**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

пояснительная записка

**по дисциплине** **«Разработка ПО ИС»**

Тема: Шахматный редактор

|  |  |  |
| --- | --- | --- |
| Студент гр. 4382 |  | Ларионов Д.С. |
|  |  | Миронов В.А. |
|  |  | Перков А.С. |
| Преподаватель |  | Заславский М.М. |

Санкт-Петербург

2017

**Аннотация**

В данной работе представлен программный проект – приложение, которое обучает пользователей шахматному искусству путем анализа и разбора шахматных партий. Пользователь имеет возможность создавать свои или же просматривать чужие партии. Для их хранения и взаимодействия с ними в данной работе используются решения документоориентированной СУБД MongoDB.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 4 |
| 1. | Качественные требования к решению | 5 |
| 2. | Сценарии использования | 6 |
| 3. | Модель данных | 10 |
| 4. | Разработанное приложение | 12 |
| 5. | Выводы | 13 |
|  | Список использованных источников | 14 |
|  | Приложения. | 15 |

**введение**

В настоящее время настольная игра “Шахматы” имеют большую популярность во всем мире. Также игра имеет богатую на сыгранные партии историю, и множество данных партий достойны для изучения как новичками, так и профессионалами. Однако инструмента, который позволил бы комбинировать анализ партий гроссмейстеров с созданием на их основе или же совершенно новых партий, в открытом доступе нет.

Целью работы является разработка приложения, которое должно помочь в обучении мудростям данной игры новых людей, должно позволить пользователям подробно проанализировать прошедшие партии – свои и чужие, и, разумеется, создавать свои.

Авторы предлагают программный проект с графическим интерфейсом, главным составляющим компонентом которого будет игровое поле с шахматными фигурами, на котором можно играть с учетом правил шахмат. Также в данном проекте пользователю будет предоставлен доступ к загрузке и удалению доступных партий, сохранению своих ходов, поиску существующей партии по дебюту, составленному самим пользователем.

**1. Качественные требования к решению**

* 1. **Текущие требования**
* Пользователь имеет возможность ходить фигурой по игровому полю.
* Все шахматные ходы подчиняются шахматным правилам
* Пользователь видит партии, которые ему доступны, он может их загружать и удалять.
* Пользователь может сохранять свою последовательность ходов в виде партии.
* Пользователь может найти существующую партию по сделанным пользователем ходам.
  1. **Перспективы на будущее**
* На каждом этапе приложение показывает доступные ходы.
* Хранение партий в облачном хранилище и взаимодействие с ним.
* Пользователь может добавлять комментарии к каждому ходу.
* Мобильная разработка.
* Голосовое управление.

**2. Сценарии использования**

**2.1. Макет UI**

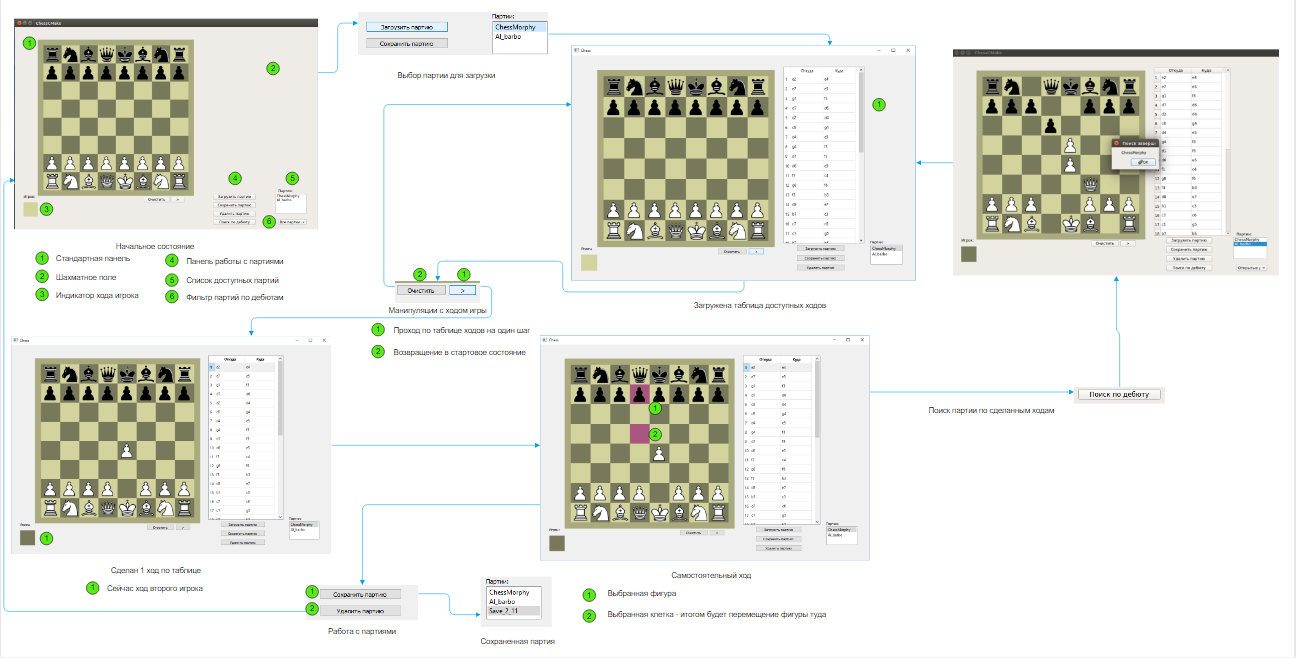
****

Рис. 1

Модель в полном размере: https://github.com/moevm/nosql-2017-chess\_is/blob/master/Interface.png

