|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатики, искусственного интеллекта и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОЙ РАБОТЕ***

***НА ТЕМУ:***

***Разработка базы данных для обеспечения***

***тренировок в фитнес-клубе***

Студент ИУ9-62Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Терентьева А. С.**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Домрачева А. Б.**

(Подпись, дата) (И.О.Фамилия)

*Москва, 2023 г.*

**ОГЛАВЛЕНИЕ**

1. Теоретические сведения 4

1.1. Базы данных 4

1.1.1. Релияционные базы данных 4

1.1.2. Язык SQL и СУБД PostgreSQL 6

1.2. Инструменты веб-разработки 10

1.3. Векторная графика 11

1.4. Кривая Безье 11

2. Реализация 12

2.1. Реализация арихтектура 12

2.2. Реализация регистрации и авторизации 12

2.3. Реализация размещения дуг связей между блоками задач 12

2.4. Функциональные возможности приложения 13

3. Оценка визуализации и обзор аналогов 14

ВВЕДЕНИЕ

Цель данной курсовой работы - разработать базу данных для фитнес-клуба, которая позволит вести учет тренировок пользователей и хранить статистику их показателей тела. База данных является важным инструментом для успешного функционирования фитнес-клуба по следующим причинам:

1. Отслеживание прогресса: база данных может хранить данные о весе, объеме талии, бедер и т.д., а также данные о тренировках, такие как количество повторений, нагрузка и т.д. Это позволяет посетителям отслеживать свой прогресс и улучшать свои результаты.
2. Удобство: база данных может быть доступна для посетителей через веб-интерфейс, что позволяет им удобно отслеживать свой прогресс и планировать свои тренировки.

В работе будут рассмотрены следующие задачи:

* Анализ требований к базе данных для фитнес-клуба;
* Проектирование структуры базы данных;
* Создание таблиц и связей между ними;
* Наполнение базы данных тестовыми данными;
* Создание запросов для извлечения информации из базы данных;
* Разработка интерфейса для работы с базой данных.

Для решения поставленных задач будет использоваться язык SQL и СУБД PostgreSQL. Также будет реализован сервер на языке JavaScript с помощью Node.js и фреймворка Express.js. Результатом работы станет функционирующая база данных для фитнес-клуба, которая будет обеспечивать учет тренировок пользователей и хранение статистики их показателей тела.

# Теоретические сведения

## Базы данных

База данных – это организованная структура данных, которая хранит информацию и предоставляет возможность эффективного доступа к ней. Базы данных используются в различных приложениях для хранения и управления большим объемом информации, такой как данные пользователей, транзакции, каталоги товаров, отчеты, статистика и т.д.

База данных обычно управляется [системой управления базами данных (СУБД)](https://www.oracle.com/cis/database/what-is-database/#WhatIsDBMS). Она используется для организации и обработки данных. Она также обеспечивает механизмы для контроля целостности данных, обеспечения безопасности и резервного копирования данных.

Данные в наиболее распространенных типах современных баз данных обычно хранятся в виде строк и столбцов, формирующих таблицу. Этими данными можно легко управлять, изменять, обновлять, контролировать и упорядочивать. В большинстве баз данных для записи и запросов данных используется язык структурированных запросов (SQL).

По структуре и способу связей основные базы данных делятся на типы: иерархические, сетевые, реляционные, нереляционные (NoSQL), документно-ориентированные, колоночные и графовые.

* + 1. Релияционные базы данных

Реляционная база данных – это тип базы данных, который использует реляционную модель данных, предложенную Эдгаром Коддом в 1970 году. Ее примерами являются Oracle, MySQL, PostgreSQL, Microsoft SQL Server и другие.

Реляционные базы данных представляют собой базы данных, которые используются для хранения и предоставления доступа к взаимосвязанным элементам информации. В реляционной модели данные представлены в виде таблиц, состоящих из строк и столбцов. Каждая таблица представляет отдельную сущность (например, пользователи, заказы, товары и т.д.), а каждая строка таблицы представляет конкретный экземпляр этой сущности, называемый записью. Каждый столбец в таблице представляет отдельное свойство данных (такое как имя, адрес или дата рождения), которое может быть использовано для фильтрации и сортировки данных. Отношения между таблицами устанавливаются с помощью ключевых полей (первичных и внешних ключей), которые связывают записи в разных таблицах.

