**Документация на проект  
 „Музикална библиотека“ по СД**

*2 курс, 1 семестър, 2020/2021 година*

*Изготвила: Радина Нунева*

1. ***Тема на проекта***

Проектът „Музикална колекция“ е реализация на музикална библиотека, която съхранява данни за потребители, песни и създадени плейлисти. Диалогът в конзолния прозорец е опростен за максимално улеснение на потребителите. Потребителските данни, както и характеристиките на песните се запазват между две стартирания на програмата, тъй като се съхраняват във файлове, намиращи се в постоянната памет. Програмата разчита на различни функционалности, всяка от които е тествана, така че потребителските грешки не са пречка за правилното изпълнение. Най-съществената опция, която музикалната библиотека предлага, е генерирането на плейлист по желани критерии. Това се случва бързо и удобно, а от потребителя се изисква единствено да въведе критериите си.

1. ***Основни обекти***
   * ***потребители***
     + потребителско име
     + парола
     + пълно име
     + дата на раждане
     + любими жанрове
     + плейлисти

Всеки, регистрирал се в системата, разполага с гореизброените данни, които може да променя и запазва повече от веднъж. По подразбиране, след регистрация, се запазват избраните потребителско име и парола, а на останалите данни се задават стойности по подразбиране. Потребителят след това има възможност да ги смени по свое осмотрение.

* + ***песни***
    - име
    - изпълнител
    - жанр
    - албум
    - година на издаване
    - рейтинг

Всяка песен, въведена и съхранена в системата, съдържа гореспоменатите характеристики. За добавяне на песен в системата, потребителят трябва да въведе всички данни коректно.

1. ***Функционалности***
   * ***първоначалнно***
     + “help” меню
     + вход
     + регистрация
     + изход

За използването на тези команди, не се изисква регистрация или активен профил. Всеки има достъп до тях.

* + ***след успешен вход***
    - промяна на данни (вкл. любими жанрове)

След използването на тази команда, потребителските промени не се запазват автоматично. Въпреки това, потребителят има възможността да обнови данните във файла с помощта на команда „save\_user\_data“

* + - изход (от профил)

При този избор автоматично се обновяват последните промени, които са направени по потребителския профил.

* + - добавяне на песен
    - задаване на рейтинг на песен

Един потребител не може да гласува повече от веднъж за една песен, независимо кога е извършено гласуването.

* + - генериране на плейлист по избор от критерии:
      * рейтинг по-голям от:
      * включване на жанрове
      * изключване на жанрове
      * само любими жанрове
      * преди година
      * от година
      * след година

Целият процес на генериране на плейлист е автоматизиран. Потребителят е небходимо единствено да въведе критериите си, които ако са повече от един, може да раздели с логическите връзки **&**(и) и |(или).

* + - записване на плейлист под име

Записване на новосъздаден плейлист, както и промяна на името на вече съществуващ такъв.

* + - зареждане на плейлист по име
    - показване на информация за всички песни от зареден плейлист

1. ***Класове, атрибути***
   * ***User***
     + **string** username
     + **string** password
     + **string** full\_name
     + **string** birth\_date
     + **unordered\_set<std::string>** fav\_genres
     + **unordered\_set<std::string>** voted\_songs
     + **unordered\_map<std::string, Playlist>** playlists

voted\_songs индикира за кои песни потребителят вече е дал своя глас. По този начин се ограничава възможността за гласуване повече от веднъж за една песен от един и същ потребител. Стойностите на този атрибут, се съхраняват също във файла и се пазят, докато потребителските данни не бъдат изтрити от системата.

* + ***Song***
    - **string** name
    - **string** artist
    - **string** genre
    - **string** album
    - **size\_t** release\_year
    - **pair <size\_t, size\_t>** rating

За да е математически коректно изчислението на рейтинга, не може да се съхранява единствено последната му пресметната стойност. Необходимо е да се пазят, както броят на гласовете, така и техният сбор. При необходимост от използване на конкретният рейтинг на песента, се пресмята тяхното частно.

* + ***Playlist***
    - **string** name
    - **const size\_t** max\_size = 20;
    - **list <string>** songs

Този клас не съдържа самите песни, а единствено списък от имената им. По този начин не се използва допълнително място за повторно съхранение на песни, които вече съществуват в системата.

* + ***Library***
    - **User** curr\_user
    - **bool** logged\_u = false
    - **Playlist** curr\_playlist
    - **bool** loaded\_pl
    - **unordered\_map <string, Song>** all\_songs

Това е най-главният клас на системата. В него се съдържат всички основни функции за диалог с потребителя. Освен основните методи, са включени и helper функции. По този начин се осъществява разделение на кода. В главните функции наблюдаваме всички помощни средства за осъществен контакт с потребителя (извеждащи се съобщения при извикване на функцията, успешно изпълнение или възникнали грешки), а в helper функциите – фундаменталните действия, които програмата извършва.

