**Реализация чата.**

Текст содержит описание реализации решения.

Мне представляется важным разделить решение задачи на следующие части (реализация каждой части должна быть независима от другой), функциональность частей следующая.

* Интерфейс пользователя (interface).
* Хранение и извлечение информации (logic).

chat\_access.h

interface

logic

chat\_interface.cpp

Рис.1. Общий дизайн решения.

Общий дизайн решения показан на рис. 1. Я рассматриваю решение этой задачи, как заготовку для дальнейшего развития. Очень вероятно, что в дальнейшем хранение и извлечение информации будет реализовано на основе какой-нибудь СУБД SQL, а интерфейс станет не консольным, а графическим (или Web-интерфейсом). По сути неизменной частью дизайна останется только интерфейс между этими двумя частями. Поэтому решения, касающеся организации этого интерфейса, стоит продумать тщательно, стараясь предвидеть те требования, которые могут возникать в дальнейшем. Помимо этого уже на этом начальном этапе я бы ввел такое понятие, как сессия пользователя.

**Характеристики данных**.

**Пользователь**. Каждый пользователь обладает следующими атрибутами.

* Имя пользователя.
* Пароль пользователя.

**Сообщение**. Сообщение обладает следующими атрибутами.

* Текст сообщения.
* Метка времени создания.
* Отправитель.
* Получатель.

Здесь также неободимо заметить следующее. В решениях подобного рода часто реализуется возможность создания групп пользователей. Группа может быть получателем сообщения, но не может быть отправителем. Поэтому стоит заранее подумать о возможности такой функциональности. Еще одна вещь, которая может понадобиться – это удаление информации (например, удаление пользователем некоторых отправленных им сообщений). Эти обстоятельства стоит также учитывать при дальнейшем проектировании.

**Интерфейс логики решения**.

Описание интерфейса содержится в chat\_access.h. Функция open\_Chat возвращает указатель на объект, описываемый абстрактным классом **Chat\_Access**. Имена всех абстрактных классов содержат слово “Access”.

Класс **User\_Access**. Такой класс обладает двумя методами.

* Регистрация пользователя.
* Авторизация пользователя.

При регистрации проверяется уникальность имени. При авторизации проверяется пароль. Оба метода возвращают указатель на объект, описываемый абстрактным классом **Session\_Access**.

Класс **Session\_Access**. Такой класс обладает следующими методами.

* Отправка адресного сообщения пользователю.
* Широковещательная отправка (всем пользователям).
* Извлечение сообщений.
* Изменение пароля.

Метод извлечения сообщений select\_Message организован так. В качестве аргумента передается условие выборки, описываемое структурой **Condition**.

Условие позволяет задавать следющее.

* Включить полученные сообщения.
* Включить отправленные сообщения.
* Имя пользователя.

Метод select\_Message возвращает указатель на объект, описываемый абстрактным классом **Message\_Collection\_Access** (выборка сообщений). Такой объект является итератором, если метод has\_Next возвращает true, то объект содержит данные очередного сообщения, которые могут быть получены при помощи методов класса. Сообщения упорядочены по времени создания.

Сообщения могут просматриваться, как в прямом порядке (сначала самое ранее), так и в обратном (сначала самое позднее). Для консольного интерфейса удобнее прямой порядок (сначала самое первое сообщение), для Web-интерфейса удобнее обратный порядок (сначала самое последнее сообщение), как во всех почтовых сервисах.

Методы Message\_Collection\_Access позволяют получить следующие данные .

* Текст сообщения.
* Метку времени.
* Информацию о том, было ли сообщение отправлено или получено.
* Имена отправителя и получателя
* Уникальный идентификатор сообщения ID. Уникальный идентификатор может понадобится в дальнейшем для удаления сообщений.

При обнаружении ошибки регистрируется исключение, обработчику передается код ошибки. Поскольку в дальнейшем по-видимому извлечение информации будет выполняться DBCS, то логика logic имеет вспомогательный характер, необходимый лишь на данном этапе развития проекта.

