Silesian University of Technology

SILESIAN UNIVERSITY OF TECHNOLOGY

FACULTY OF AUTOMATIC CONTROL, ELECTRONICS AND COMPUTER SCIENCE

PROGRAMME: INFORMATICS

Computer Programming (sem. 4)

Final Project Report

Author: Youssef AL BALI

Instructor: Wojciech Dudzik

Date: 09/06/2022

TOPIC ANALYSIS

The topic of the project is a basic GYM management system application.

<u>The main features of the application are</u>:

- Various account types:
  - For the gym clients;
  - For the trainers;
  - For the gym administrators.
- Managing members and trainers of the given gym;
- The ability for the member register/login
  - See his info
  - Change password
  - Update data
  - Reserve a private session with a trainer in his category
  - Make a workout program with a trainer in his category and based on his level.
- ***Admin*** console which shows the list of members, trainers and sessions with the ability to:
  - Add new members/trainers;
  - Remove existing members/ trainers;
  - Update a member's data or reset his password if needed
  - See all scheduled sessions in the gym
- A ***Trainer*** console with almost the same functionalities as ***Member***
- Graphical user interface implemented with Qt.
- The program is implemented fully in C++ (Latest STD) in order to cover the topics from thematic classes:

  <u>Threads</u>: for the background database access ***(reading/updating),*** also used with functions that can be executed concurrently.

  <u>Ranges</u>: for an easier and faster loop iteration, used with std algorithms like (adjacent_find, copy_if, find_if…)

<u>Filesystem</u>: Used along with reading and updating the database, additionally used in **Csv_creator_class** to generated new versions of files each time.

<u>Regular expressions</u>: Used for checking the password security requirements, and along with filesystem in Csv_creator_class.

- When it comes to the database, I didn't use any ready-made database engine. Instead, I have implemented my own customized one using JavaScript Object Notation to represent the entities and sessions (source library https://github.com/nlohmann/json was used for parsing **JSON**.)
- For timing management, I used **Boost.Date_Time** library in order to store and write the timing slots.

EXTERNAL SPECIFICATION

From the **Admin** point of view, he firstly needs to login:

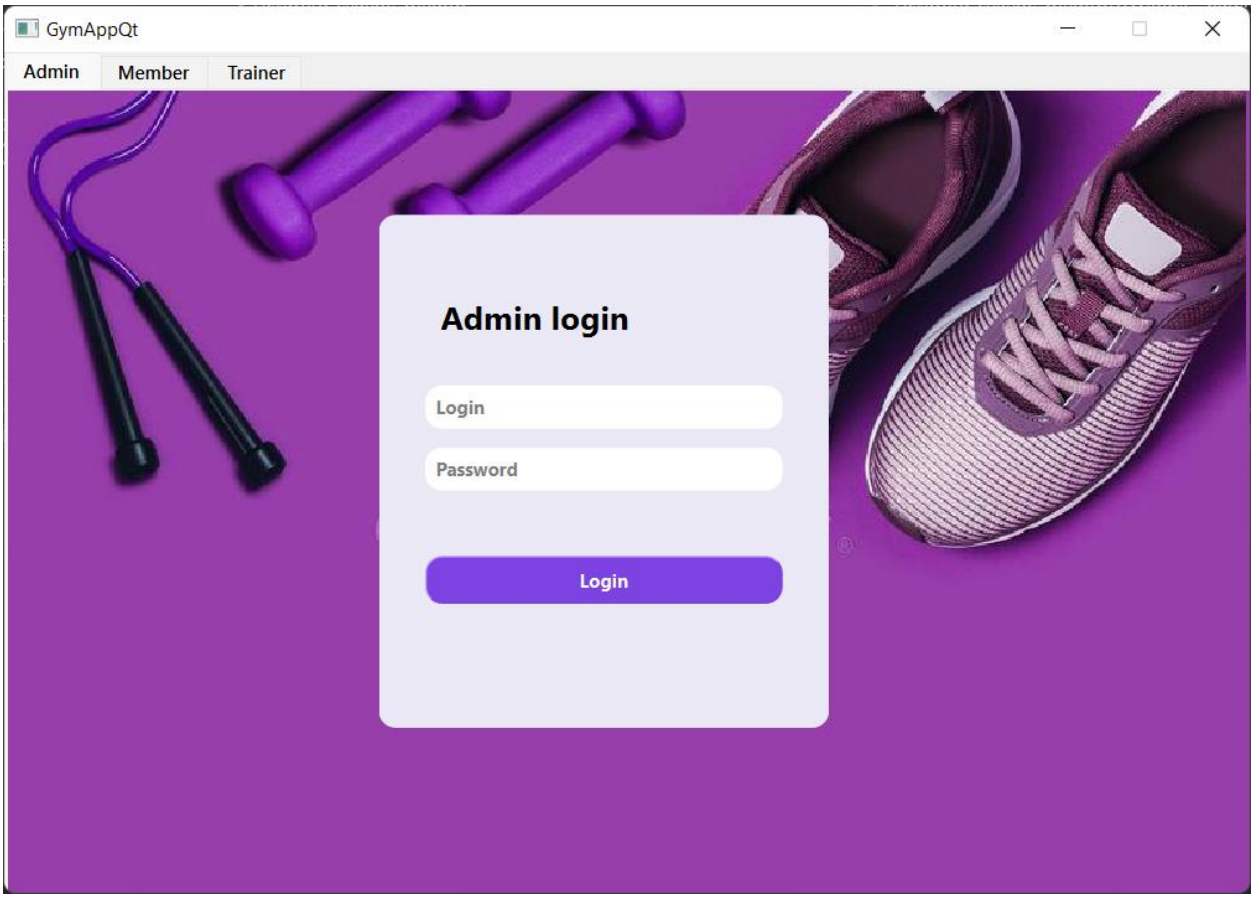


Figure 1 Admin login

- He’s able to see his info, and list of Members with the ability to update their data or erase them from the gym or reset the password if needed