* + ***Terminal***

Целта на този клас е да задържи main-а максимално чист и пригледен. Клас Терминал има единствен метод – save(), който обработва входната команда на потребителя и обменя информацията с клас Library.

1. ***Работа с файлове***
   * ***избор на файлов формат за съхранение на*** 
     + *потребители*
       - ИЗБОР: **txt file**
       - триене/добавяне/презаписване на потребител  
         (презаписване на целия файл)
       - всеки атрибут на нов ред
     + *песни*
       - ИЗБОР: **txt file**
       - данните изцяло се прочитат от файла
       - при изход данните се презаписват във файла
       - няма директна работа с файла
       - няма нужда от избирателен/директен достъп до съдържанието на файла
       - човешки четими данни
       - обхождане с последователен достъп
       - единен формат на данни от тип “Song”
   * ***функции***
     + *потребители*
       - записване във файла
         * …\n (username)
         * …\n (password)
         * …\n (full name)
         * \_.\_.\_\n (birth date)
         * …**+**…**+**…\n (fav genres)
         * …^…^…\n (voted\_songs)
         * name|…**+**…**+**…~name|…**+**…**+**…\n (playlists)
       - прочитане от файла
         * >>username
         * >>password
         * getline (full name)
         * >>birth date
         * getline (string) ->fav genres
         * getline (string) ->voted\_songs
         * geltine(string) ->playlists
       - промяна във файла
         * презаписване на целия файл във втори, след което първият се изтрива, а вторият се преименува
         * if (line == искаме промяна) -> промяна
       - търсене във файла
         * всеки обект – 7 реда
     + *песни*
       - записване във файла
         * …\n (name)
         * …\n (artist)
         * …\n (genre)
         * …\n (album)
         * ----\n (release year)
         * \_ \_\n (rating)
       - прочитане от файла
         * getline(name)
         * getline(artist)
         * getline(genre)
         * getline(album)
         * >>release\_year
         * getline(string) -> pair<int,int> rating
       - промяна във файла
         * презаписване на целия файл
       - търсене във файла
         * всеки обект – 6 реда
2. ***Функции / методи***
   * ***User***
     + ***конструктори и помощни функции***
     + ***сетъри/гетъри***
     + ***информативни методи***
   * ***Song***
     + ***конструктори и помощна функция***
     + ***сетъри/гетъри***
     + ***информативен метод***
   * ***Playlist***
     + ***конструктори и помощна функция***
     + ***сетъри/гетъри***
     + ***метод за зареждане на данни от файл***
   * ***Library***
     + ***конструктори***
     + ***сетъри/гетъри***
     + ***информативни методи***
     + ***команди:***
       - ***welcome()***
       - ***help()***
       - ***exit()***
       - ***sign\_in()***
       - ***sign\_up()***
       - ***change\_data()***
       - ***save\_user\_data()***
       - ***log\_out()***
       - ***add\_song()***
       - ***rate\_song()***
       - ***generate\_playlist()***

Тази функция използва следния алгоритъм. Потребителския “expression” (… & … | … & …) се разделя на “statement”-и ( ... & …), които се разбиват на команди. Последователно се проверяват команда по команда, като ако някоя не е изпълненя, целият statement става 0 и се проверява следващият, ако има такъв. При намиране на statement, който е 1, целият expression връща set с избраните песни, за да се сортират по име, след косето биват добавени във финален вид в лист, който е част от новосъздадения плейлист. По този начин потребителския вход е разбит на малки части и обработен последователно.

* + - * ***save\_playlist()***
      * ***load\_playlist()***
      * ***show\_all\_info()***
    - ***и съответни хелп-функции към тях***
  + ***Terminal***
    - ***start()***

Целта на този клас е да улесни и опрости използването на библиотеката. Единствено се изисква използването на функцията му, която извършва обмена на данни между потребителя и методите в клас Library

1. ***Структури от данни***
   * ***Основни***
     + ***unordered\_map***
     + ***unordered\_set***
     + ***set***
     + ***list***
   * ***Мотивация за избор***