**Дизайн решения.**

Предлагается гибкий дизайн, позволяющий хранить данные не только в RAM, но и в NVM. Дизайн такого решения показан на рис.2 (более подробно показан в приложении к данному документу).

chat\_interface.cpp

Interface

chat\_access.h

chat\_logic.cpp

Logic

chat\_object.h

chat\_object.cpp

Data Base

chat\_storage.h

Storage

file\_storage.cpp

ram\_storage.cpp

Рис. 2. Полный дизайн решения.

Вся информация в БД представлена в виде объектов, данные которых размещаются в некторой среде хранения (RAM или NVM), каждому объекту назначается уникальный идентификатор ID. Доступ к объектам осуществляется при помощи экземпляров некотрых классов. Имеется три следущих уровня.

**Логика решения**. Этот уровень (Logic) абстрагирован не только от способа хранения объектов, но и от способа их представления, имеет значение только то, какими данными обладают объекты, и какие связи между объектами существуют. Этот уровень использует интерфейс, описанный в файле chat\_object.h.

**Реализация БД.** На этом уровне (Data Base) объекты представлены, как структуры, построенные из блоков, находящихся в среде хранения. Каждый блок – последовательность байт, каждому блоку назначается уникальный идентификатор. Представление объектов никак не зависит от способа хранения блоков. Этот уровень использует интерфейс, описанный в файле chat\_storage.h.

**Хранилище данных**. На этом уровне (Storage) реализовано хранение блоков в среде размещения данных. Такой средой может быть как RAM, так и NVM. В частности, это может быть дисковый файл.

Реализация каждой из этих компоненет зависит только от интерфейса, предоставляемого компонентой более низкого уровня. В этом решении абстрагирование осуществляется следующим образом: компонента более высокого уровня реализует свои алгоритмы при помощи объекта более низкого уровня, объекта, описываемого абстрактным классом. Насколько я помню, у “бандитов” GoF такой прием дзю-до назывался “стратегия”. Это обеспечивает разделения решения на несколько уровней.

**Интерфейс базы данных**.

Интерфейс, описанный в chat\_object.h, представляет объекты в виде абстрактных классов. Каждый вид объекта представлен абстрактным классом, эеземпляр такого класса дает доступ к объектам. Каждый такой класс обладает методом создания объекта. Все классы, описывающие объекты, являются наследниками абстрактного класса Object\_Class, обладающим виртуальными методами сохранения объекта save и открывания данных объекта open. Помимо этого каждый такой класс обладает методом создания объекта (специфическим для каждого класса).

Абстрактный класс **List\_Access** описывает объект, представляющий список идентификаторов объектов. Список организован именно как двусвязный список, чтобы при необходимости можно было эффективно организовать операцию исключения элемента. То обстоятельство, что список двусвязный, полезно еще для решения следующей задачи: выборка сообщений может быть упорядочена по возрастанию и убыванию меток времени.

Класс обладает следющими методами.

* Методы получения первого и последнего элемента списка.
* Методы перехода к следующему и к предыдущему элементу списка.
* Метод включения элемента.

**Пользователь**. Пользователь описывается абстрактным классом **User\_Access**. Методы позволяют следующее.

* Получить имя пользователя.
* Получить пароль пользователя.
* Изменить пароль пользователя.
* Получить идентификатор списка ссылок на сообщения.

**Ссылка на сообщение**. Описывается абстрактным классом **Message\_Link\_Access**, методы которого позволяют следующее.

* Получить информацию о том, было ли сообщение отправлено или получено.
* Получить идентификатор сообщения message\_id.

**Сообщение**. Описывается абстрактным классом **Message\_Access**, методы которого позволяют следующее.

* Получить текст сообщения.
* Получить метку времени создания.
* Получить идентификатор отправителя sender\_id.
* Получить идентификатор получателя recipient\_id.

Функция get\_Chat\_Class открывает доступ к объектам чата и возвращает указатель на объект класса **Chat\_Class\_Access**, позволяющий создавать экземпляры всех классов User\_Access, Message\_Access, Message\_Link\_Access, List\_Access. Метод close закрывает доступ к объектам чата.

**Логика решения**.

Логика решения реализована в chat\_logic.cpp, модуль содержит реализацию классов – наследников абстрактных классов, описанных в chat\_access. Схема данных (связи между объектами различных классов) представлена на рис.3. Объект класса Session содержит идентификатор пользователя, который, в свою очередь, позволяет получить идентификатор списка сообщений пользователя.