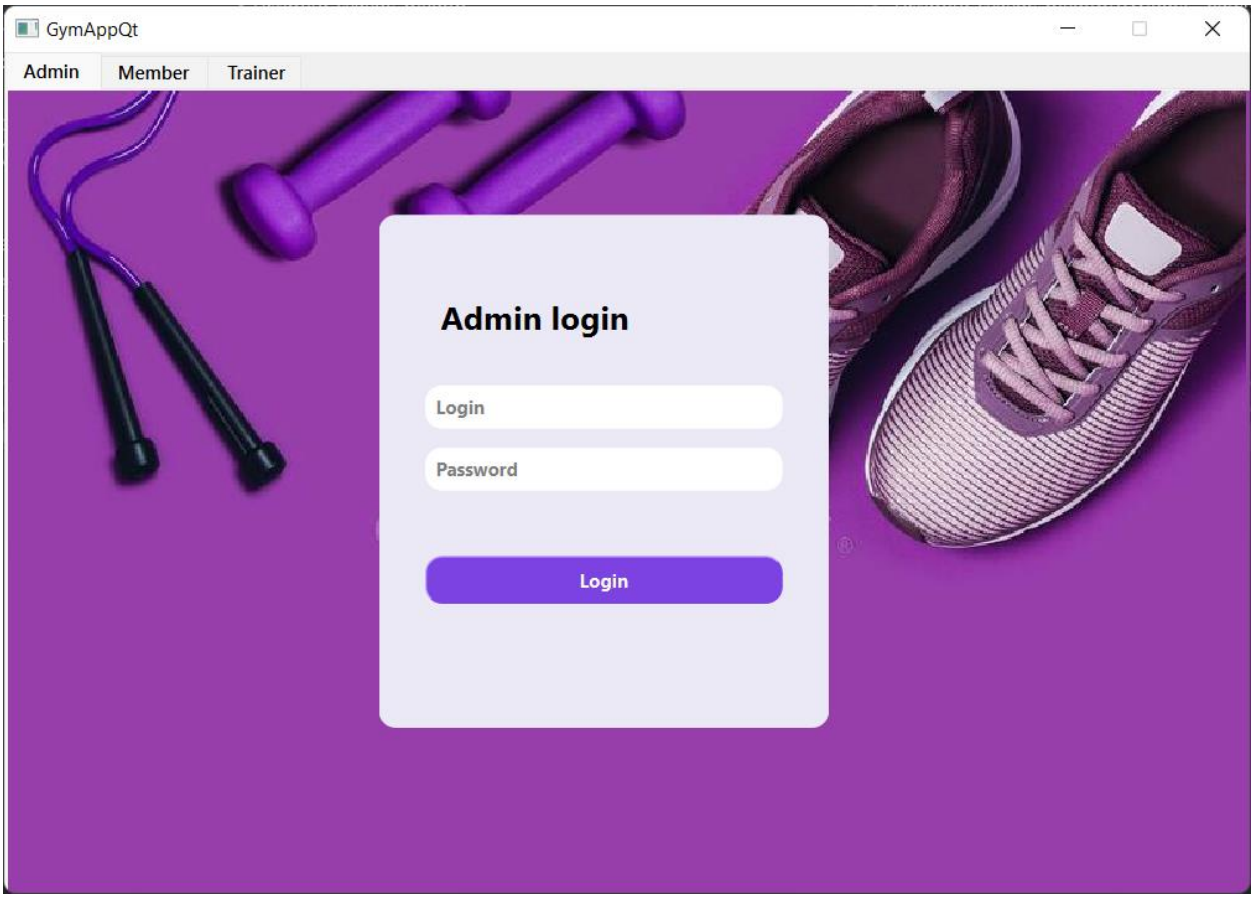


Figure 2 Admin Member view

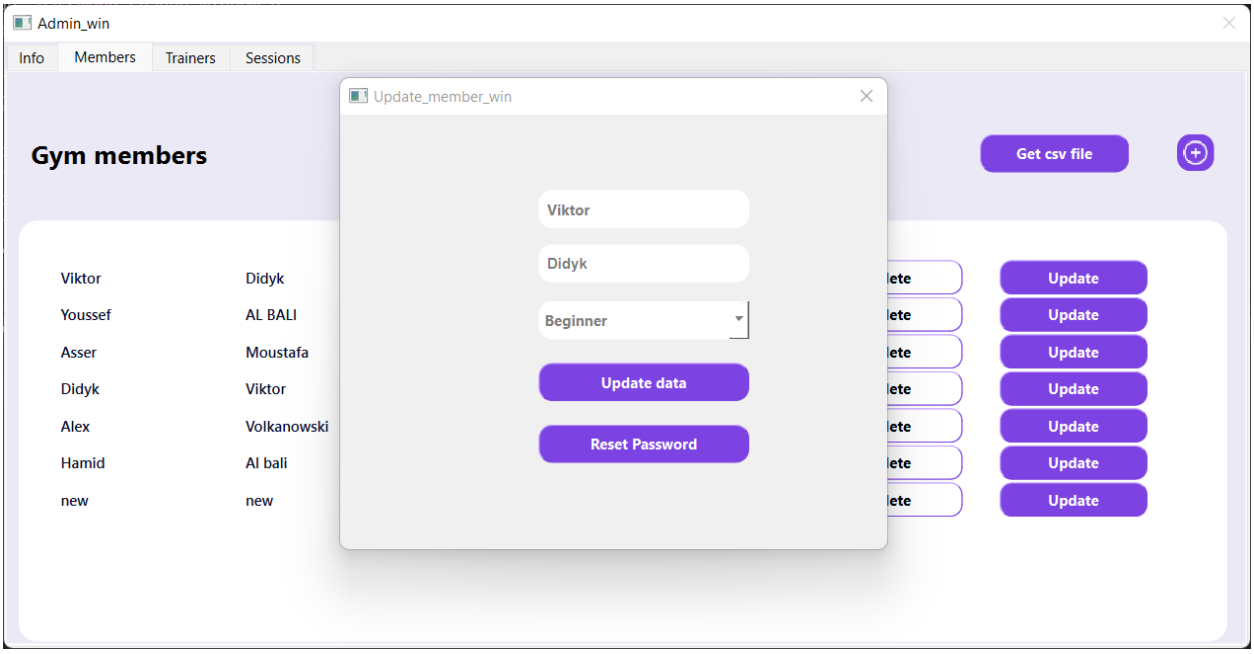


Figure 3 Admin update Member view

- See current Trainers in the gyms with ability to delete them

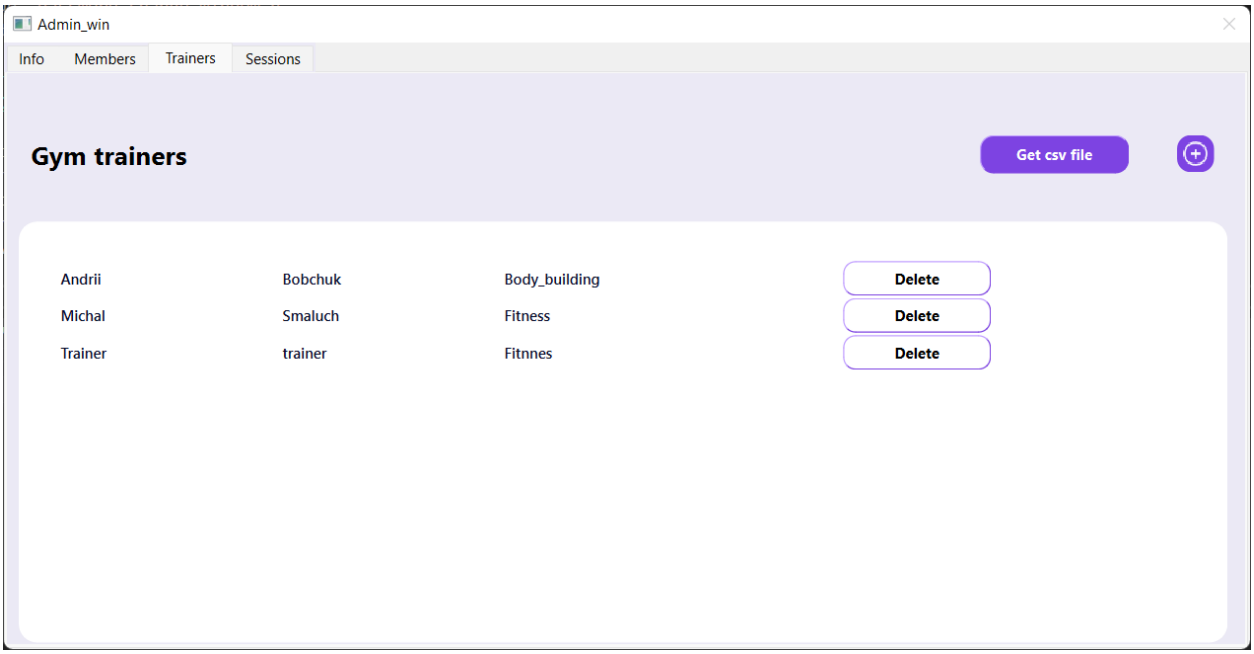


Figure 4 Admin trainers view

- Register a new **Trainer**/**Member** by clicking the add button

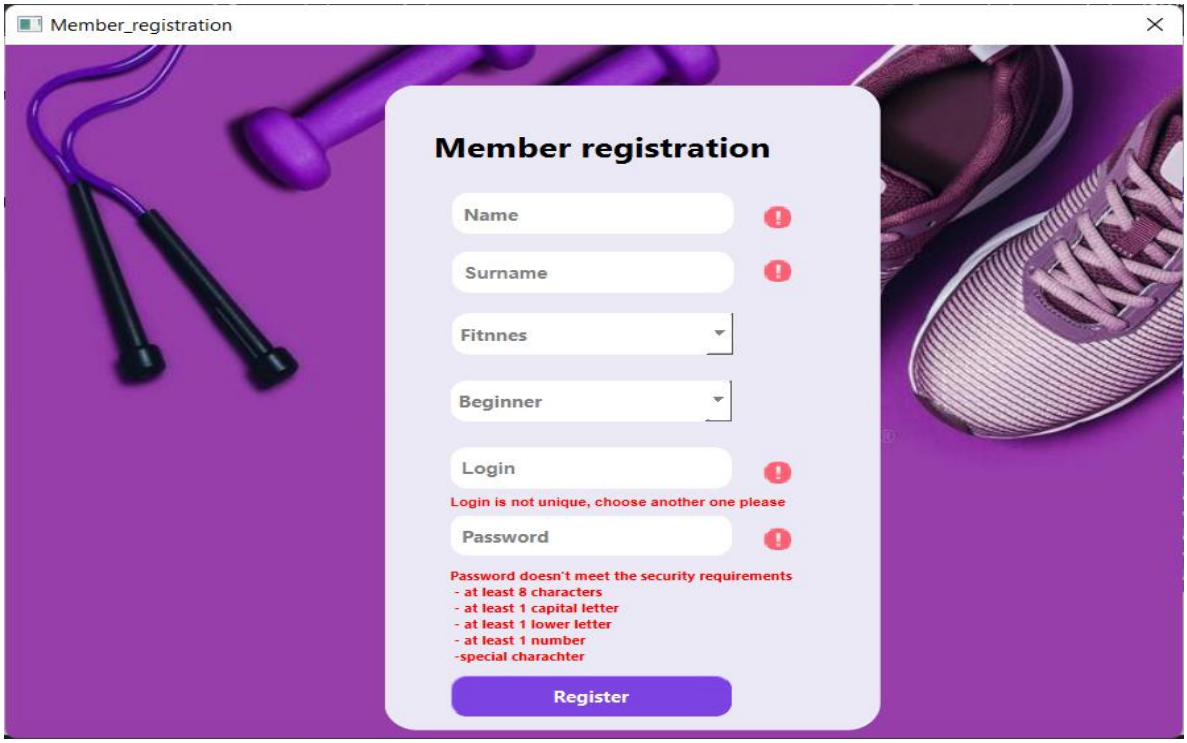


Figure 5 Member registration window

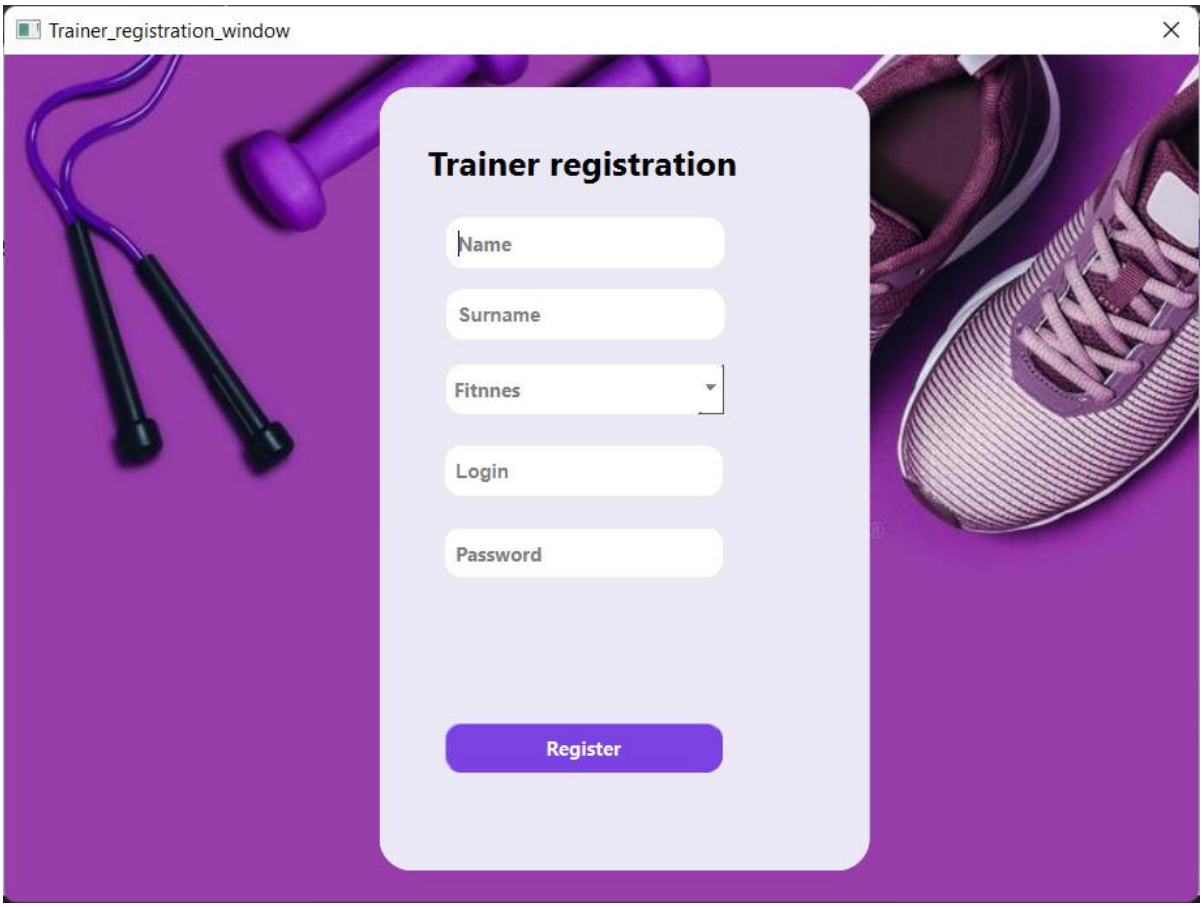
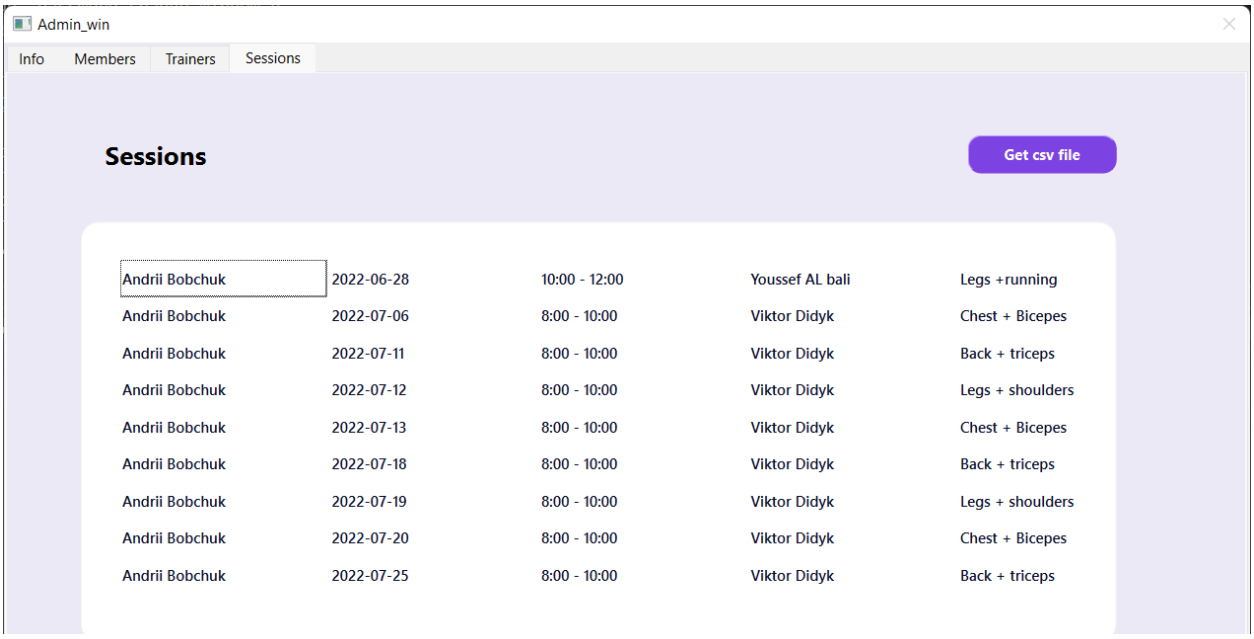


Figure 6 Trainer registration window

- See all the scheduled sessions in the gym stating from today



Andrii Bobchuk	2022-06-28	10:00 - 12:00	Youssef AL bali	Legs +running
Andrii Bobchuk	2022-07-06	8:00 - 10:00	Viktor Didyk	Chest + Bicepes
Andrii Bobchuk	2022-07-11	8:00 - 10:00	Viktor Didyk	Back + triceps
Andrii Bobchuk	2022-07-12	8:00 - 10:00	Viktor Didyk	Legs + shoulders
Andrii Bobchuk	2022-07-13	8:00 - 10:00	Viktor Didyk	Chest + Bicepes
Andrii Bobchuk	2022-07-18	8:00 - 10:00	Viktor Didyk	Back + triceps
Andrii Bobchuk	2022-07-19	8:00 - 10:00	Viktor Didyk	Legs + shoulders
Andrii Bobchuk	2022-07-20	8:00 - 10:00	Viktor Didyk	Chest + Bicepes
Andrii Bobchuk	2022-07-25	8:00 - 10:00	Viktor Didyk	Back + triceps

Figure 7 Admin sessions view

- Get a new versions of Csv files each time, from all the three mentioned views by clicking the **‘Get csv button’**

Trainer	Day	Slot	Member	Sessions
Andrii Bobchuk	27-06-22	8:00 - 10:00	Viktor Didyk	Back + triceps
Andrii Bobchuk	28-06-22	10:00 - 12:00	Youssef AL bali	Legs +running
Andrii Bobchuk	06-07-22	8:00 - 10:00	Viktor Didyk	Chest + Bicepes
Andrii Bobchuk	11-07-22	8:00 - 10:00	Viktor Didyk	Back + triceps
Andrii Bobchuk	12-07-22	8:00 - 10:00	Viktor Didyk	Legs + shoulders
Andrii Bobchuk	13-07-22	8:00 - 10:00	Viktor Didyk	Chest + Bicepes
Andrii Bobchuk	18-07-22	8:00 - 10:00	Viktor Didyk	Back + triceps