Реляционная модель базы данных также определяет отношения между таблицами, которые могут быть использованы для связывания данных из разных таблиц. Эти отношения могут быть один-к-одному, один-ко-многим или многие-ко-многим. Они обеспечивают возможность связывания данных из разных таблиц и извлечения информации из нескольких таблиц с помощью запросов на языке SQL.

Типы связей между таблицами:

* «один-к-одному» [1:1]. Каждая запись в одной таблице связана с одной и только одной записью в другой таблице. Для создания связи один-к-одному необходимо добавить в одну из таблиц уникальный внешний ключ, который ссылается на первичный ключ другой таблицы;
* «один-ко-многим» [1:N]. Каждая запись в одной таблице может быть связана с несколькими записями в другой таблице. В таких связях сущность на стороне 1 называют родительской, а на стороне N – дочерней. Для создания связи один-ко-многим необходимо добавить в дочернюю таблицу внешний ключ, который ссылается на первичный ключ родительской таблицы;
* «многие-ко-многим» [N:M]. Этот тип связи означает, что каждая запись в одной таблице может быть связана с несколькими записями в другой таблице, и наоборот. Для создания связи многие-ко-многим необходимо создать третью таблицу, которая будет содержать два внешних ключа, один из которых ссылается на первичный ключ одной таблицы, а другой - на первичный ключ другой таблицы;
  + 1. Язык SQL и СУБД PostgreSQL

SQL (Structured Query Language) – это язык программирования, используемый для управления данными в реляционных базах данных. SQL позволяет создавать, изменять и удалять таблицы и записи, а также выполнять запросы на выборку данных из таблиц.

PostgreSQL – это это объектно-реляционная СУБД (система управления базами данных), которая использует язык SQL для управления данными. PostgreSQL предоставляет множество возможностей для хранения и обработки данных, включая поддержку многопоточности, транзакций, хранимых процедур и т.д. PostgreSQL имеет множество преимуществ, таких как высокая производительность, надежность и стабильность, безопасность данных и поддержка масштабирования. Он также имеет открытый исходный код, что позволяет пользователям изменять код и разрабатывать свои собственные решения на основе этой СУБД.

Основные типы данных в PostgreSQL:

* Целочисленные значения:

integer;

serial – специальный тип данных, который автоматически генерирует уникальные значения при вставке новых записей в таблицу. Основан на типе данных integer со встроенной функцией default и sequence и ограничением not null. Значение данного типа образуется путем автоинкремента значения предыдущей строки. Как правило, используется для определения идентификатора строки;

* Символьные строки:

text;

varchar(n) или character varying(n) – принимает параметр n - максимальную длину строки в символах;

* Дата и время:

date – дата в формате yyyy-mm-dd;

timestamp - дата и время в формате yyyy-mm-dd hh:mi:ss;

timestamp with time zone - это тип данных для хранения даты и времени в формате yyyy-mm-dd hh:mi:ss tz. Он содержит дату и время с точностью до микросекунд и учитывает часовой пояс;

* Булевый тип: boolean;
* Бинарные данные: bytea;
* Массивы: array;
* Данные в формате JSON: json;

Основные действия над таблицами в PostgreSQL:

1. Создание новой таблицы. Пример синтаксиса:

CREATE TABLE table\_name (column1 datatype1, ...);

1. Изменение структуры таблицы, такое как добавление, изменение или удаление столбцов. Пример синтаксиса:

ALTER TABLE table\_name

ADD COLUMN column\_name datatype;

1. Удаление таблицы. Пример синтаксиса:

DROP TABLE table\_name;

1. Вставка новых строк в таблицу. Пример синтаксиса:

INSERT INTO table\_name (column1, column2, ...)

VALUES (value1, column2, ...);

1. Обновление существующих строк в таблице. Пример синтаксиса:

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

1. Удаление строк из таблицы. Пример синтаксиса:

DELETE FROM table\_name WHERE condition;

1. Извлечение данных из таблицы. Пример синтаксиса:

SELECT column1, column2, ... table\_name WHERE condition;

SELECT \* FROM table\_name; – получение всех столбцов таблицы

Данные операции могут выполняться на уровне отдельных записей в таблице или на уровне всей таблицы.

Ограничения (constraints) в базах данных используются для определения правил, которым должны соответствовать данные в таблице. Ограничения могут использоваться для обеспечения целостности данных, защиты от ошибок ввода и других важных задач.

