

- конструктор с параметрами

```
person() = default;
```

- конструктор по-умолчанию

```
virtual std::string serialize_ui();
```

- виртуальный метод сериализатора для интерфейса

```
std::string get_first_last_name();
```

- получение имени и фамилии

```
UID get_uid() const;
```

- получение UID

```
static void ask_names_dob(std::string req, std::string& f_name,
    std::string& last_name, std::string& dob);
```

- статический метод для запроса данных для регистрации

```
};
```

4.4 Реализация класса passenger – пассажира

```
class passenger : public person {
passenger() = default;
passenger(std::string f_name, std::string l_name, std::string dob);
UID card;
    - UID проездного пассажира
std::string serialize_ui() override;
UID get_card();
    - получение UID проездного
};
```

4.5 Реализация класса driver – водителя

```
class driver : public person {
driver() = default;
    - конструктор по-умолчанию
driver(std::string first_name, std::string last_name,
    std::string dob, vehicle::vehicle_type v_type, date lic_exp);
    - конструктор с параметрами
std::string serialize_ui() override
{
return person::serialize_ui()+serialize_license();
};
```

- сериализатор для интерфейса

```
std::string serialize_license();
```

- получение информации о лицензии

```
const date& get_license_expiration() const;
```

- получение даты истечения лицензии

```
vehicle::vehicle_type get_license_type() const;
```

- получение типа лицензии

```
};
```

4.6 Реализация класса `drivers_manager` – менеджера водителей

```
class drivers_manager : public manager {
driver& find_driver(UID);
    - поиск водителя по UID
driver& add_driver(std::string first_name, std::string last_name,
    std::string dob, std::string v_type, std::string exp_date);
    - добавление нового водителя
std::string serialize_all_drivers();
    - сериализация всех водителей
bool check_if_driver_exists(UID);
    - проверка существования водителя
};
```

4.7 Реализация класса `route` – маршрута

```
class route {
UID get_uid() const;
    - получение UID
UID get_route_driver() const;
    - получение UID водителя
void set_route_driver(UID driver);
    - установка UID водителя
UID get_vehicle() const;
    - получение UID транспортного средства
void set_vehicle(UID veh);
    - установка транспортного средства
route() = default;
```

- конструктор по-умолчанию.

```
route(UID vehicle, UID route_driver, std::string src,
std::string dst);
```

- конструктор с параметрами

```
std::string serialize_route();
```

- сериализация направления рейса

```
std::string serialize_full_route();
```

- сериализация полного маршрута

```
std::string serialize_stats();
```

- сериализация статистики маршрута

```
void increment_popularity();
```

- увеличить популярность

```
void delete_stop(uint8_t id);
```

- удалить остановку из маршрута

```
void add_stop(UID u, time_t arrival_time, bool need_to_stop);
```

- добавить остановку в маршрут

```
bool check_arrival_time(UID bus_stop_uid, time_t& arrival_time);
```

- узнать время прибытия ТС на остановку

```
};
```

4.8 Реализация класса `routes_manager` – менеджера маршрутов

```
class routes_manager : public manager{
route& add_route(UID vehicle, UID driver, std::string src_point,
std::string dst_point);
- добавление маршрута
route& find_route(UID);
- поиск маршрута по UID
std::list<route> get_routes_with_stop(UID stop_id);
- получить список маршрутов с остановкой
std::string serialize_all_routes_path();
- сериализовать все направления маршрутов
std::string serialize_all_routes_stats();
- сериализовать всю статистику по маршрутам
};
```

4.9 Реализация класса `travel_card` – проездного

```
class travel_card {
    UID uid;
    uint8_t remaining_trips;
    - количество поездок
    travel_card();
    - конструктор по-умолчанию
    ~travel_card() = default;
    - деструктор
    explicit travel_card(UID);
    - конструктор с аргументом
    uint8_t get_remaining_trips() const;
    - получения количества поездок
    void set_remaining_trips(uint8_t remaining_trips);
    - установка поездок
    void decrement_trips();
    - уменьшение количества поездок
    UID get_uid() const;
    - получение UID
    void set_uid(UID u);
    - установка UID
};
```

4.10 Реализация класса `travel_cards_manager` – менеджера проездных

```
class travel_cards_manager : public manager {
    void create_if_not_exists(UID);
    - создать проездной, если не существует
    travel_card& find_travel_card(UID);
    - поиск проездного по UID
    static bool validator(travel_card&);
    - валидатор с функцией декремента поездок
};
```