Andrii Bobchuk	19-07-22	8:00 - 10:00	Viktor Didyk	Legs + shoulders
Andrii Bobchuk	20-07-22	8:00 - 10:00	Viktor Didyk	Chest + Bicepes
Andrii Bobchuk	25-07-22	8:00 - 10:00	Viktor Didyk	Back + triceps
Andrii Bobchuk	26-07-22	8:00 - 10:00	Viktor Didyk	Legs + shoulders
Andrii Bobchuk	27-07-22	8:00 - 10:00	Viktor Didyk	Chest + Bicepes
Andrii Bobchuk	01-08-22	8:00 - 10:00	Viktor Didyk	Back + triceps
Andrii Bobchuk	02-08-22	8:00 - 10:00	Viktor Didyk	Legs + shoulders
Andrii Bobchuk	03-08-22	8:00 - 10:00	Viktor Didyk	Chest + Bicepes
Andrii Bobchuk	08-08-22	8:00 - 10:00	Viktor Didyk	Back + triceps
Andrii Bobchuk	09-08-22	8:00 - 10:00	Viktor Didyk	Legs + shoulders
Andrii Bobchuk	10-08-22	8:00 - 10:00	Viktor Didyk	Chest + Bicepes
Andrii Bobchuk	15-08-22	8:00 - 10:00	Viktor Didyk	Back + triceps
Andrii Bobchuk	16-08-22	8:00 - 10:00	Viktor Didyk	Legs + shoulders
Andrii Bobchuk	17-08-22	8:00 - 10:00	Viktor Didyk	Chest + Bicepes
Andrii Bobchuk	22-08-22	8:00 - 10:00	Viktor Didyk	Back + triceps
Andrii Bobchuk	23-08-22	8:00 - 10:00	Viktor Didyk	Legs + shoulders
Andrii Bobchuk	24-08-22	8:00 - 10:00	Viktor Didyk	Chest + Bicepes
Andrii Bobchuk	29-08-22	8:00 - 10:00	Viktor Didyk	Back + triceps
Andrii Bobchuk	30-08-22	8:00 - 10:00	Viktor Didyk	Legs + shoulders
Andrii Bobchuk	31-08-22	8:00 - 10:00	Viktor Didyk	Chest + Bicepes
Andrii Bobchuk	05-09-22	8:00 - 10:00	Viktor Didyk	Back + triceps
Andrii Bobchuk	06-09-22	8:00 - 10:00	Viktor Didyk	Legs + shoulders
Andrii Bobchuk	07-09-22	8:00 - 10:00	Viktor Didyk	Chest + Bicepes
Andrii Bobchuk	12-09-22	8:00 - 10:00	Viktor Didyk	Back + triceps
Andrii Bobchuk	13-09-22	8:00 - 10:00	Viktor Didyk	Legs + shoulders
Andrii Bobchuk	14-09-22	8:00 - 10:00	Viktor Didyk	Chest + Bicepes
Andrii Bobchuk	19-09-22	8:00 - 10:00	Viktor Didyk	Back + triceps
Andrii Bobchuk	20-09-22	8:00 - 10:00	Viktor Didyk	Legs + shoulders

Figure 8 Csv file view

- And finally, Logout from the main Info window.

