**Описание решения**.

Решение представляет собой модификацию ответа на задание 15.9 (описание содержится в <https://github.com/dmitryfedotov-56/hash> в папке DOCS), который, в свою очередь, является развитием итогового проекта по блоку “Основные конструкции C++”. Подробные описания функциональности и реализации итогового проекта содержатся в папке DOCS в <https://github.com/dmitryfedotov-56/chat>.

Данное решение предусматривает возможность хранения данных только в RAM. Функциональность решения не изменилась совершенно. В папке DEMO содержится демонстрационное приложение stlb.exe. Так же, как и ранее, при запуске приложения создаются три пользователя (см. таблицу), выполняется обмен сообщеними между пользователями.

|  |  |
| --- | --- |
| **Пользователь** | **Пароль** |
| user1 | pass1 |
| user2 | pass2 |
| user3 | pass3 |

Interface

chat\_interface.cpp

chat\_access.h

Logic

chat\_class.h

chat\_logic.cpp

sha1.cpp

sha1.h

STL

chat\_object.h

DBCS

stl\_object.cpp

Рис. 1. Архитектура решения.

Подробное описание общих принципов построения решения содержится в файле “реализация чата” в папке DOCS на <https://github.com/dmitryfedotov-56/chat>. Архитектура данного решения представлена на рис.1. Стрелки соответствуют директивам include. Заголовочные файлы показаны белым цветом.

Функциональность каждого слоя представлена заголовочным файлом. Реализция каждого слоя независима от реализации других слоев. Интерфейс реализован в файле chat\_interface.cpp.

Функциональность слоя Logic представлена файлом chat\_access.h, содержащим описания абстрактных классов. Файл Файл chat\_class.h содержит описание классов – наследников абстрактных классов, описанных в chat\_access.h. Файл chat\_logic.cpp содержит реализацию классов, описаных в chat\_class.h.

Реализация слоев Interface и Logic не подверглась каким-либо изменениям, она полностью сохранена. Функциональность слоя DBCS представлена файлом chat\_object.h, содержащим описание абстрактных классов. Реализация классов – наследников классов, описанных в chat\_object.h, содержится в stl\_object.cpp, при этом используется функциональность STL и sha1.

Для реализации списка сообщений пользователя мне представляется логичным использовать list, т.к. сообщния отправляются часто, и включение в список должно выполняться эффективно, с другой стороны, при формировании выборки сообщений все равно приходится последовательно просматривать весь список сообщений пользователя.

На первый взгляд для реализации списка пользователей представляется логичным использовать unordered\_map, потому что поиск пользоваетеля по имени должен выполняться быстро (при, возможно, большом числе пользователей), а регистрация выполняется сравнительно редко.

Недостатком такого подхода являлось бы то, что unordered\_map хранит пары { <ключ>, <значение>}, хотя в данном случае <ключ> содержит само <значение>, т.к. имя пользователя хранит объект класса User. Таким образом, ключ будет дублироваться. Конечно, можно было бы поступать по принципу “память есть – ума не надо”, но мне захотелось создать более оптимальное решение. Для этого пришлось создать класс User\_Ptr - обертку для указателя на объект класса User\_Object, для User\_Ptr определить оператор сравнения на равенство == и структуру Hash\_User\_Ptr, позволяющую вычислять hash объекта класса User\_Ptr (он вычисляется, как станадартный hash для класса string).

Значения (а в данном случае это экземпляры класса User\_Ptr) хранятся в структуре данных класса unordered\_set. К сожалению, для поиска по ключу приходится “заворачивать” ключ во вспомогательный экземпляр класса User\_Object, (поэтому был добавлен альтернативный конструктор класса User\_Object), указатель на такой вспомогательный экземпляр User\_Object “заворачивать” в экземпляр класса User\_Ptr.