4.11 Реализация класса uid_generator – генератора UID-ов

```
class uid_generator {  
    static UID generate(); - генерирует рандомный UID  
};
```

4.12 Реализация класса user – пользователя

```
class user {  
    std::string login;  
    std::string password;  
    std::string get_hash(std::string source_str);  
    - в будущем функция будет использовать хэширование пароля  
    user(std::string login, std::string password);  
    - конструктор с параметрами  
    user() = default;  
    - конструктор по-умолчанию  
    bool validate_credentials(std::string, std::string);  
    - аутентификация  
    static void ask_credentials(std::string, std::string&,  
        std::string&);  
};
```

4.13 Реализация класса user_admin – администратора

```
class user_admin : public user {  
    user_admin() = default;  
    - конструктор по-умолчанию  
    user_admin(const std::string&, const std::string&);  
    - конструктор с параметрами  
    void create_driver();  
    - создание водителя  
    void view_drivers();  
    - просмотр водителей  
    void modify_driver();  
    - редактирование водителя  
    void create_vehicle();  
    - создание транспортного средства (ТС)
```



```

void view_vehicles();
    - просмотр ТС
void create_bus_stop();
    - создание остановки
void view_bus_stops();
    - просмотр остановок
void modify_bus_stop();
    - редактирование остановок
void create_route();
    - создание маршрута
void add_stop_to_route();
    - добавление остановки в маршрут
void serialize_routes_stats();
    - сериализатор статистики маршрута
void route_serialize_information();
    - сериализация информации о маршруте
};

```

4.14 Реализация класса `user_passenger` – пользователя пассажира

```

class user_passenger : public user, public passenger {
user_passenger() = default;
    - конструктор по-умолчанию
user_passenger(std::string f_name, std::string l_name,
    std::string dob,
    std::string login, std::string password);
    - конструктор с параметрами
void buy_trips();
    - покупка поездок
void view_remaining_trips();
    - просмотр оставшихся поездок
void enter_bus();
    - вход в ТС
void get_arrival_time();
    - просмотр времени прибытия ТС на остановку

```

```
};
```

4.15 Реализация класса `users_manager` – менеджера пользователей

```
class users_manager : public manager {
std::list<user_passenger> l_passengers;
std::list<user_admin> l_admins;
user_passenger& sign_up_passenger();
    - регистрация пассажира
user_admin& sign_up_admin();
    - регистрация администратора
user_passenger& sign_in_passenger();
    - аутентификация пассажира
user_admin& sign_in_admin();
    - аутентификация администратора
};
```

4.16 Реализация класса `vehicle` – транспортного средства

```
class vehicle {
enum vehicle_type {
bus,
e_bus,
tram
}; - типы ТС
static std::map<vehicle_type, std::string> vehicle_type_string;
    - соотношение ТС и названия
UID uid;
std::string registration_mark;
    - регистрационный номер
vehicle_type type;
uint8_t capacity;
    - вместительность
vehicle() = default;
    - конструктор по-умолчанию
UID get_uid() const;
    - просмотр UID
```

```

const std::string& get_registration_mark() const;
    - просмотр регистрационного знака
vehicle_type get_type() const;
    - просмотр типа ТС
uint8_t get_capacity() const;
    - просмотр вместительности
vehicle(std::string registration_mark, vehicle::vehicle_type type
, uint8_t capacity);
bool validate_card(travel_card&);
    - валидатор проездного
virtual int get_travel_distance();
    - максимальное расстояние к передвижению
virtual std::string serialize_ui();
    - сериализация ТС для интерфейса
static std::string serialize_vehicle_type(vehicle_type);
    - сериализация типа ТС
static vehicle_type parse_vehicle_type(std::string);
    - парсинг типа ТС в тип ТС
static std::string view_existing_types();
    - вывод существующих типов ТС
};

```

4.17 Реализация класса tram – трамвая

```

class tram : public vehicle {
tram() = default;
tram(std::string, uint8_t);
};

```

4.18 Реализация класса bus – автобуса

```

class bus : public vehicle {
uint8_t fuel_bank_size;
    - вместительность топливного бака
double fuel_consumption;
    - расход топлива
bus() = default;
    - конструктор по-умолчанию

```

```
bus(std::string, uint8_t, uint8_t, double);  
    - конструктор с параметрами  
};
```

4.19 Реализация класса e_bus – электробуса

```
class e_bus : public vehicle {  
double battery_consumption;  
    - потребление заряда  
e_bus() = default;  
    - конструктор по-умолчанию  
e_bus(std::string registration_mark, uint8_t capacity,  
double battery_consumption);  
    - конструктор с параметрами  
};
```