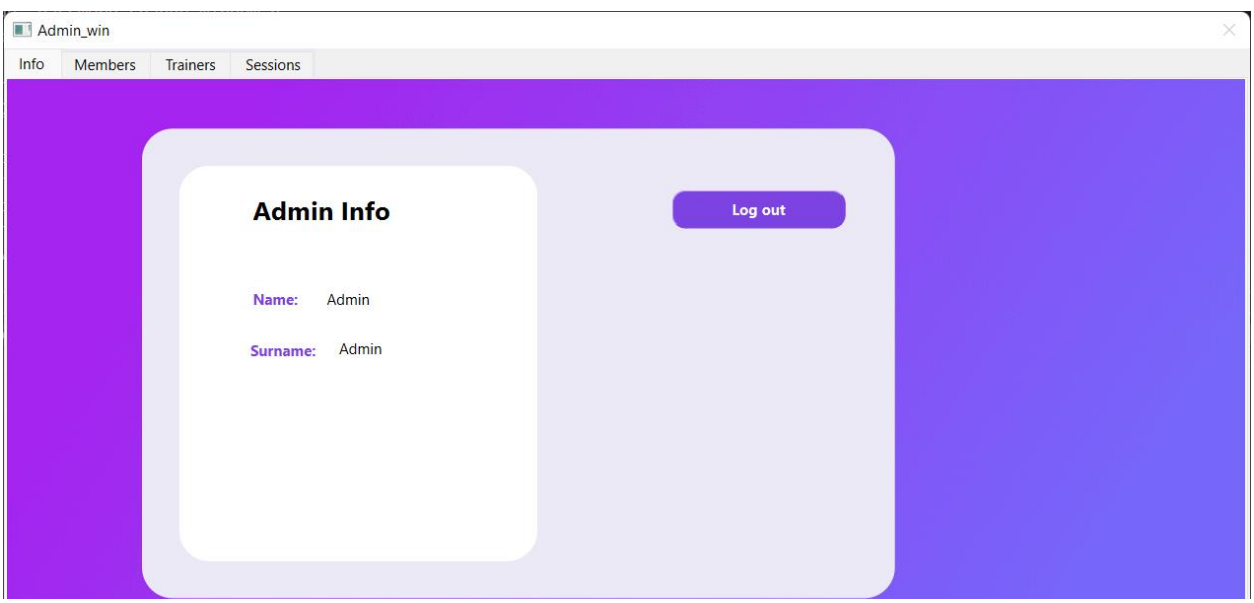


Figure 9 Admin Info view

The **member** starts the session by either login or register *(see Figure 5 Member registration window)*

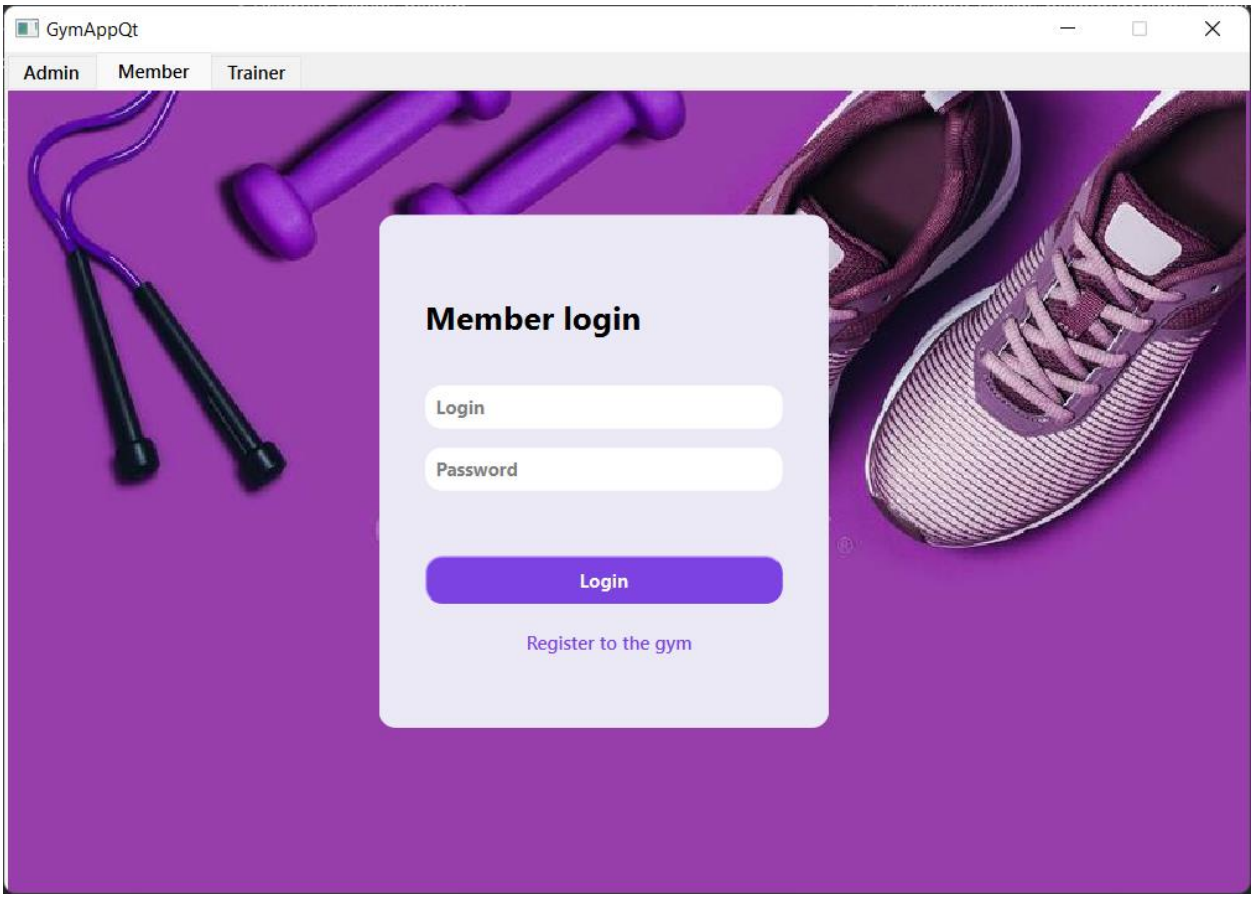


Figure 10 Member login

- After login in the **Member** will firstly see the main info window where he can see his information and can change his password or resign from the gym immediately .

