



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт комплексной безопасности и специального приборостроения

Кафедра «Цифровые технологии обработки данных»

ОТЧЕТ

по практической работе

«3. Хранилище “ключ-значение” Redis »

по дисциплине «Нереляционные системы управления базами данных»

Выполнил

Смирнов И.А.

фамилия, имя, отчество

шифр

21Б0700

группа

БСБО-11-21

Проверил

К.Т.Н., доцент

ученая степень, должность

Ильин Д.Ю.

фамилия, имя, отчество

Москва 2023г.

Цель практической работы

Цель настоящей практической работы – научиться использовать хранилище «ключ-значение» Redis.

Задачи практической работы

Для достижения поставленной цели необходимо решить следующие задачи:

1. Спроектировать программное обеспечение и отразить в результирующих схемах применение Redis с целью, определенной вариантом задания.
2. Установить Redis и осуществить к нему ручной доступ через любое доступное программное обеспечение.
3. Разработать программное обеспечение, использующее Redis с целью, определенной вариантом задания.
4. Протестировать программное обеспечение и продемонстрировать корректность его работы.
5. Подготовить ответы на контрольные вопросы.
6. Составить отчет о проведенной работе.

Вариант задания

Цель применения Redis: Кеширование веб-страниц

Ход работы

Начнем с проектирования по. На рис 1 приставлена схема ПО.