Основные типы ограничений в PostgreSQL:

1. PRIMARY KEY: определение первичного ключа таблицы – уникального идентификатора, который однозначно идентифицирует каждую запись в таблице.
2. FOREIGN KEY: используется для установления связи между двумя таблицами. Ограничение FOREIGN KEY ссылается на столбец первичного ключа в другой таблице и гарантирует, что значение в этом столбце существует в таблице-родителе.
3. NOT NULL: указание, что значение в столбце не может быть NULL.
4. CHECK: проверка, что значение в столбце соответствует определенному условию.
5. UNIQUE: указание, что значение в столбце должно быть уникальным. Это означает, что две записи в таблице не могут иметь одинаковые значения в этом столбце.

Пример синтаксиса каждого из ограничений:

CREATE TABLE table\_name (

column1 datatype1 PRIMARY KEY,

column2 datatype2 NOT NULL,

column3 datatype3 CHECK (column3 > 0),

column 4 datatype4 UNIQUE,

....

FOREIGN KEY (column3) REFERENCES parent\_table (parent\_column)

);

Триггеры – это специальные хранимые процедуры в базах данных, которые автоматически запускаются при определенных событиях, таких как, например, вставка, обновление или удаление записей в таблицах. Триггеры могут выполнять определенные действия, например, проверять целостность данных, генерировать отчеты действий, обновлять связанные таблицы и т.д.

Существуют несколько типов триггеров:

* BEFORE – запускается до выполнения операции
* AFTER – запускается после выполнения операции
* INSTEAD OF – запускается вместо выполнения операции

Пример синтаксиса создания триггера:

CREATE TRIGGER trigger\_name

AFTER INSERT OR UPDATE OR DELETE ON table\_name

FOR EACH ROW

EXECUTE PROCEDURE function\_name();

Функции – это хранимые процедуры в PostgreSQL, которые могут принимать аргументы и возвращать результаты. Создание функций в PostgreSQL происходит с помощью языка SQL или PL/pgSQL. PL/pgSQL - это язык программирования, который был разработан специально для работы с PostgreSQL.

Пример создания функции на языке PL/pgSQL:

CREATE FUNCTION function\_name(arg1 type1, ..)

RETURNS ret\_type\_1 AS $$

BEGIN

RETURN (SELECT ret\_column1 FROM table\_name WHERE column1 = arg1);

END;

$$ LANGUAGE plpgsql;

В PostgreSQL также существует множество встроенных функций, которые могут быть использованы для работы с данными, такие как функции для работы с датами и временем, строками, математические функции и т.д.

Индексы в базах данных – это структуры данных, которые ускоряют поиск и сортировку записей в таблицах. Они создаются на одном или нескольких столбцах таблицы и позволяют быстро находить записи, удовлетворяющие определенному условию. Индексы на нескольких столбцах называются составными индексами. Индексы в базах данных могут быть уникальными или неуникальными. Уникальный индекс ограничивает значения в столбце таблицы, чтобы они не повторялись, тогда как неуникальный индекс позволяет повторяющимся значениям. Пример создания уникального индекса, связывающего два:

CREATE UNIQUE INDEX index\_name ON table\_name (column\_1, column\_2);

Оператор RAISE используется для генерации ошибок или исключений во время выполнения SQL-запросов. Его синтаксис:

RAISE exception\_type 'message';

где exception\_type - тип исключения (например, EXCEPTION, WARNING, NOTICE), а message - текст сообщения об ошибке.

## Инструменты веб-разработки

HTML [3] – Hyper Text Markup Language – Язык разметки гипертекста. это код, который используется для структурирования и отображения веб-страницы и её контента. технологии разработки сайтов страница, которую понимает веб-браузер.

Кроме HTML-кода в браузер загружаются и обрабатываются CSS - каскадные таблицы стилей, и программы (обычно называемые скриптами) на таких языках, как Java, JavaScript, TypeScript.

CSS – это инструмент, позволяющий придать единый вид различным типовым элементам сайта, в частности - легко настраивать и изменять облик всех элементов страницы.

Node.js [4] – это среда выполнения JavaScript, построенная на движке V8. Она позволяет запускать JavaScript на стороне сервера, что позволяет разрабатывать серверные приложения и API. Node.js основан на событийно-ориентированной и неблокирующей модели ввода-вывода, что делает его эффективным и масштабируемым для обработки большого количества запросов.