4.20 Реализация класса vehicle_manager – менеджера ТС

```
class vehicles_manager : public manager{  
std::list<bus> l_buses;  
    - список автобусов  
std::list<e_bus> l_e_buses;  
    - список электробусов  
std::list<tram> l_trams;  
    - список трамваев  
bus& find_bus(UUID);  
    - поиск автобуса по UUID  
bus& add_bus(std::string, uint8_t, uint8_t, double);  
    - добавление нового автобуса  
std::string serialize_all_buses();  
    - сериализация всех автобусов  
e_bus& find_e_bus(UUID);  
    - поиск электробуса по UUID  
e_bus& add_e_bus(std::string, uint8_t, double);  
    - добавление нового электробуса  
std::string serialize_all_e_buses();  
    - сериализация всех электробусов
```

```

tram& find_tram(UID);
    - поиск трамвая по UID
tram& add_tram(std::string, uint8_t);
    - добавление нового трамвая
std::string serialize_all_trams();
    - сериализация всех трамваев
bool check_if_vehicle_exists(UID);
    - проверка на существование транспорта с UID
vehicle::vehicle_type get_vehicle_type(UID);
    - возвращает тип ТС по его UID
vehicle& get_vehicle_by_id(UID);
    - возвращает приведенное к vehicle ТС по UID
};

```

4.21 Реализация класса controller – управления ИС из терминала

```

class controller {
explicit controller(users_manager& u);
    - конструктор с аргументом менеджера пользователей
void run();
    - запуск контроллера
void admin_ui(user_admin& a);
    - UI администратора
void passenger_ui(user_passenger& p);
    - UI пассажира
users_manager& umanager;
    - объект менеджера пользователей
struct func_admin {
std::string description;
std::_Mem_fn<void (user_admin::*)()> function;
};
    - структура интерфейса администратора, содержащая
    описание функции и саму функцию
static std::map<int, struct func_admin> admin_functions;
    - карта индекса функции и структуры для администратора
struct func_passenger {

```

```

std::string description;
std::_Mem_fn<void (user_passenger::*)()> function;
};
    - структура интерфейса пользователя, содержащая
    описание функции и саму функцию
static std::map<int, struct func_passenger> passenger_functions;
    - карта индекса функции и структуры для пассажира
};

```

4.22 Реализация класса application – основного класса приложения

```

class date {
uint16_t year = 0;
uint8_t month = 0;
uint8_t day = 0;
date(uint16_t year, uint8_t month, uint8_t day);
    - конструктор с численными параметрами
date(std::string);
    - конструктор с функцией парсинга даты из строки
date() = default;
    - конструктор по-умолчанию
std::string serialize_ui() const;
    - сериализация даты
void set_date(uint16_t y, uint8_t m, uint8_t d);
    - установка с численными параметрами
void set_date(std::tm);
    - установка из объекта времени
void set_date(std::string);
    - установка с функцией парсинга даты из строки
static bool date_parser(std::string& s, tm& date);
    - парсер даты из строки в объект времени
};

```

4.23 Реализация класса renderer – вывод сообщений

```

class renderer {
static void render_boot_screen()

```

- вывод заставки при загрузке ИС

```
static void render_message(const std::string& msg)
```

- вывод сообщения на экран

```
static void render_error(const std::string& err)
```

- вывод ошибки на экран

```
static void render_headline(const std::string& headline)
```

- вывод заголовка

```
};
```

4.24 Реализация класса manager – обязательные к реализации функции менеджера

```
class manager {
public:
virtual void save_db(const std::string&) = 0;
    - сохранение данных менеджера в БД
virtual void load_db(const std::string&) = 0;
    - загрузка данных из БД в менеджер
};
```

4.25 Реализация класса application – основного класса приложения

```
class application {
void run();
    - запуск приложения
void quit();
    - завершение работы приложения
~application();
    - деструктор
users_manager umanager;
    - менеджер пользователей
};
```

5 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

5.1 Схемы алгоритмов

5.1.1 Алгоритм функции `void user_passenger::`

`enter_bus()`. Это функция-член класса `user_passenger`, задача которой — обработать процесс входа пассажира в ТС с валидацией и бизнес-логикой. Схема алгоритма представлена в Приложении Б (ГУИР.400201.00В ПД).

5.1.2 Алгоритм функции `void user_passenger::`

`get_arrival_time()`. Это функция-член класса `user_passenger`, задача которой — обработать процесс запроса пользователем расписания прибытия ТС на остановку. Схема алгоритма представлена в Приложении В (ГУИР.400201.001 ПД).