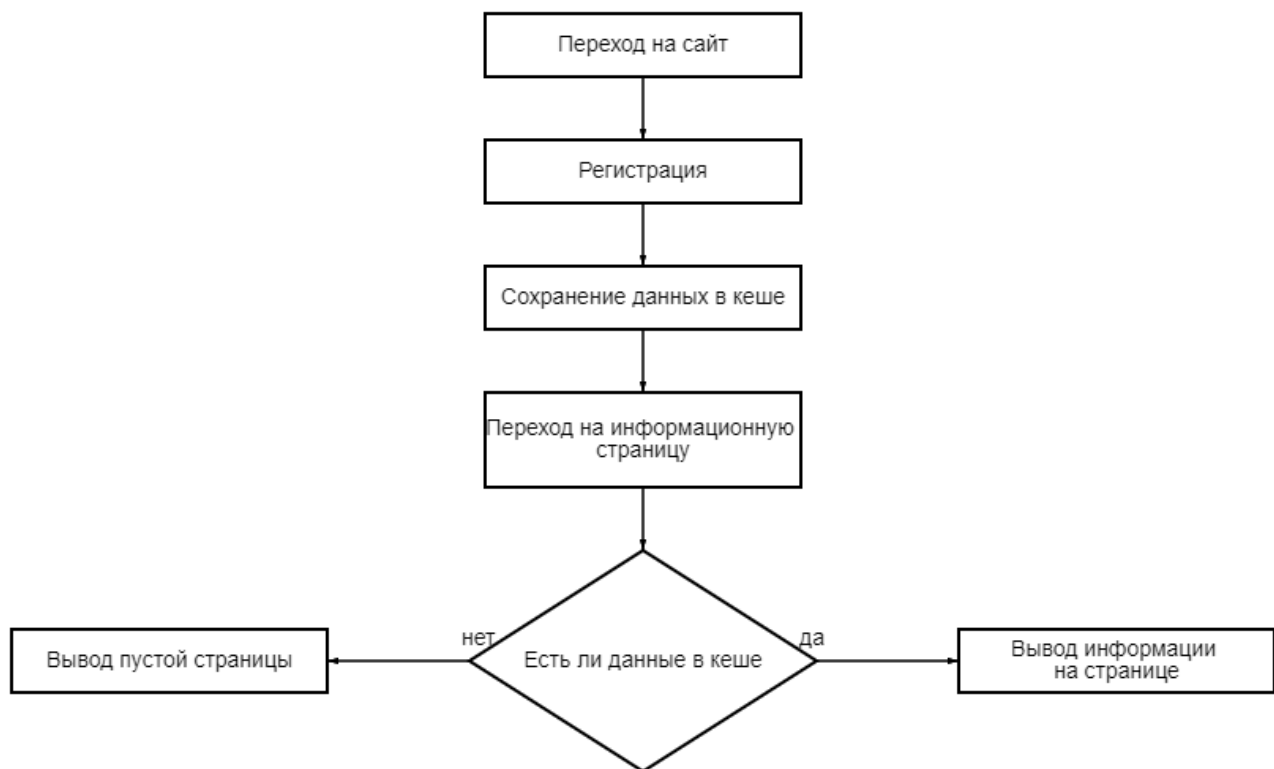


Рис 1. Схема ПО

Установил Docker, запустил Redis, установил AnotherRedisDesktopManager.

На листинге 1 представлен код программы. Использовал библиотеки Redis, Express.

```
import express from 'express';
import path from 'path';
import redis from 'redis';

const __dirname = path.resolve();
const PORT = 3000;
const LINK = 'http://localhost:3000';

const app = express();
let client = '';
let userData = {
  name: '',
  surname: '',
  age: '',
  email: '',
  city: '',
  password: '',
};

(async () => {
  client = redis.createClient();

  client.on("error", (error) => console.log('Что-то пошло не так', error));

  await client.connect();
})();

app.use(express.urlencoded({extended: true}));
app.set('view engine', 'ejs');
app.set('views', path.resolve(__dirname, 'Laba_3/templates'));

async function saveData(userData) {
  await client.set('user', userData, {EX: 30, NX: true});
}

async function getData() {
  try {
    const userData = await client.get('user');
    return JSON.parse(userData);
  } catch (error) {
    console.error(`Error fetching data: ${error}`);
    return null;
  }
}

app.get('/', async (req, res) => {
  try {
    const userData = await getData();

    if (userData) {
      res.render('index', userData);
    } else {
      res.render('index', {
        name: '',
        surname: '',
        age: '',
        email: '',
        city: '',
        password: '',
      });
    }
  }
});
```

```

    }
  } catch (error) {
    console.error(`Error rendering about page: ${error}`);
    res.status(500).send('Internal Server Error');
  }
});

app.post('/', async (req, res) => {
  userData = {
    name: req.body.name,
    surname: req.body.surname,
    age: req.body.age,
    email: req.body.email,
    city: req.body.city,
    password: req.body.password
  };

  try {
    await saveData(JSON.stringify(userData));
    console.log('Данные успешно сохранены');
  } catch (err) {
    console.log(`Ошибка сохранения: ${err}`);
  }

  res.redirect('/about');
});

app.get('/about', async (req, res) => {
  // try {
  //   const userData = await getData();
  //   res.render('about', userData);
  // } catch (error) {
  //   console.error(`Error rendering about page: ${error}`);
  //   res.status(500).send('Internal Server Error');
  // }
  try {
    const userData = await getData();

    if (userData) {
      res.render('about', userData);
    } else {
      res.render('about', {
        name: '',
        surname: '',
        age: '',
        email: '',
        city: '',
        password: '',
      });
    }
  } catch (error) {
    console.error(`Error rendering about page: ${error}`);
    res.status(500).send('Internal Server Error');
  }
});

app.get('/index', async (req, res) => {
  try {
    const userData = await getData();

    if (userData) {
      res.render('index', userData);
    } else {
      res.render('index', {
        name: '',
        surname: '',
        age: '',
        email: '',
      });
    }
  }
});