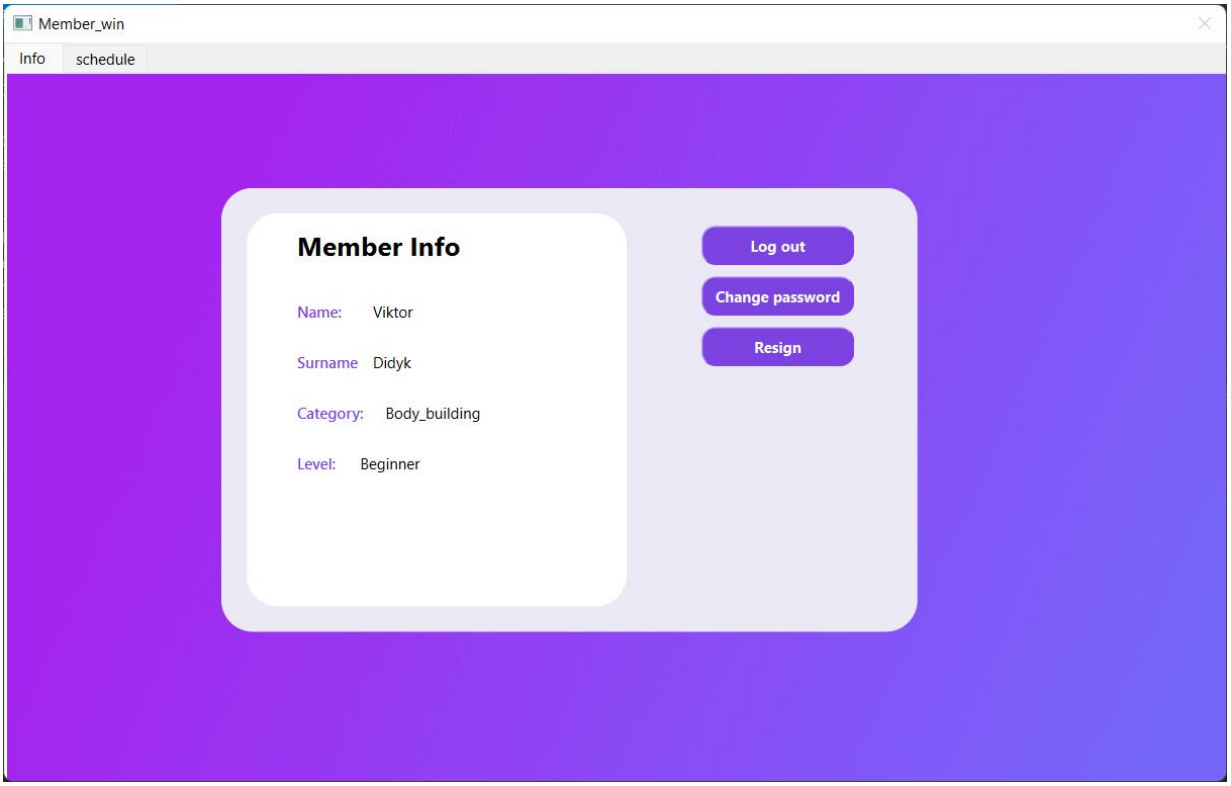


Figure 11 Member info window

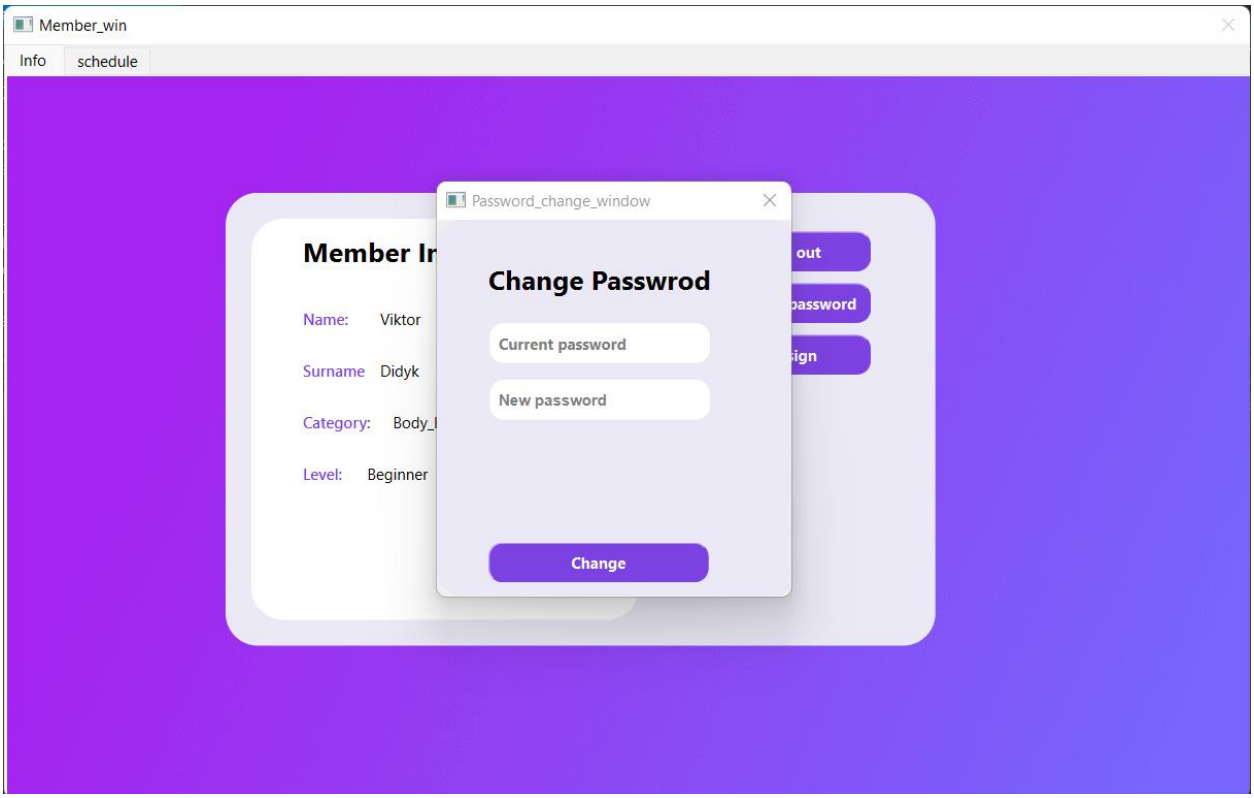


Figure 12 Change password window

- See his scheduled workouts and being able to delete one or more sessions **(The program will not allow deletion of sessions scheduled for the current date).**

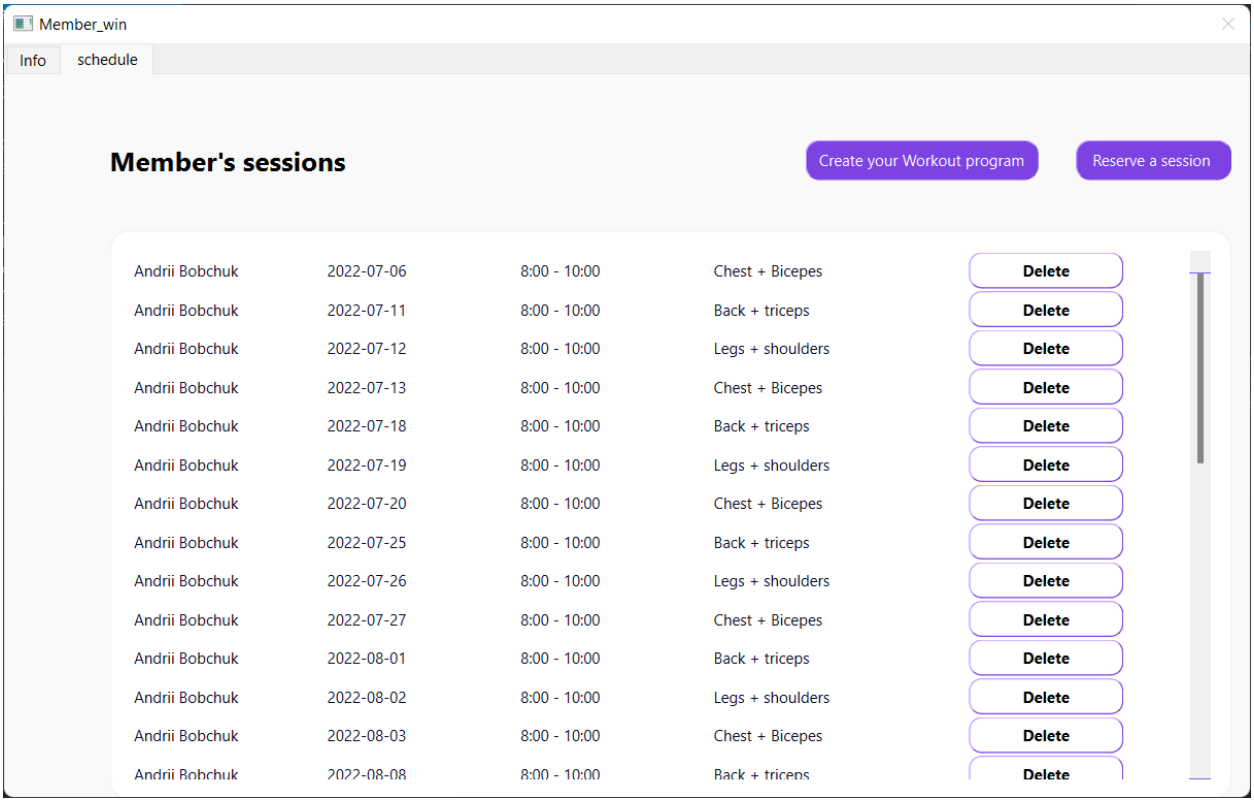


Figure 13 Member sessions view

- Reserve a private session by clicking the **'Reserve a session'** button
  - The Member is able to choose a trainer to workout with based on his category.
  - The member is able to choose a slot staring from tomorrow up to one year from the current date.
  - The program checks if both of the Trainer and Admin are available at the chosen time slot.

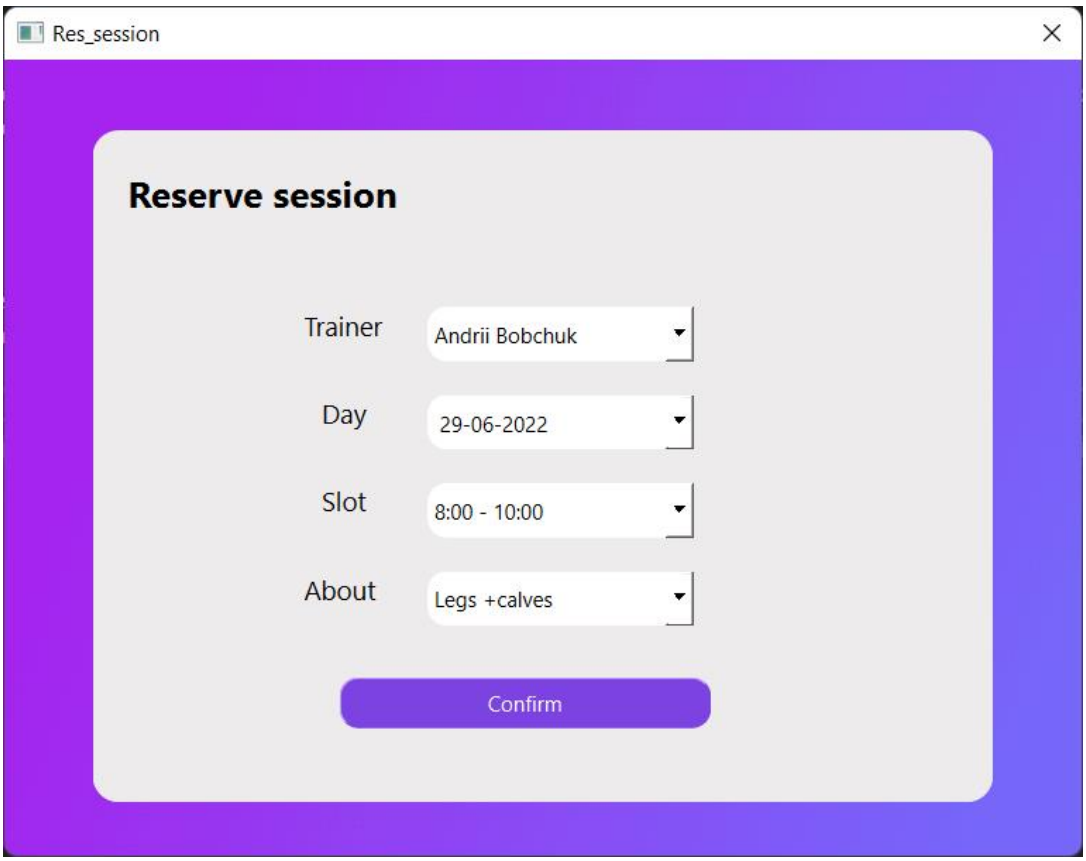


Figure 14 Private reservation view

- Reserve a special workout program for a period of time **(1 month, 3 months…)**
  - The program will get a list of available programs from the database based on the member *category* and *level*.
  - The Member is able to choose the trainer in his category
  - After selecting the desired program from the popup window, the program should be visible in the main reservation table with the ability to:
  - Choose the starting week, and the program will warn the user if the chosen sessions are not in the same week, two sessions are in the same day and if the trainer will not be available in one of the sessions.
  - If none of that happened the sessions will be reserved for the chosen period.
  - The following pictures demonstrates the process by steps:

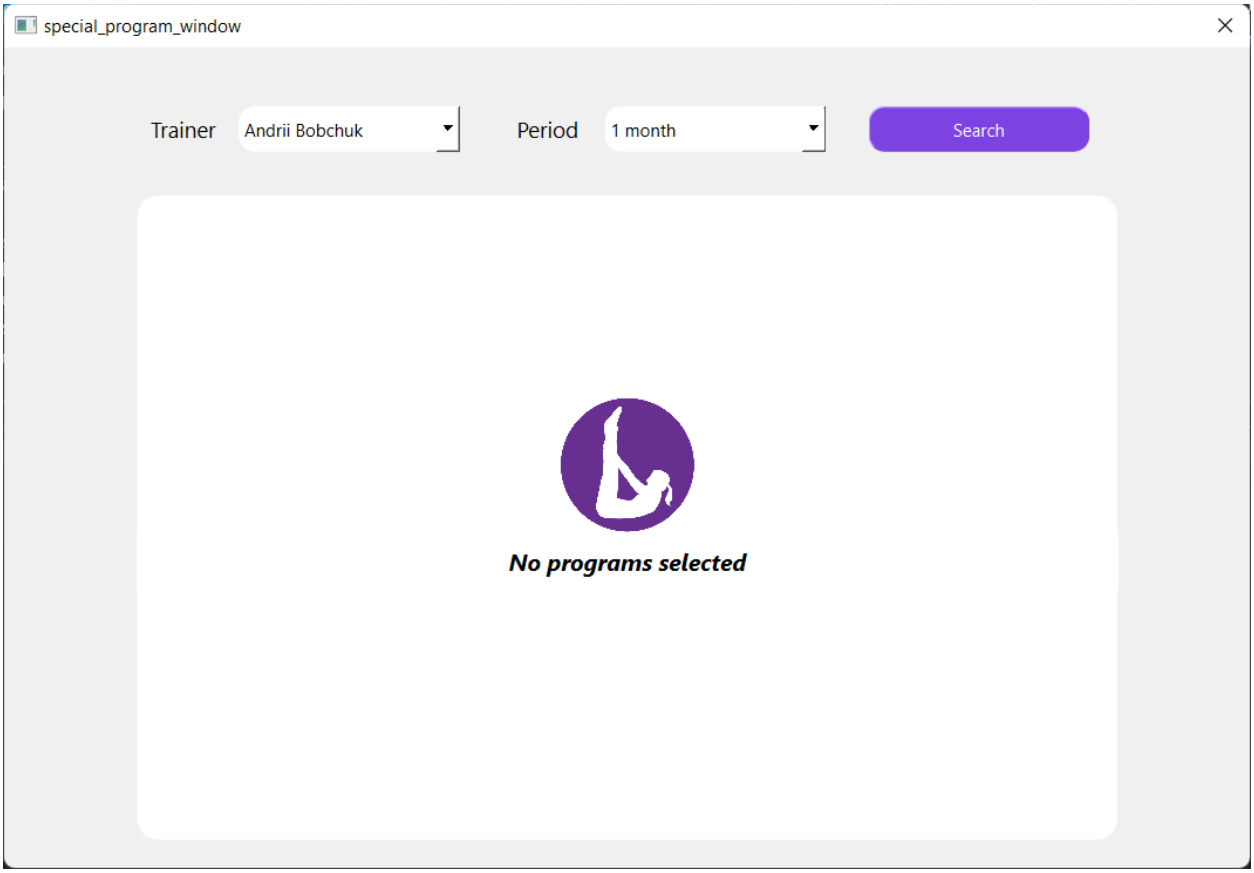


Figure 15 Program reservation window (1)

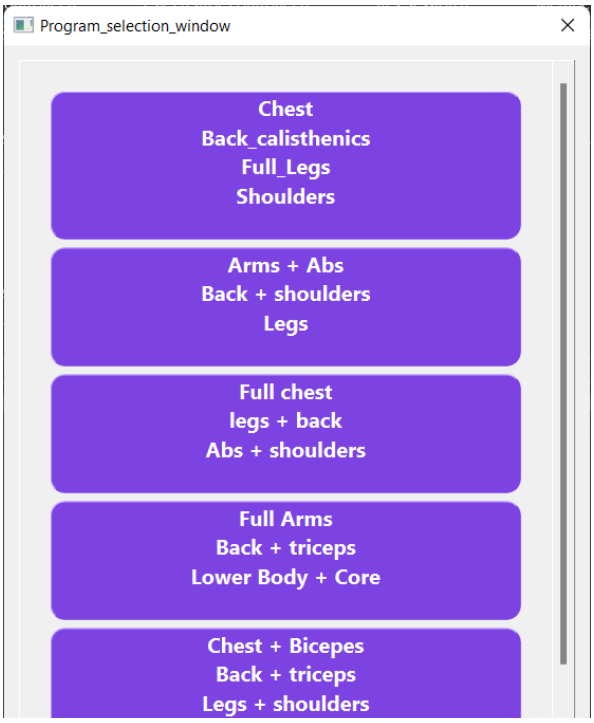


Figure 16 Program selection window

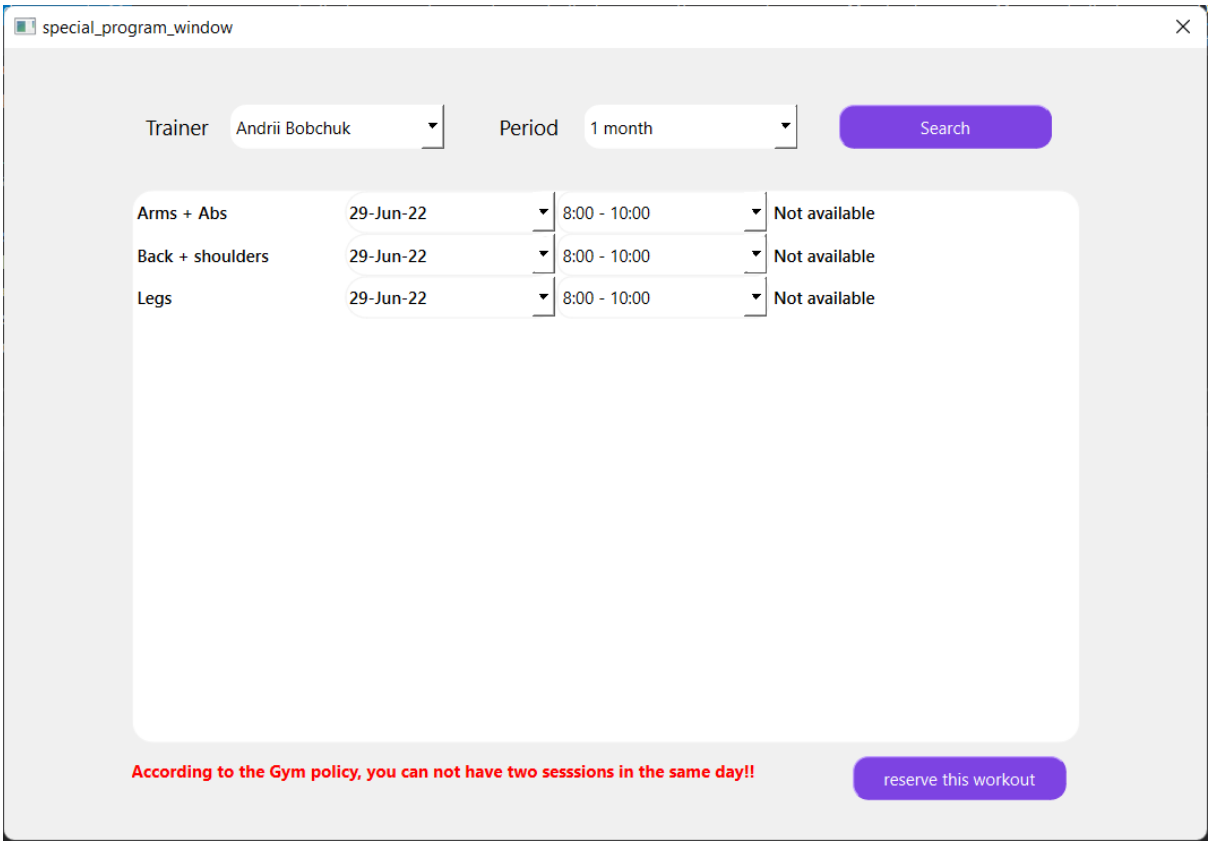


Figure 17 Program selection window (2)

Finally, for the Trainer console has almost the same functionalities as the **Member** view excluding the reservation process.