5.2 Алгоритмы по шагам

5.2.1 Алгоритм[5] по шагам функции

`user_admin::add_stop_to_route()`. Данная функция-член класса `user_admin` служит для добавления остановочного пункта в маршрут.

1. Начало
2. Вывод существующих маршрутов
3. Запрос на ввод UID маршрута для добавления
4. Поиск маршрута. Если найден – продолжаем, если была ошибка – завершение работы функции
5. Вывод существующих остановок
6. Запрос на ввод UID остановки
7. Поиск остановки. Если найдена – продолжаем, если была ошибка – завершение работы функции
8. Запрос на ввод времени прибытия ТС на эту остановку
9. Добавление остановки через соответствующий метод

5.2.2 Алгоритм по шагам функции

`user_admin::create_route()`. Данная функция-член класса `user_admin` служит для добавления остановочного пункта в маршрут.

1. Начало
2. Вывод существующих автобусов, трамваев, электробусов
3. Запрос на ввод UID ТС для нового маршрута

4. Вывод существующих водителей
5. Запрос на ввод UID водителя для нового маршрута
6. Проверка на соответствие типа лицензии водителя и типа ТС создаваемого маршрута. Если совпадают – продолжаем, если была ошибка – завершение работы функции
7. Запрос на ввод точки отправления и прибытия
8. Добавлени нового маршрута через соответствующий метод менеджера маршрутов

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Результаты работы части функционала информационной системы представлены на скрин-шотах ниже.

I - - - - - H - - - - - W / - - - - - W
 I / I I W I I I W I I I C I W I
 I I I I W I I - - - - -
 - I I - - I I W I I I I W I
 I - - - - - I - - - - - I W - - - - -

```
You want to
login as admin (1)
login as passenger (2)
sign up admin (3)
sign up passenger (4)
exit (0)?
1
Enter admin's username and password:
admin
admin
1. Create driver
2. View drivers
3. Modify drivers
4. Create vehicle
5. View vehicles
6. Create bus stop
7. View bus stops
8. Modify bus stop
9. Create route
10. Add stop to route
11. Serialize routes stats
12. Serialize route information
0. Exit
```

1

a

| - - - ||- \ \ / -|.' ----- \ \

| / - \ \ \ | - \ \ / -| | C--- \ \

| | | \ \ / | - .

-| | - -| | \ \ / | - | \ \) |

|-----| |-----| \ \-----'

```
You want to
login as admin (1)
login as passenger (2)
sign up admin (3)
sign up passenger (4)
exit (0)?
2
Enter passengers's username and password:
sasha
0
1. Buy trips
2. View remaining trips
3. Enter bus
4. Get arrival time
0. Exit
```

1

6

Рисунок 6.1 – Процесс аутентификации а - администратора, б - пассажира

<pre> 2 ----- [Person-64] Name: Kerl Meraki Date of birth: 1998.3.22 License for Tram valid until 2026.4.4 [Person-193] Name: Ivan Oskin Date of birth: 2000.2.2 License for Bus valid until 2025.6.23 ----- </pre> <p style="text-align: center;">а</p>	<pre> 7 ----- [Bus-stop-4] Name: Vostok Coordinates: Coordinates: 1 1 Options: [Bus-stop-254] Name: Gikalo Coordinates: Coordinates: 2 3 Options: Rain cover, USB Charger [Bus-stop-162] Name: Urucca Coordinates: Coordinates: 2 3 Options: ----- </pre> <p style="text-align: center;">б</p>
<pre> 5 Enter vehicle type to view (Bus, Electro-bus, Tram, All): Electro-bus ----- [Electro-bus-172] Registration mark: 2222RH-1 Passenger capacity: 200 [Electro-bus-32] Registration mark: 2346AM-7 Passenger capacity: 80 ----- </pre> <p style="text-align: center;">в</p>	<pre> 11 [Route-5] "Ozero -> Pl.Peramogi" Used by 0 people [Route-170] "Urucca -> Gikalo" Used by 1 people </pre> <p style="text-align: center;">г</p>

Рисунок 6.2 – Просмотр существующих а - водителей, б - остановок, в - транспортных средств, г - статистики маршрутов

```

4
Enter bus stop id, where you are: [ 4 Vostok | 254 Gikalo | 162 Urucca | ]
254
[Bus-stop-254]
[Route-170] "Urucca -> Gikalo" at 25