```

```

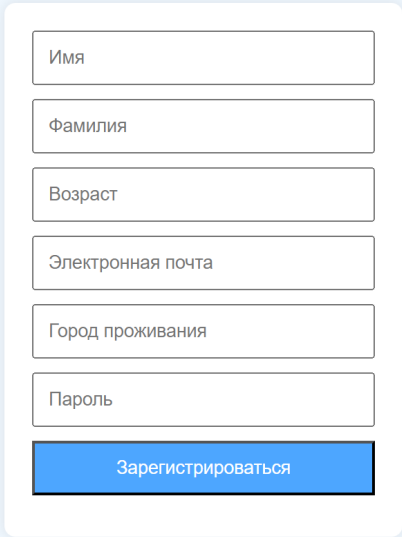
        city: '',
        password: '',
    });
}
} catch (error) {
    console.error(`Error rendering about page: ${error}`);
    res.status(500).send('Internal Server Error');
}
});

app.listen(PORT, () => {
    console.log(`Сервер запущен на ${PORT} порту ${LINK}`);
});

```

Листинг 1 – код программы.

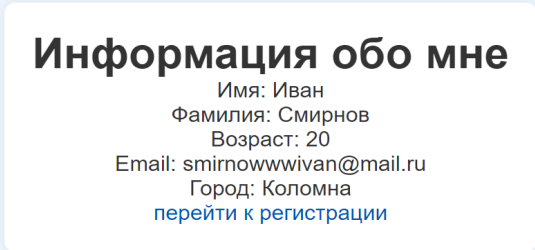
На странице отображается форма регистрации.



The image shows a registration form centered on a light blue background. The form is a white rounded rectangle containing several input fields and a button. The fields are labeled: 'Имя' (Name), 'Фамилия' (Surname), 'Возраст' (Age), 'Электронная почта' (Email), 'Город проживания' (City of residence), and 'Пароль' (Password). Below these fields is a blue button with the text 'Зарегистрироваться' (Register).

Рис. 1 – форма регистрации.

После регистрации переходим на информационную страницу.



The image shows an 'About me' information page centered on a light blue background. The page is a white rounded rectangle with the title 'Информация обо мне' (Information about me) in bold. Below the title, the user's details are listed: 'Имя: Иван' (Name: Ivan), 'Фамилия: Смирнов' (Surname: Smirnov), 'Возраст: 20' (Age: 20), 'Email: smirnowwivan@mail.ru', and 'Город: Коломна' (City: Kolomna). At the bottom, there is a blue link that says 'перейти к регистрации' (go to registration).

Рис. 2 – вывод хешированных данных.

Если данные в редисе не было, то выводится пустая страница.