Абстрактные классы и реализации.

|  |  |
| --- | --- |
| **Абстрактный класс** | **Реализация** |
| Chat\_Access | Chat |
| Session\_Access | Session |
| Message\_Collection\_Access | Мessage\_Collection |

Методы класса Chat реализованы следующим образом.

1. При создании проверяется уникальность имени пользователя, создается соответствующий объект и помещается в список пользователей.
2. При авторизации пользователя проверяется пароль.

sender\_id

message\_id

User

Message Link List

Message Link

…

User List

Message

User

Message Link List

Message Link

message\_id

recipient\_id

Рис. 3. Схема данных.

Адресная отправка сообщения.

1. Создается объект сообщения
2. Ссылка на сообщение включается в список ссылок отправителя с информацией о том, что сообщение отправлено.
3. ссылка на сообщение включается в список ссылок получателя с информацией о том, что сообщение получено.

Широковещательная рассылка сообщений.

1. Создается объект сообщения
2. Ссылка на сообщение включается в список ссылок отправителя с информацией о том, что сообщение отправлено.
3. Ссылка на сообщение включается в в списки всех пользователей (за исключением отправителя) с информацией о том, что сообщение получено.

Выборка сообщений.

1. Создается объект класса Message\_Collection, содержащий идентификатор списка ссылок на сообщения для текущего пользователя сессии.
2. Итератор Message\_Collection последовательно просматривает список, выбирая те сообщения, которые удовлетворяют заданному условию.

Я не стал для класса Chat использовать паттерн Singleton, т.к. экземпляр класса Chat в любом случае необходим, нет смысла создавать его динамически, поэтому функция open\_Chat возвращает указатель на статический экземпляр класса Chat.

Интересно понять, насколько легко можно внедрить те функции, которые пока не реализованы, а именно удаление сообщений и группировка пользователей. Принципиальных проблем я здесь не вижу, но логика решения должна измениться следующим образом.

Необходимо будет создать новые классы объектов.

1. Класс Group, представляющий группу пользователей.
2. Класс Group\_Link, представляющий ссылку на группу пользователей.
3. Класс User\_Link, представляющий ссылку на пользователя.

У класса Group должен быть список объектов User\_Link – список пользователей группы, у класса User должен быть объектов Group\_Link – список групп, в которые включен пользовователь.

Группа рассматривается, как особый вид получателя сообщений, у класса Group должен быть список объектов Message\_Link – список сообщений, отправленных в группу. Отправка сообщения в группу предполагает включение ссылки на сообщение в такой список. Широковещательная рассылка должна быть организована иначе, чем теперь: должна быть создана группа по умолчанию, включающая всех пользователей, при создании нового пользователя он автоматически включается в такую группу.

При такой организации я не вижу никаких проблем с удалением сообщения. Ссылка на любое сообщение содержится только в двух списках – списке отправителя и в списке получателя. Нужно будет только в объекте класса Message хранить два идентификатора объектов Message\_Link, чтобы можно было легко и эффективно исключить ссылку на сообщение из этих двух списков. Я вижу здесь только одну небольшую сложность: при формировании выборки сообщений нужно организовать сортировку сообщений по времени. Впрочем, такую сортировку может довольно легко реализовать сам итератор Message\_Collection, просматривая не только список ссылок на сообщения пользователя, но и списки ссылок на сообщения групп, в которые пользователь включен, и выбирая каждый раз самое раннее (или самое позднее в зависимости от требуемого порядка) сообщение.

**Интерфейс хранилища данных**.

Такой интерфейс описан в storage.h. Операции с блоками выполняются методами объекта, описываемого абстрактным классом Storage\_Access, класс позволяет выполнять следующие операции с блоками.

* Создать блок определенного размера и сохранить в нем информацию.
* Прочитать заданное число байт из блока с заданным идентификатором.
* Сохранить заданное число байт в блоке с заданным идентификатором.
* Удалить блок с заданным идентификатором.
* Создать строковый блок.
* Разместить в памяти строку из строкового блока.

Функция open\_Storage открывает доступ к среде хранения данных и возвращает указатель на объект класса Storage\_Access. Метод Storage\_Access::close закрывает доступ к среде хранения.