**2.2. Сценарии использования для задачи**

* **UC**: Загрузка партии.
* **Действующее лицо**: Пользователь приложения.
* **Предусловия**: Пользователь имеет перед собой главное окно со списком доступных партий
* **Основной сценарий**:

1. Пользователь выбирает партию из списка кликнув на него.
2. Пользователь нажимает на кнопку “Загрузить партию”
3. Приложение загружает из своей базы данных выбранную партию
4. Приложение выводит последовательность ходов в списке справа от доски

* **UC**: Ход игрока по выбранной партии
* **Действующее лицо**: Пользователь приложения.
* **Предусловия**: Пользователь имеет перед собой главное окно с шахматной доской с фигурами и со списком загруженных ходов
* **Основной сценарий**:

1. Пользователь нажимает на кнопку “->”
2. Приложение проверяет выбранный ход и использует его
3. Приложение отображает новое положение фигур на доске
4. Приложение помечает совершенный ход в списке ходов.

* **UC**: Самостоятельный ход игрока
* **Действующее лицо**: Пользователь приложения.
* **Предусловия**: Пользователь имеет перед собой главное окно с шахматной доской с фигурами
* **Основной сценарий**:

1. Пользователь выбирает фигуру и указывает куда ей ходить
2. Приложение проверяет выбранный ход на соответствие правилам игры
3. Приложение отображает новое положение фигур на доске

* **UC**: Сохранение партии.
* **Действующее лицо**: Пользователь приложения.
* **Предусловия**: Пользователь имеет перед собой главное окно со списком доступных партий
* **Основной сценарий**:

1. Пользователь нажимает на кнопку “Сохранить партию”.
2. Приложение сохраняет все прошедшие ходы в базу данных с автоматически созданным названием.
3. Приложение выводит сохраненную партию в соответствующем списке.

* **UC**: Проход партии заново.
* **Действующее лицо**: Пользователь приложения.
* **Предусловия**: Пользователь имеет перед собой главное окно с шахматной доской с фигурами и со списком загруженных ходов
* **Основной сценарий**:

1. Пользователь нажимает на кнопку “Очистить”.
2. Приложение расставляет обратно все фигуры в изначальную позицию.

* **UC**: Поиск партии по дебюту
* **Действующее лицо**: Пользователь приложения.
* **Предусловия**: Пользователь имеет перед собой главное окно с шахматной доской с фигурами и со списком загруженных ходов, пользователь ходил несколько раз.
* **Основной сценарий**:

1. Пользователь нажимает на кнопку “Поиск партии по ходам”
2. Приложение ищет партию по соответствию с последовательностью совершенных ходов.
3. Приложение отображает найденную партию в списке ходов, с выделением текущего хода.
4. Приложение выводит сообщение о загрузке. Расширения: Если приложение не находит партию
5. Приложение выводит сообщение об ошибке.

* **UC**: Удалить партию
* **Действующее лицо**: Пользователь приложения.
* **Предусловия**: Пользователь имеет перед собой главное окно со списком доступных партий
* **Основной сценарий**:

1. Пользователь нажимает на кнопку “Удалить партию”.
2. Приложение удаляет партию из базы данных.
3. Приложение выводит обновленный список.

**3. Модель данных**

**3.1. Нереляционная модель данных**

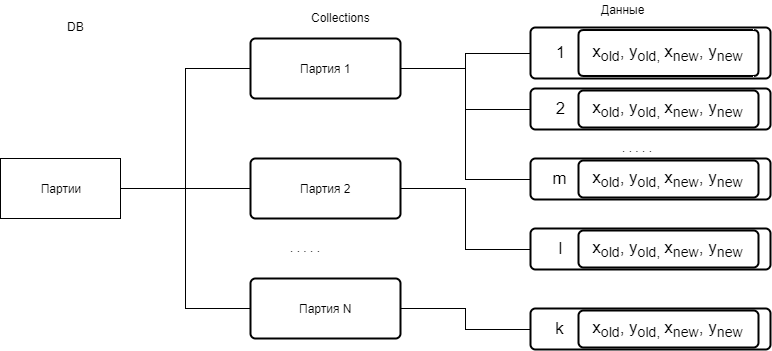
****

Рис. 2 Графическое представление модели

В данной БД коллекции представляют собой шахматные партии, состоящие из ходов. Ходы имеют 4 поля – координаты хода, откуда и куда. Координаты имеют целочисленный тип [0-7]. Пример:

{

“xold”:”3”,

“yold”:”4”,

“xnew”: “5”,

“ynew”:”6”,

}

### Оценка удельного объема информации, хранимой в модели:

{

\_id:12 байт

“xold”:1 байт,

“yold”:1 байт,

“xnew”: 1 байт,

“ynew”:1 байт,

}

Поле \_id создается автоматически и имеет формат ObjectId. Итого 16 байтов. Если имеется N партий и в среднем в каждой партии по M записей, то объем памяти равен 16MN байт.