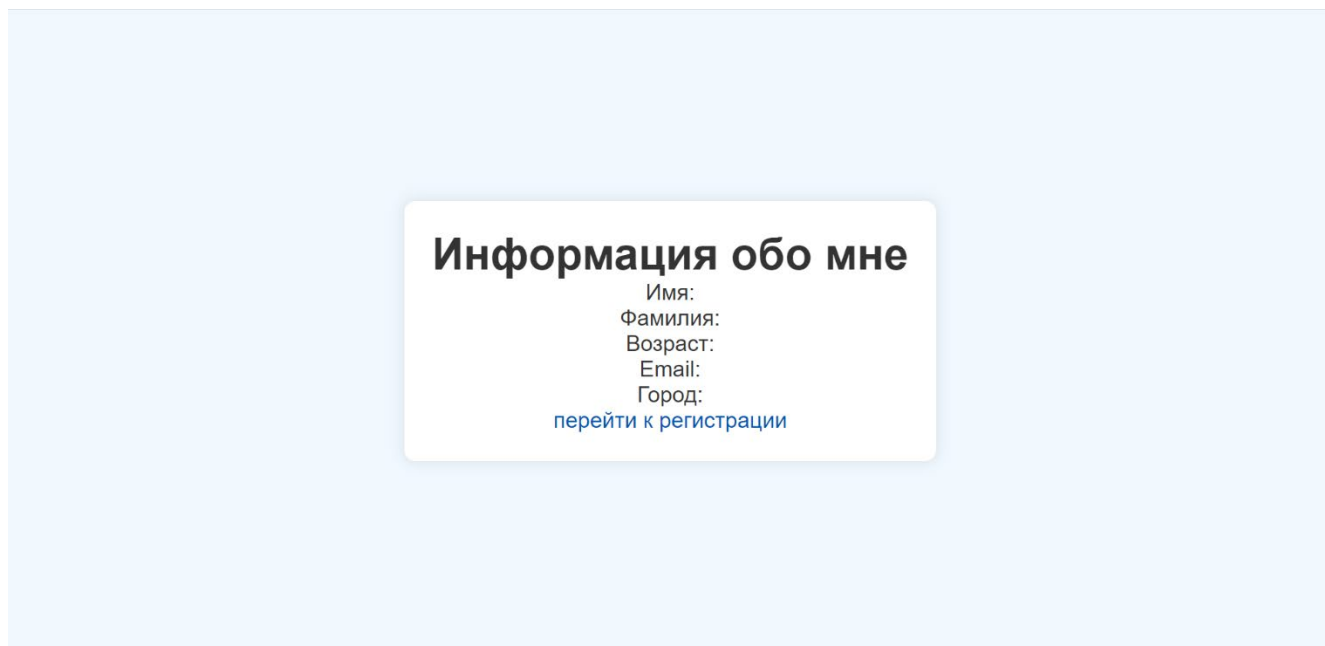


Рис. 3 – пустая страница.

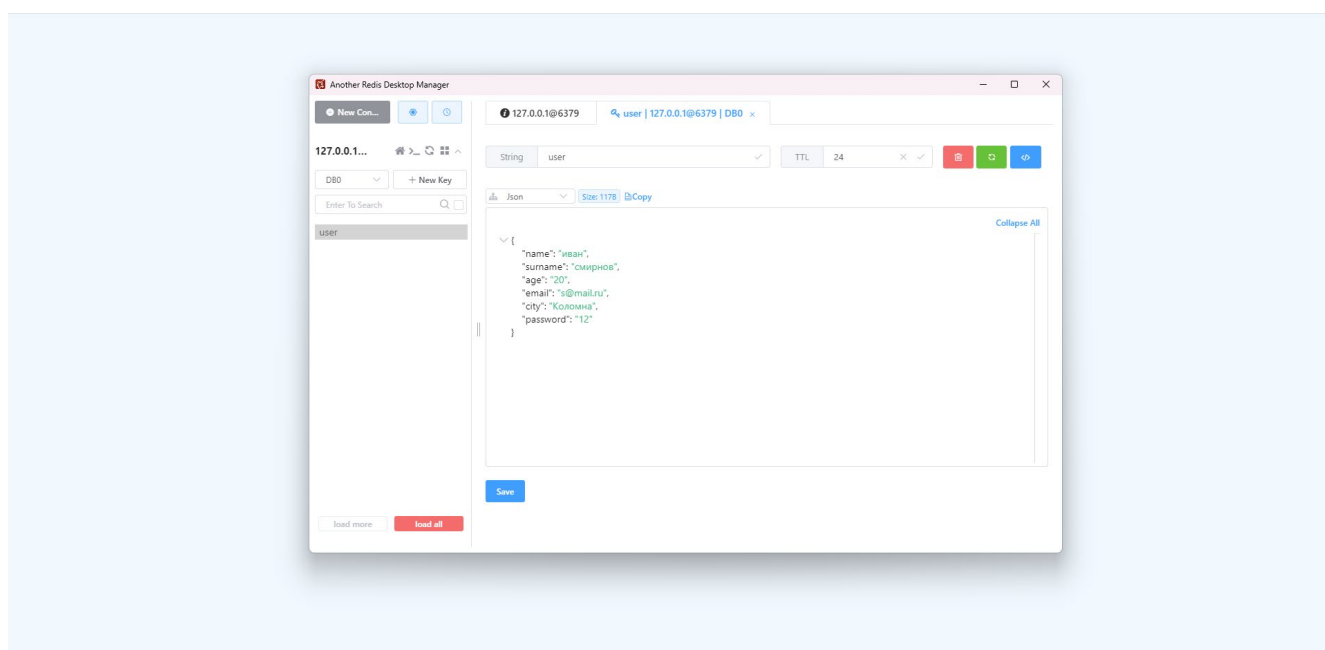


Рис. 4 – Redis.