**Реализация БД. Представление объектов**.

Реализованы классы которые являются наследниками абстрактных классов, определенных в chat\_object.h (см. таблицу).

Абстрактные классы и реализации.

|  |  |
| --- | --- |
| **Абстрактный класс** | **Реализация** |
| User\_Access | User\_Class |
| Message\_Access | Message\_Class |
| Message\_Link\_Access | Мessage\_Link\_Class |
| List\_Access | List\_Class |

object\_block

Object\_Class

Some\_Access

Some\_Class

Some\_Block

Object\_Block

Методы create, open, save , реализуемые через object\_block

Абстрактные методы create, open, save

Реализация методов craeate, open, save для конкретного класса

Методы create, open, save , наследуемые от Object\_Class

Применение шаблонного класса Block\_Template

Рис. 4. Схема реализации классов объектов.

Все такие классы реализованы в соответствии со схемой, показанной на рис. 4 (белым цветом показаны абстрактные классы, вместо Some подставляется User, Message, Message\_Link, List). Реализован также класс Chat\_Class, являющегося наследником класса Chat\_Class\_Access.

Класc Object\_Class содержит указатель на экземпляр некоторого класса Some\_Block, который является наследником абстрактного класса Object\_Block, имеющего абстрактные методы create, open, save. Some\_Block реализует методы open, save, create. Object\_Class реализует такие методы, вызывая методы Some\_Block. Класс Some\_Block получен для каждого Some\_Class применением шаблона Blocк\_Template. Класс Some\_Class, является наследником Some\_Access, являющегося наследником Object\_Class, поэтому Some класс также обладает методами create, open, save, реализованными в Object\_Class. Фактически используется паттерн Bridge (связывает иерархии Some\_Class -> Object\_Class и Some\_Block -> Object\_Block).

**Размещение блоков**.

Имеются две реализации класса – наследника класса Storage\_Access. Обе реализации могут рассматриваться только как временные решения, используемые для тестирования и полезные в плане proof of concept.

*a) Размещение в RAM.*

Тривиальная реализация класса RAM\_Srorage (см. файл ram\_storage.cpp). Идентификатором блока является указатель. Такая реализация позволяет легко удалять блоки, но данные сохраняются только во время работы приложения. По завершении работы приложения все данные удаляет ОС.

*a) Размещение в файле.*

Реализация класса File\_Storage также тривиальна (см. см. файл ram\_storage.cpp). Блоки последовательно размещаются в файле. Идентификатором блока является смещение блока от начала файла. Функция open\_Storage открывает файл, метод File\_Storage::close закрывает файл.

Такая реализация не обладает важными характеристиками, необходимыми для практического использования.

1. Не позволяет освобождать память, которая не используется (память блоков, которые логически удалены).
2. Не обеспечивает целостность данных при сбоях (отсутствует транзакционная обработка).

Конечно, эти проблемы тоже поддаются решению. Тем не менее, я считаю нецелесообразным дальнейшую работу в этом направлении. Я рассматривал это задание, как упражнение в создании гибкого дизайна. В дальнейшем задача хранения и извлечения информации будет решаться при помощи СУБД SQL. Придет могучий “сиквел”, и многие проблемы просто исчезнут!

Приложение. Схема решения.

Схема решения представлена на рис. 5. Стрелки соответствуют директивам include. Заголовочные файлы показаны белым цветом. Пунктирными линиями показаны альтернативные реализации интерфейса.

Interface

chat\_interface.cpp

chat\_access.h

Logic

chat\_class.h

chat\_logic.cpp

chat\_object.h

DBCS

chat\_object.cpp

storage\_access.h

Storage

ram\_storage.cpp

file\_storage.cpp

Рис. 5. Схема зависимостей модулей.

Решение может быть представлено в виде нескольких слоев: Interface, Logic, DBSC, Storage. Функциональность каждогог 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.

Функциональность слоя DBCS представлена файлом chat\_object.h, содержащим описание абстрактных классов. Реализация классов – наследников классов, описанных в chat\_object.h, содержится в chat\_object.cpp, при этом используется функциональность sha1 и hash\_table.

Функциональность слоя Storage описывается файлом storage\_access.h, описывающим доступ к среде размещения данных. Имеются две альтернативных реализации этой функциональности - ram\_storage.cpp и file\_storage.cpp.