Запросы к модели, с помощью которых реализуются сценарии использования:

* Сохранение партии: db.newGame.insert([{“xold”:”3”, “yold”:”4”, “xnew”: “5”, “ynew”:”6”,} ,...,{ “xold”:”3”, “yold”:”4”, “xnew”: “5”, “ynew”:”6”}])

K операций – по размеру партий.

* Удаление партии: db.game.drop()
* Список партии db.getCollectionNames()
* Поиск по дебюту: цикл из db.game.find({“xold”:”3”, “yold”:”4”, “xnew”: “5”,“ynew”:”6”}) N\*K операций (K – размер дебюта)

Альтернативный вариант: каждая партия является единым документом. Тогда на одну партию нужен всего один \_id, следовательно, объем памяти: N(12+M\*4). Однако в данном варианте сильно усложняется реализация запросов, в особенности поиска по дебюту.

**3.2. Реляционная модель данных**

Графическое представление данной модели совпадает с рис. 2.

В данной БД коллекции заменяются таблицами. Каждая запись имеет 4 поля – индекс, координаты хода, откуда и куда. Также в запись включается автоматически сгенерированный индекс размером в 4 байта. Координаты имеют целочисленный тип [0-7]. Пример:

{

“id”:”47”,

“xold”:”3”,

“yold”:”4”,

“xnew”: “5”,

“ynew”:”6”,

}

Объем памяти: 8NM байт.

Запросы:

* Сохранение партии: "create table "+name+"(id integer primary key, xold integer(1), yold integer(1), xnew integer(1),ynew integer(1))" "insert into "+table+"(xold,yold, xnew, ynew) values(?, ?, ?, ?)" 1+K операций
* Удаление партии: "drop table "+name
* Список партии Select \*
* Поиск по дебюту "SELECT TOP K xold,yold, xnew, ynew FROM "+table и сравнение. Либо составной запрос по таблицам. N\*K операций.

**3.3. Сравнение моделей**

Сравнение приведенных моделей показывает, что реляционная модель занимает меньше памяти, однако взаимодействие с MongoDB требует чуть меньше операций, к тому же сами операции проще в исполнении. При большом количестве партии и ходов разница в скорости поиска и вставки будет заметна гораздо слабее, чем разница в памяти. Поэтому предпочтительна SQL модель.

1. **Разработанное приложение**

Программный проект представляет из себя GUI приложение, разработанное на языке C++/Qt в объектно-ориентированной парадигме c использованием шаблона MVC и представленное под Linux-платформу. Приложение полностью локальное, однако при запуске оно автоматически разворачивает свой сервер MongoDB. Связь с БД происходит при помощи специального драйвера для С++.

Все снимки приложения можно посмотреть в разделе Макет UI и в Приложении А.

В Приложении Б представлена документация по сборке и развертыванию приложения.

Ссылка на github: <https://github.com/moevm/nosql-2017-chess_is>

1. **Вывод**

**Результаты:**

В ходе работы было разработано приложение, которое способно удовлетворить все поставленные задачи: пользователь может взаимодействовать с игровым полем, производя ходы шахматными фигурами, которые не противоречат правилам игры. Данные ходы пользователь может сохранять, загружать, удалять и находить в самом приложении.

**Недостатки и пути для улучшения полученного решения:**

На данный момент модель памяти, которую использует приложение, не является эффективным, как показывает сравнение. Необходимо перейти к реляционной модели либо видоизменить нереляционную, искать иные решения.

**Будущее развитие решения:**

В связи с огромным количеством существующих партий, необходимо изменить способ их хранения, как вариант – облачные хранилища. Также возможна мобильная разработка данного приложения.

**список использованных источников**

1. Документация MongoDB <https://docs.mongodb.com/manual/>

2. Banker K. MongoDB In Action: Manning 2012

3. MVC: что это такое https://tproger.ru/articles/mvc/

4. Документация Qt http://doc.qt.io/qt-5

**приложение А**

**Снимки экрана приложения**





**приложение б**

**Документация по сборке и развертыванию приложения**

Исходные коды компилируются при помощи cmake – CMakeList.txt представлен в гитхабе.

Также само приложение представлено в виде deb-пакета. Установка производится при помощи стандартных средств Ubuntu. Если на устройстве уже установлено MongoDB, приложение при первом запуске загрузит все известные ему партии на локальный сервер.

MongoDB устанавливается отдельно по инструкции, приведенной в документации.