Express.js [5] – это гибкий фреймворк для разработки веб-приложений на Node.js. Он предоставляет интерфейс для создания маршрутов и обработки HTTP-запросов и отправки ответов на них клиенту.

EJS (Embedded JavaScript) [6] – это шаблонизатор для JavaScript и Node.js. Он позволяет создавать динамические HTML-страницы, вставлять переменные, выполнять условные операторы и циклы, а также подключать другие шаблоны. EJS позволяет разделить представление и логику приложения, что делает код более читаемым и поддерживаемым. В Express.js сгенерированные HTML-страниц отправляются клиенту с помощью функции response.render.

Node.js, Express.js и EJS предоставляют разработчикам инструменты для создания серверных приложений и веб-сайтов с динамическим содержимым. Они позволяют эффективно работать с запросами, маршрутизацией и шаблонизацией.

# Проектирование базы данных

## Анализ предметной области

В качестве предметной области было выбрано приложения для поддержания тренировок и ведения статистики их результатов. Необходимо разработать базу данных, которая будет соответствовать следующим требованиям:

1. Обеспечение хранения пользователей. База данных должна предоставлять следующие возможности пользователя для управления его аккаунтом:

* Регистрация (создание аккаунта)
* Редактирование персональной информации
* Удаление аккаунта

1. Обеспечение хранения статистики. База данных должна предоставлять следующие возможности пользователям для ведения статистики показателей тела и результатов тренировок:

* Создание нового параметра, ведения статистики показателей которого хочет начать пользователь
* Получение параметров, созданных пользователем
* Получение, добавление, редактирование и удаление установленных показателей. Каждому показателю соответствует параметр, а параметр, в свою очередь, может содержать несколько показателей
* Учет даты и времени установления показателей

1. Обеспечение хранения комплексов упражнений. База данных должна предоставлять следующие возможности пользователям для просмотра комплекса упражнений:

* Создание новой тренировки
* Получения всех тренировок, хранящихся в базе данных
* Получение создателя тренировки
* Получение и изменение списка добавленных тренировок в профиль пользователя

## Модель «сущность-связь»

На основе описанной предметной области была создана модель «сущность-связь», включающая четыре сущности:

1. USER – сущность, являющаяся абстракцией пользователя.

Идентификатор:

* user\_id;

Атрибуты:

* name – отображаемое имя пользователя;
* user\_login – уникальное имя пользователя;
* password – хэш пароля пользователя;
* email – электронная почта пользователя;
* gender – пол пользователя;
* birthday – день рождения пользователя;

1. PARAMETER – сущность, являющаяся абстракцией параметра.

Идентификаторы:

* user\_id – идентификатор пользователя, создавшего параметр;
* parameter\_id;

Атрибуты:

* name – название параметра;
* updated\_at – время последнего добавления показателей тела;

1. BODYDATA – сущность, являющаяся абстракцией показателя теля.

Идентификаторы:

* parameter\_id – идентификатор соответствующего параметра;
* user\_id – идентификатор пользователя, создавшего параметр;
* id;

Атрибуты:

* value – значение показателя тела;
* date – время установления значения показателя;

1. WORKOUT – сущность, являющаяся абстракцией тренировки.

Идентификатор:

* id;

Атрибуты:

* name – название тренировки;
* info – описание тренировки;

Диаграмма данной модели приведена на рис. 1.

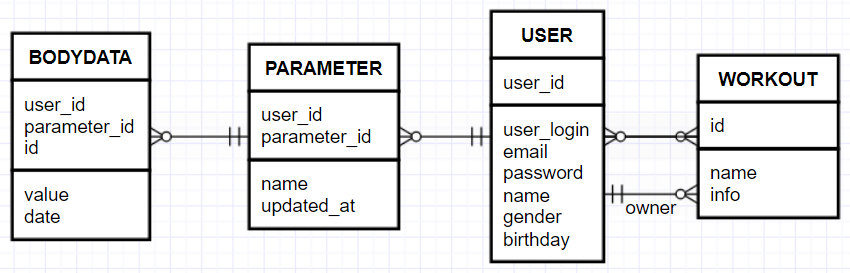


рис. 1 Модель «сущность-связь»

У каждого параметра обязательно должен быть автор, который его создал, а сам пользователь может создавать несколько параметров (или ни одного), между сущностями USER и PARAMETER установлена связь «один-ко-многим» (с минимальными кардинальными числами 1 и 0 соответственно).