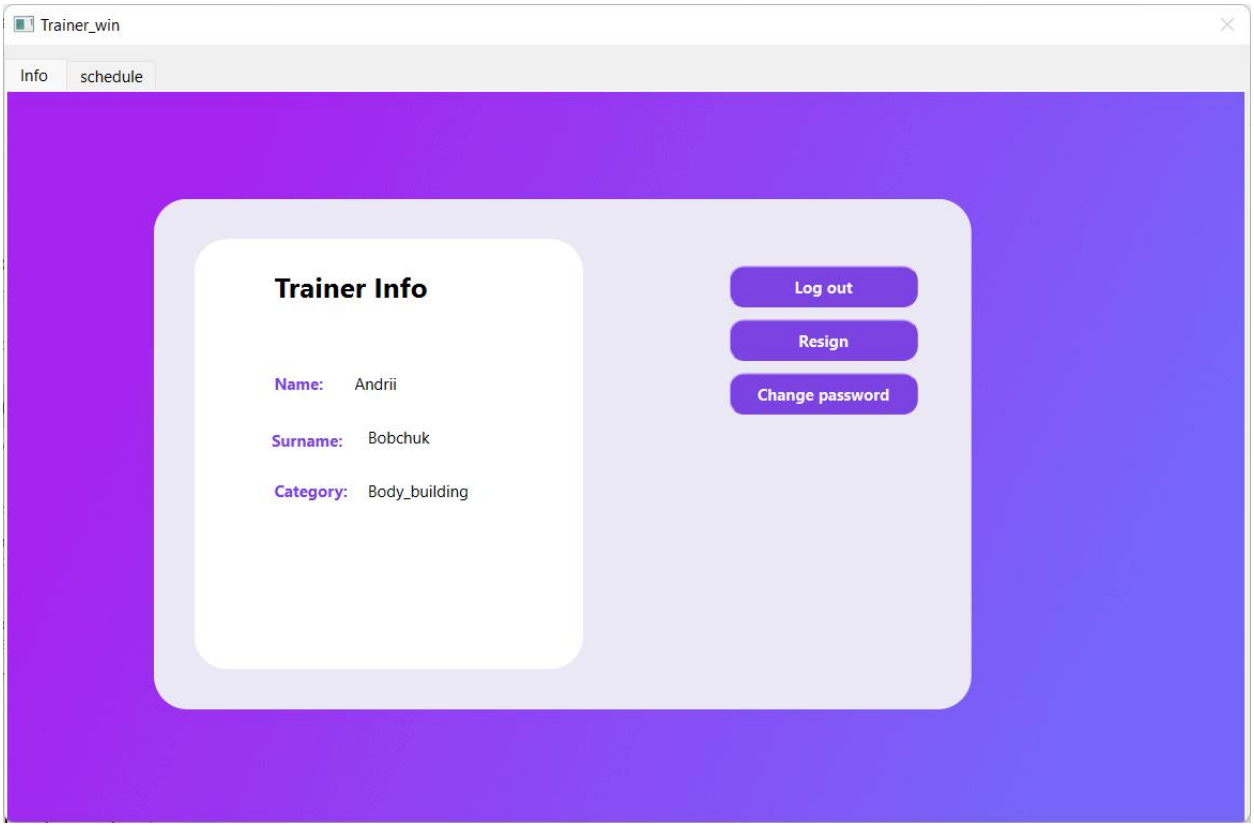


Figure 18 Trainer Info view

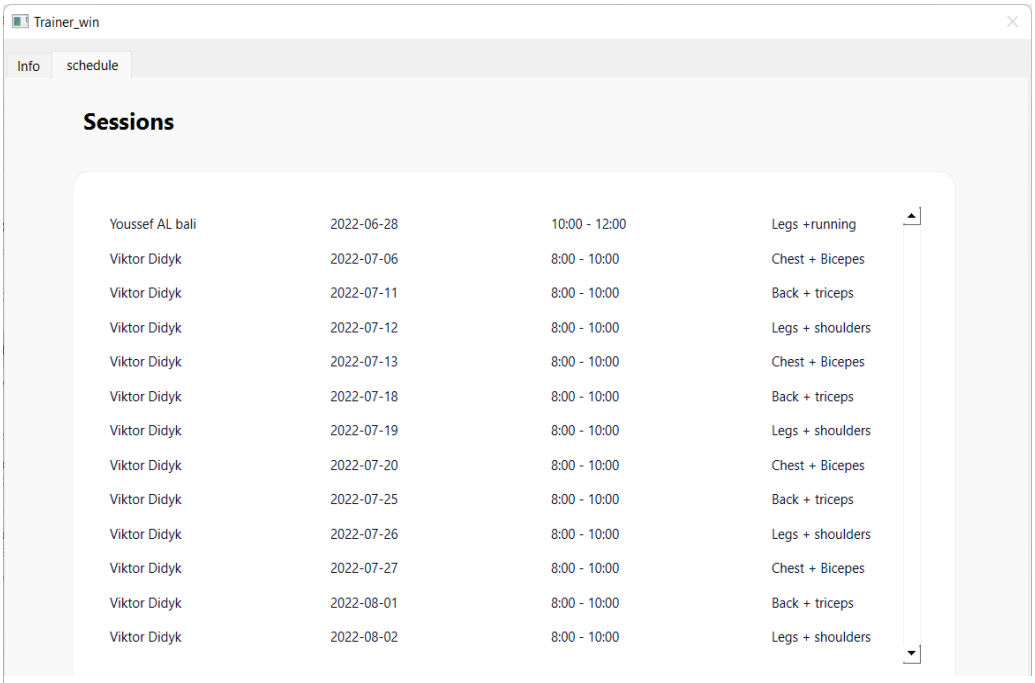


Figure 19 Trainer sessions view

INTERNAL SPECIFICATION

Please note that the classes descriptions, class hierarchy diagram and important functions are mentioned and described in the doxygen file appended below.

MVC PATTERN

In order to maintain the codebase and the features of the application The MVC design pattern was used and the following diagram demonstrates an example on how the program behaves under this pattern for the Admin Entity

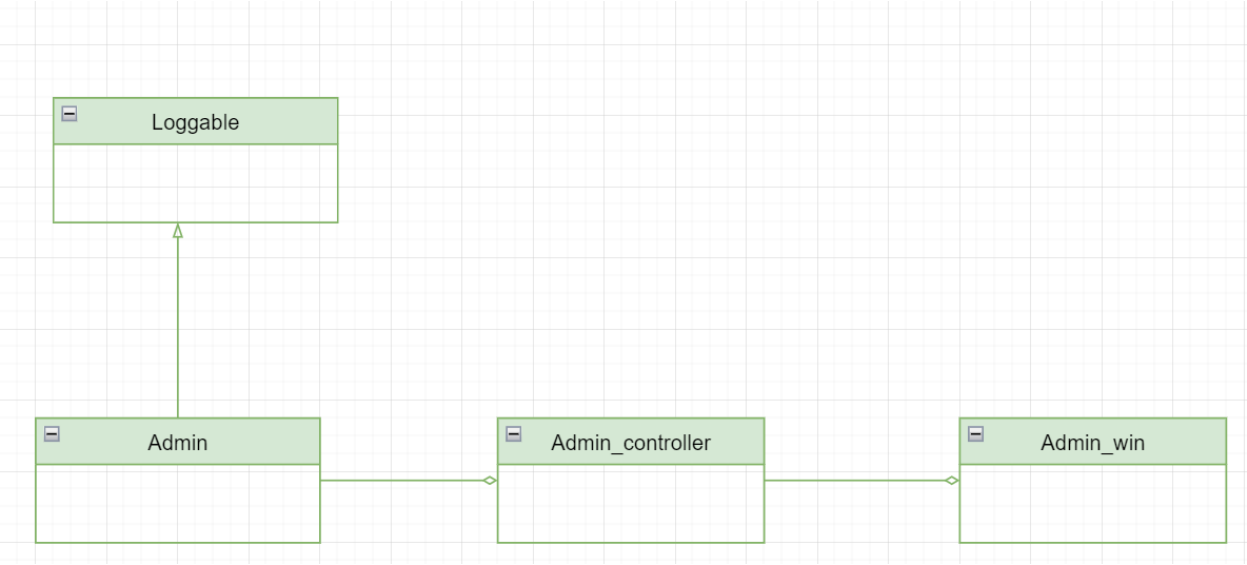


Figure 20 Admin MVC pattern

Basically, a view can access just the controller not the model and depends on the function the controller can either access its Model or just perform some operation within the class itself.

The same is implemented for all other views (***see doxygen file***) excluding the small ones like Update_member_window or Change_user_password since they can use just the parent controller.

DATABASE_MANAGER

The class handles all the operations performed on the database.

Its functionality is needed always during run time hence it was implemented as a **Singleton** It holds a pointer to the Database class since it always performs operations on this object during runtime.

```
class Database_manager {
    static Database_manager* s_instance;
    std::shared_ptr<Database> database;

protected:
    Database_manager() { ... }
public:

    std::unordered_map<int, std::shared_ptr<Trainer>> get_trainers_from_db();
    std::unordered_map<int, std::shared_ptr<Member>> get_members_from_db();
    std::shared_ptr<Admin> get_admin_from_db();
    std::vector < std::string > get_member_logins();
    std::vector < std::string > get_trainers_logins();
    std::unordered_map<std::string, int > get_trainers_names_ids();
    std::unordered_map<int, std::shared_ptr<Trainer>> get_trainers_by_category(std::string category);
    static Database_manager* getInstance();

    void create_new_member(std::string name, std::string surname, std::string category, std::string login, size_t password, std::string level);
    void create_new_trainer(std::string name, std::string surname, std::string category, std::string login, size_t password);

    void delete_member(int _id);
    void delete_trainer(int _id);

    bool update_data();

    std::map<Specific_slot, std::shared_ptr<Main_session>> get_private_sessions_from_db() { return database->get_sessions();};

    std::shared_ptr<session> add_ids_to_private_sessions(Specific_slot &s, int member_id, int trainer_id, std::string description);
    void delete_ids_from_sessions(Specific_slot& s, int member_id);

    std::pair<int, int> total_members_trainers();

    std::vector<std::shared_ptr<Workout_program>> get_body_building_program(std::string _level);

    std::vector<std::shared_ptr<Workout_program>> get_fitnnes_program(std::string _level);

    void reservartion_cleanup_member(int id);
    void reservartion_cleanup_trainer(int id);
};
```

Figure 21 Database_mamnger Class

RESERVATION PROCESS

In order to store, update and make the reserved sessions, a struct Specific_slot was created and its objects were used as a Map key of the schedule map in the database and entities.