```

Рисунок 6.3 – Просмотр расписания на остановке

```
12
-----
[Route-5] "Ozero -> Pl.Peramogi"
[Route-170] "Urucca -> Gikalo"
-----

Select route to serialize:
170
[Route-170] "Urucca -> Gikalo"
Vehicle UID: 13
Stop list:
[Stop-254] "Gikalo" at 25
[Stop-4] "Vostok" at 20
[Stop-254] "Gikalo" at 13
```

Рисунок 6.4 – Просмотр выбранного маршрута

```
1
How many trips you want to buy? 50
1. Buy trips
2. View remaining trips
3. Enter bus
4. Get arrival time
0. Exit
2
You have 113 remaining trips for your 223 travel card
1. Buy trips
2. View remaining trips
3. Enter bus
4. Get arrival time
0. Exit
█
```

Рисунок 6.5 – Пополнение проездного

```
3
Enter bus stop id, where you are: [ 4 Vostok | 254 Gikalo | 162 Urucca | ]
254
[Route-170] "Urucca -> Gikalo"
Enter route, you want to enter: 170
1. Buy trips
2. View remaining trips
3. Enter bus
4. Get arrival time
0. Exit
2
You have 112 remaining trips for your 223 travel card
```

Рисунок 6.6 – Вход в ТС на остановке и уменьшение количества поездок

```
8
[ 4 Vostok | 254 Gikalo | 162 Urucca | ]
Enter stop's UID to modify: 254
What field you want to modify?
1. Name
2.Coordinates
3. Options
: 3
There are few available stop options:
[0] Rain cover
[1] USB Charger
[2] Coffe Machine

Enter options indexes (with space separator):
0 1
```

Рисунок 6.7 – Редактирование остановки

ЗАКЛЮЧЕНИЕ

В данном разделе подведены итоги разработки информационной системы городского автотранспорта. В ходе разработки и тестирования системы было проверено, что в ней выполняются все ее необходимые функции:

1. Создавать учетную запись администратора
2. Создавать учетную запись пассажира
3. Создавать нового водителя
4. Просмотр имеющихся водителей
5. Редактирование водителя
6. Создание нового транспортного средства (*ТС*)
7. Просмотр имеющихся транспортных средств
8. Создание остановочного пункта
9. Просмотр имеющихся остановочных пунктов
10. Редактирование остановочного пункта
11. Создание нового маршрута
12. Добавление остановочного пункта в маршрут
13. Просмотр статистики маршрута
14. Просмотр информации о маршруте
15. Покупка проездных билетов
16. Просмотр количества оставшихся поездок
17. Входить в транспортное средство
18. Просмотр времени прибытия *ТС* на остановочном пункте

Для реализации системы были использованы библиотеки по работе с сериализацией (*Boost*) и стандартные алгоритмы языка C++. В ходе выполнения курсовой работы были получены навыки работы с языком C++, проработке архитектуры приложения, а так же закреплены навыки работы с объектно-ориентированным программированием. В проекте активно используются основные концепции ООП. В дальнейшем данную информационную систему можно будет усовершенствовать и добавить больший функционал, к примеру шифрование пароля, создание API-endpoint'ов для сторонних интеграций и создания *GUI*. Работа над курсовым проектом помогла развить навык конструирования объектно-ориентированных программ, навык работы со сторонними библиотеками, настройки сборки проектов на языке C++ и работу над архитектурой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Страуструп, Бьерн. Язык программирования C++ / Бьерн Страуструп. — М. : Издательство БИНОМ, 2004. — 1098 с.
- [2] Луцик Ю.А. Ковальчук А.М., Лукьянова И.В. Объектно-ориентированное программирование на языке C++ / Лукьянова И.В. Луцик Ю.А., Ковальчук А.М. — М. : Издательство БИНОМ, 2003. — 203 с.
- [3] Фримен Э. Фримен Э., Сьерра К. Бейтс Б. Паттерны проектирования / Сьерра К. Бейтс Б. Фримен Э., Фримен Э. — М. : Питер, 2004. — 656 с.
- [4] В.А., Скляров. Язык C++ и объектно-ориентированное программирование / Скляров В.А. — М. : Выш.шк., 1997. — 478 с.
- [5] Дейтел Х., Дейтел П. Как программировать на C++ / Дейтел П. Дейтел Х. — М. : Издательство БИНОМ, 2001. — 1152 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программы

ПРИЛОЖЕНИЕ Б
(обязательное)
Диаграмма классов приложения

ПРИЛОЖЕНИЕ В
(обязательное)
Блок-схемы алгоритмов методов

ПРИЛОЖЕНИЕ Г
(обязательное)
Ведомость документов