Показателей тела одного параметра, может быть неограниченное количество (в том числе и ни одного). Каждый показатель соответствует одному определенному параметру. Между сущностями PARAMETER и BODYDATA установлена связь «один-ко-многим» (с минимальными кардинальными числами 1 и 0 соответственно).

Так как у комплекса упражнений должен быть единственный создатель и при этом пользователь может быть создателем одновременно нескольких комплексов, связь USER – WORKOUT (owner) имеет тип «один-ко-многим» (с минимальными кардинальными числами 1 и 0 соответственно).

Рассмотрим связь USER – WORKOUT. Пользователь может добавить несколько тренировок в профиль, с другой стороны, несколько пользователей могут добавить одну тренировку в свои профили. Поэтому связь USER – WORKOUT имеет тип «многие-ко-многим» (с минимальными кардинальными числами 0 и 0 соответственно).

## Преобразование модели «сущность-связь» в реляционную модель

Из модели «сущность-связь» согласно процедуре преобразования была получена реляционная модель, включающая 5 таблиц (атрибуты двух моделей совпадают):

1. USER – сущность, являющаяся абстракцией пользователя.

Идентификатор:

* user\_id – идентификатор пользователя;

Атрибуты:

* user\_login – альтернативный идентификатор пользователя;
* email – уникальный адрес электронный почты;

1. PARAMETER – сущность, являющаяся абстракцией параметра.

Идентификаторы:

* parameter\_id;

Атрибуты:

* user\_id – идентификатор пользователя, создавшего параметр;

1. BODYDATA – сущность, являющаяся абстракцией показателя теля.

Идентификаторы:

* id;

Атрибуты:

* user\_id – идентификатор пользователя, создавшего параметр;
* parameter\_id – идентификатор соответствующего параметра;

1. WORKOUT – сущность, являющаяся абстракцией тренировки.

Идентификатор:

* id;

Атрибуты:

* owner\_id – идентификатор пользователя, создавшего тренировку;

Диаграмма данной модели приведена на рис. 2.

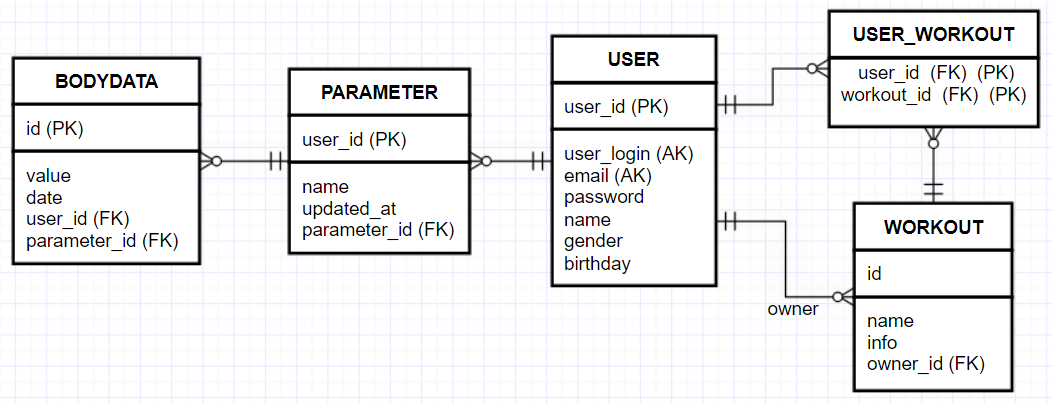


рис. 2 Реляционная модель

# Реализация

В данной части содержится описание реализации приложения.

Клиент и веб-интерфейс проекта разработан с использованием языка разметки HTML, языка каскадных таблиц стилей CSS и языка программирования JavaScript. Сервер проекта разработан на языке JavaScript с помощью Node.js и фреймворка Express.js. При разработке базы данных использовался язык SQL и СУБД PostgreSQL. Сервер базы данных размещается в качестве контейнера в Docker. Репозиторий проекта находится на странице [1].

## Реализация арихтектура

Шкала должна поддерживать прокрутку, масштабирование и смену уровня

Docker — это инструмент для создания, развертывания и запуска приложений в контейнерах. Для данной работы было принято решение использовать его для развертывания сервера базы данных PostgreSQL.