При започване на проекта поставих пред себе си 3 варианта за структура от данни, която да използвам за съхранение на всички песни в системата - B tree, AVL tree и Hash Table. Тъй като данните се съхраняват в постоянната памет и темата на проекта предполага работа с големи количества данни (на песни и потребители), установих, че най-добрият избор за работа е B Tree, тъй като работата с такова дърво е много бърза. Не случайно то се използва и за основа на бази от данни. Тъй като предметът е структури от данни, а не бази от данни, реших, че усилията и времето, които бих вложила в изучаване на този тип работа, могат да бъдат приложени в нещо по-лесно и подходящо за сега. Това доведе до избора на AVL tree. Това ми се стори удачно, тъй като приех тази структура като алтернатива на предишната, като основната разлика беше, че този път се работи не с постоянната памет, а с RAM паметта. Все пак независимо от темата на проекта, на такъв ранен етап, програмата не е приспособена за работа с огромни количества данни. Нека не забравяме, че най-много един потребител може да използва системата. И така този избор ми се стори най-логичен, след като мислех, че за използване на Hash Table трябва сама да измисля хеш-функция, което ми се стори като прекалено трудоемка и непосилна задача. След имплементиране и приспособяване на AVL Tree към типа от данни, които ще съхранява – Song, установих, че е възможно да се използва Hash Table без да се налага да я имплементирам. Това директно промени нагласата ми към проекта и замених създадената от мен структура с unordered\_map. Хеш таблицата според мен е най-подходяща за такъв тип съхраняване на данни, тъй като средната сложност на операциие търсене, добавяне и изтриване е константна ( О(1)). За сравнение, сложността на същите операции при другите ми две възможности (самобалансиращи се дървета) е логаритмична, което не е бавно, но не е за предпочитане. Освен това няма реална нужда от азбучно сортирано съхранение на данните, а единствено от възможно най-бърз достъп до тях. Предпочетох избора на unordered\_set пред вектор, тъй като е по-подходящ за произволно добавяне и изтриване на елемент. В метода за генериране на плейлист, създавам копие на оригиналния unordered\_map, от който да премахвам песни, които не отговарят на критериите. След финалният му вид, взимам единствено имената(т.е ключовете) и ги добавям в set, за да се сортират възможно най-бързо. След проверка в различни източници в интернет, реших, че това е най-бързият начин, заради логаритмичната му сложност, която е втората по бързина след константната. Освен това заделянето на допълнителна памет не е проблем, тъй като след изпълнение на функцията, тя се освобождава, тъй като това е само временна структура за съхранение на данните. След това имената на песните се записват в лист, който е атрибут на клас Плейлист. За плейлистите използвах list, тъй като предразполага за линейно обхождане (например при бъдещо добавяне на опция за слушане на песни), както и добавяне и изтриване от края с константна сложност. Изборът на използваните структури от данни е базиран на проучване и запознаване с основните характеристики и предимства на всяка една от тях.

1. ***Трудности, коментари и ползи***

Работата по този проект за пореден път ми доказа колко важна е практическата дейност в програмирането. За периода на изготвяне на системата, научих много нови и полезни неща, като например да имплементирам сама самобалансиращи се дървета, което пък допринесе за разбирането на цялостната концепция на структурата „дърво“, която без тези положени услия, нямаше да усвоя толкова добре. Тъй като дърветата са частен случай на графите, добих основни познания и по тази тема. Това улесни работата ми на лекции и упражнения, тъй като разбирах в дълбочина материала. Събуденият в мен на упражнения интерес относно хеш-таблиците, се засили още повече, когато ми се наложи да видя практическото им приложение и да се задълбоча в употребата им. Добре запомних разликите между unordered\_map, unordered\_set, ordered\_map, ordered\_set и съответните multi-. Видях приложението на свързаните списъци, които да призная до преди проекта не считах за особено полезни. Упражних използването на queue, макар и в последствие да премахнах тази структура от проекта си. Освен това имах възможността да запълня доста от пропуските си по отношение на предишен материал, който сме изучавали. Научих се да използвам catch/try/throw, което преди беше табу за мен. Запознах се добре с работата с потоци – четене и записване, които миналата година по обектно ориентирано програмиране бяха огромна трудност за мен. Открих колко полезен и интересен може да бъде stringstream-а. Отделих време за подготовка за изпълнението на проекта, запознах се с някои тънкости и подробности, които извън контекста на проекта не знам дали бих срещнала. През цялото време интересът ми към проекта се разрастваше и допълнителната информация вече не ми се струваше натоварваща и сложна, а я възприемах като шанс за знания и подобрение. Мисля, че темата на проекта също помогна за поддържането на интереса, тъй като откривах реално приложение на системата, можех да си направя аналогични сравнения с други музикални библиотеки, които съм използвала и крайната цел беше удовлетворяваща. Радвам се, че имах възможността да развия този проект и да придобия всички тези знания.