I decided to use map in order to not allow duplicates of slots, as well to store the map in order so the user will see the sessions scheduled in the table from the up to bottom starting from the earliest date.

```
struct Specific_slot {
    boost::gregorian::date sample_date;
    int slot{};
    int get_day() { return sample_date.day_of_week(); }
    Specific_slot() = default;
    Specific_slot(boost::gregorian::date _date, int _slot) {
        sample_date = _date;
        slot = _slot;
    }
    friend bool operator ==(const Specific_slot& a, const Specific_slot& b) {
        if (a.sample_date == b.sample_date && a.slot == b.slot)
            return true;
        else
            return false;
    }
    bool operator<(const Specific_slot& a) const {
        if (a.sample_date == sample_date)
            return a.slot > slot;
        return a.sample_date > sample_date;
    }
};
```

Figure 22 Specific_slot struct

– CONSTANTS.H

I made a file containing global constants to avoid hardcoding strings and numbers. This makes it simple to easily change things like the database file paths, slots warning and confirmation strings.

```
static const  fs::path MY_PATH = (LR"(.\Database)");
static const fs::path ADMIN_PATH = (LR"(.\Admin_files)");
static const fs::path SESSIONS_PATH = ADMIN_PATH /"sessions_files";
static const fs::path TRAINERS_PATH = ADMIN_PATH /"trainers_files";
static const fs::path MEMBERS_PATH = ADMIN_PATH /"members_files";
static const fs::path TRAINERS_FILE_PATH = MY_PATH / "trainers.json";
static const fs::path MEMBERS_FILE_PATH = MY_PATH / "members.json";
static const fs::path SESSIONS_FILE_PATH = MY_PATH / "sessions.json";
static const fs::path ADMIN_FILE_PATH = MY_PATH / "admin.json";
static const fs::path PROGRAMS_FILE_PATH = MY_PATH / "gym_programs.json";
```

Figure 23 Constants (1)

```
static const QString DAYS_DUPLICATE_FOUND = "According to the Gym policy, you can not have two sesssions in the same day!!";
static const QString PROGRAM_RES_ALMOST_DONE = "Please note that all your workouts schedules will get removed since you signed for a special
static const QString UNSUCCESFUL_LOGIN = "Username or password are incorrect! \nPlease try Again";
static const QString session_reservation_done = "Session reserved succefully \nPlease be at the exact time";
static const QString session_booked = "Session already booked \nPlease choose another time slot";
static const QString different_weeks_found = "Please choose the same starting week!";
static const QString PASSWORD_RESET_CONFIRMATION = "Are you sure you want to reset this member's password";
static const QString PORHIBITED_SESSION_DELETION = "The session is scheduled for today you can not delete it!";
static const QString PASSWORD_SECURITY_REQUIREMNETS = "Password doesn't meet the security requirements\n"
                                                      " - at least 8 characters\n"
                                                      " - at least 1 capital letter\n"
                                                      " - at least 1 lower letter\n"
                                                      " - at least 1 number\n"
                                                      " -special charachter";
static const QString LOGIN_UNIQUE_REQUIREMNET = "Login is not unique, choose another one please";
static const QString USER_DELETION_CONFIRMATION = "Are you sure you want to delete this user from the gym ?";
static const QString USER_UPDATE_CONFIRMATION = "Are you sure you want to update this user";
static const QString UPDATE_DATA_FAILED = "Your data was not updated, somthing went wrong";
static const QString REGISTRATION_DONE = "Registration done successfully";
static const QString WARNING_IMAGE_SOURCE = " < img src = 'warning.png' / > ";
static const QString PROGRAM_NOT_FOUND = "We couldn't find a program for you";
static const QString ADD_ICON_IMAGE = "Add_icon(2).png";
static const QString PROGRAM_FOUND{ "We found a program for you, Click yes to confirm!\n" };
static const QString BUTTON_DELETE_STYLE = "#bt_delete{ \nborder-style: outset; \nborder-radius: 10px; \nborder-color:#7D43E2; \nborder-widt
```

Figure 24 Constants (2)

DATABASE

The following pictures demonstrate the schemes of my database:


```
{
  "Members" :
  [
    {
      "Level" : "Beginner",
      "Sport_option" : "Body_building",
      "id" : 1,
      "login" : "",
      "name" : "Viktor",
      "password" : 14695981039346656037,
      "surname" : "Didyk"
    },
    {
      "Level" : "Beginner",
      "Sport_option" : "Fitnnes",
      "id" : 9,
      "login" : "New",
      "name" : "new",
      "password" : 14695981039346656037,
      "surname" : "new"
    },
    {
      "Level" : "Intermediate",
      "Sport_option" : "Fitness",
      "id" : 2,
      "login" : "yabali123",
      "name" : "Youssef",
      "password" : 2305781866815333943,
      "surname" : "AL BALI"
    },
```


Figure 25 Members in Json

```
{
  "Trainers": [
    {
      "Category": "Body_building",
      "id": 5,
      "login": "trainer",
      "name": "Andrii",
      "password": 14695981039346656037,
      "surname": "Bobchuk"
    },
    {
      "Category": "Fitness",
      "id": 6,
      "login": "Michal",
      "name": "Michal",
      "password": 14079092472701498582,
      "surname": "Smaluch"
    },
    {
      "Category": "Fitnnes",
      "id": 10,
      "login": "trainer1",
      "name": "Trainer",
      "password": 14695981039346656037,
      "surname": "trainer"
    }
  ]
}
```


Figure 26 Trainers in Json

```
{
  "Private_sessions" :
  [
    {
      "Date" : "2022-06-28",
      "Slot" : 2,
      "member_trainer_ids" :
      [
        {
          "description" : "Legs +running",
          "member_id" : 2,
          "member_name" : "Youssef AL bali",
          "trainer_id" : 5,
          "trainer_name" : "Andrii Bobchuk"
        }
      ]
    },
    {
      "Date" : "2022-06-28",
      "Slot" : 5
```


Figure 27 Sessions in Json

OOP TECHNIQUES AND PATTERNS USED

- Singleton Design Pattern: Database_manager ***(Figure 21)***
- Encapsulation: Every class is equipped with all necessary getters & setters.
- Inheritance: Evident in the models and controllers.
    - Trainer and Member inherits from User class.
    - Special_program reservation_controller inherits from the Base_reservation_controller class.
    - …
- Polymorphism: Evident in the User and User_controller
    - **virtual** void update_full_name_sessions()= 0;
    - **virtual** void delete_user() = 0;
    - …
- Abstraction

- Chain of Responsibility Design Pattern: communication between controllers and views.

TECHNIQUES COVERED DURING THEMATIC CLASSES

Threads: for the background database access (reading/updating), also used with functions that can be executed concurrently.

- Example

```
inline bool Database::read_data()
{
    bool members_read, trainers_read, admin_read, sessions_read;
    std::thread th_mbs_rd(&Database::read_members, this, std::ref(members_read));
    std::thread th_tr_rd(&Database::read_trainers, this, std::ref(trainers_read));
    std::thread th_ad_rd(&Database::read_admin_data, this, std::ref(admin_read));
    std::thread th_ses_rd(&Database::read_private_sessions, this, std::ref(sessions_read));
    th_mbs_rd.join();
    th_tr_rd.join();
    th_ad_rd.join();
    th_ses_rd.join();
    if (members_read && trainers_read && admin_read && sessions_read) {
        assign_sessions();
        return true;
    }
    return false;

}
```

Ranges: for an easier and faster loop iteration, used with std algorithms like (adjacent_find, copy_if, find_if…)

- Example:

```
std::ranges::copy_if(existing_programs.begin(), existing_programs.end(),
    std::back_inserter(fit_programs), [&_level](std::shared_ptr<Workout_program> &program) {
        return program->get_program_level() == _level;
    }
);
return fit_programs;
```

Filesystem: Used along with reading and updating the database, additionally used in Csv_creator_class to generated new versions of files each time.

- Example

```
inline fs::path Csv_creator_controller::get_path(fs::path dir_path, std::string filename)
{
    fs::path path_trial = dir_path /( filename + ".csv");
    if (!fs::exists(path_trial)) {
        return path_trial;
    }
    else {
        int file_number = get_file_number(dir_path);
        dir_path /= (filename + "("+std::to_string(file_number) + ").csv");
        return dir_path;
    }
}
```

Regular expressions: Used for checking the password security requirements, and along with filesystem in Csv_creator_class.

- Example

```
inline bool Validator::Check_password_security(std::string user_password)
{
    std::regex upper_case_expression{ "[A-Z]+" };
    std::regex lower_case_expression{ "[a-z]+" };
    std::regex digit_expression{ "[0-9]+" };
    std::regex eight_charachters{ ".{8,}" };
    std::regex special_charachters{ R"([!"\\$%&'()+,-./:;=#@?[\]^_`{|}~*])" };

    bool upper_case = std::regex_search(user_password, upper_case_expression);
    bool lower_case = std::regex_search(user_password, lower_case_expression);
    bool number_case = std::regex_search(user_password, digit_expression);
    bool eight_char = std::regex_search(user_password, eight_charachters);
    bool sp_char = std::regex_search(user_password, special_charachters);
    if (upper_case && lower_case && number_case && eight_char && sp_char)
        return true;
    else
        return false;
}
```

TESTING AND DEBUGGING

During the development I was constantly manually testing the program. Here is the list of some of the detected and fixed bugs:

THREADS ACCESING THE SAME JSON OBJECT

After implementing the Database update function to update JSON data concurrently.

The program was crushing most of the time, and I figured out that the threads were assigning the JSON object at the same time, that's exactly when I decided to separate the JSON files into separate ones in order to update them concurrently and safely.

Each function uses a separate file and JSON object.

```
bool members_updated, trainers_updated, admin_updated, sessions_updated;
std::thread th_mbs_upd(&Database::update_members, this, std::ref(members_updated));
std::thread th_tr_upd(&Database::update_trainers, this, std::ref(trainers_updated));
std::thread th_ad_upd(&Database::update_admin, this, std::ref(admin_updated));
std::thread th_ses_upd(&Database::update_private_sessions, this, std::ref(sessions_updated));
th_mbs_upd.join();
th_tr_upd.join();
th_ad_upd.join();
th_ses_upd.join();
if (members_updated && trainers_updated && admin_updated && sessions_updated) {
    return true;
}
return false;
```

PARENT WINDOW REFRESHING

The issue was that when the pop_up window of some parent window changes an information in the model, after closing it changes are not visible in the parent since the data is not reloaded.

The issue lasted for a long period, but it was fixed at the end.

After reading the documentation, I found out that the child window is able to send a signal to its parent window and by that the parent can execute its methods for reloading the data.

The following example consists of the Member window (parent) and the reservation_window (child)

```
void Member_win::receiveSignalChanged()
{
    set_sessions();
}
```

In reservation_window Constructor:

```
connect(this, SIGNAL(sendSignalChanged()), parent, SLOT(receiveSignalChanged()));
```

In reservation_window::on_confirm_button_clicked():

```
emit sendSignalChanged();
this->hide();
parentWidget()->show();
```