## Реализация регистрации и авторизации

После настройки базы данных и подключения к ней созда на рис. 1

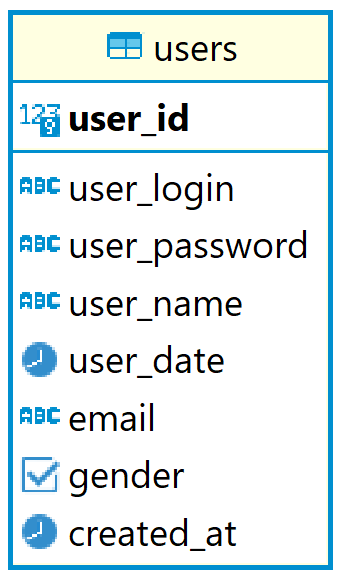


рис. 3

## Реализация размещения дуг связей между блоками задач

Для отрисовки

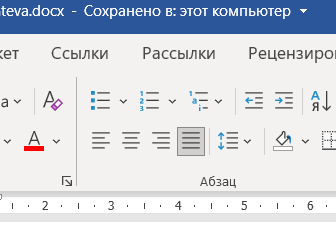


рис. 4 пример3

ребер используется кривой на рис. 3

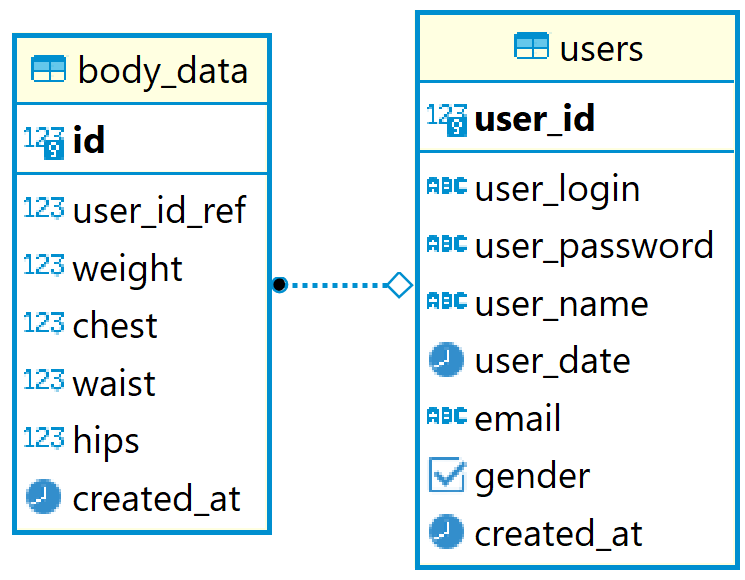


рис. 5 пример2

## Функциональные возможности приложения

Интерфейс страницы представлен на ого графика

# Оценка визуализации и обзор аналогов

Как можно заметить по

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы были рассмотрены различные виды представления графика работ бизнес-проекта, такие как диаграмма Ганта и сетевой график, и технологии их визуализации средствами компьютерной графики. На основе изученной информации была разработана программа, позволяющая демонстрировать и редактировать план проекта.

В ходе выполнения курсовой работы были приобретены практические навыки разработки реляционных БД в СУБД PostgreSQL и их применении для создания приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Репозиторий проекта – <https://github.com/mathhyyn/bd_coursework>
2. Документация PostgreSQL – <https://www.postgresql.org/docs/>
3. Документация веб-технологий для разработчиков – <https://developer.mozilla.org/ru/docs/Web>
4. Документация Node.js – <https://nodejs.org/ru/docs>
5. Документация Express.js – <https://expressjs.com/>
6. Документация шаблонизатора ejs – <https://ejs.co/#docs>
7. Документация библиотеки для работы с PostgreSQL в Node.js – <https://node-postgres.com/>
8. Документация библиотеки express-session – <https://www.npmjs.com/package/express-session>
9. Документация генератора графиков и диаграмм – <https://www.chartjs.org/docs/latest/>

ПРИЛОЖЕНИЕ

export const tasks = JSON.parse

Листинг A – входные данные: массивы задач и зависимостей

    return elem;

};

Листинг B – функция отрисовки кривых

};

Листинг C – функция отрисовки ломанных простым алгоритмом

function

};

Листинг D – функция отрисовки ломанных сложным алгоритмом