Контрольные вопросы

В рамках практической работы, подготовьте ответы на следующие вопросы:

1. Каковы области применения Redis в информационных системах?

- Кэширование: Redis используется как высокопроизводительное кэширование данных, что позволяет ускорить доступ к часто используемым данным.
- Хранение сеансов: Redis может использоваться для хранения данных сеансов пользователей, обеспечивая быстрый доступ и масштабируемость.
- Очереди задач: Redis поддерживает структуры данных для организации очередей задач, что делает его подходящим для обработки асинхронных задач в распределенных системах.
- Реальное время (Real-time) аналитика: Использование структур данных Redis, таких как HyperLogLog и Bitmaps, позволяет эффективно выполнять операции для аналитики в режиме реального времени.

2. Какие ограничения имеет Redis в случае использования для обмена сообщениями между сервисами?

- Отсутствие гарантии доставки сообщений: Redis предоставляет асинхронную модель и не гарантирует, что сообщение будет доставлено.
- Отсутствие встроенного механизма обработки ошибок: Redis не предоставляет встроенных механизмов обработки ошибок, и разработчику нужно самостоятельно обеспечивать надежность обмена сообщениями.
- Отсутствие широких возможностей обработки сообщений сразу в нескольких сервисах: Redis предоставляет базовую поддержку

публикации/подписки, но для более сложных сценариев могут потребоваться дополнительные механизмы.

3. Какие библиотеки могут использоваться для программно-взаимодействия с Redis?

- StackExchange.Redis: Официальная библиотека для работы с Redis на платформе .NET, используемая в представленном коде.
- Jedis: Библиотека для языка Java.
- redis-py: Библиотека для языка Python.
- hiredis: Низкоуровневая библиотека на языке C для взаимодействия с Redis.

4. Какие настройки хранения данных предоставляет Redis?

- Типы данных: Redis поддерживает различные типы данных, такие как строки, хэши, списки, множества и т.д.
- Пространство ключей: Redis имеет глобальное пространство ключей, поэтому важно поддерживать уникальные ключи для различных наборов данных.
- Журналирование: Redis может быть настроен на выполнение журналирования (RDB и AOF) для сохранения данных на диск и обеспечения их восстановления после перезапуска.

5. Какими особенностями характеризуются хранилища «ключ-значение»?

- Простота и скорость: Хранилища "ключ-значение" обеспечивают простой интерфейс для хранения и извлечения данных, что обеспечивает высокую производительность.
- Гибкость: Ключи и значения могут быть различных типов данных, что обеспечивает гибкость при проектировании структур данных.
- Масштабируемость: Многие хранилища "ключ-

значение" могут легко масштабироваться горизонтально, что позволяет обрабатывать большие объемы данных и запросов

- Ограниченная функциональность: Хранилища "ключ-значение" могут быть ограничены функциональностью, поскольку они не предоставляют сложных запросов и операций, типичных для реляционных баз данных.

Вывод

В ходе данной практической работы мы успешно достигли поставленной цели – освоили применение хранилища "ключ-значение" Redis. Полученные знания о структуре и функциональности Redis позволят эффективно использовать это хранилище в различных сценариях, таких как кэширование данных, управление сессиями пользователей, организация очередей задач и другие. Опыт работы с основными типами данных в Redis, а также освоение библиотек и инструментов для взаимодействия с этим хранилищем, создадут надежную основу для разработки масштабируемых и высокопроизводительных приложений. В результате выполнения практической работы, мы укрепили понимание ключевых принципов работы с Redis и готовы применять их в будущих проектах, где требуется эффективное управление данными в формате "ключ-значение".