Online-Geschenk

Advanced Databases B1

UNDER THE GUIDANCE OF

Professor Frank Hefter

Submitted by Team Members - (Team Topaz)

Akhila Srinath (11014053)

Ashuthosh Manika GaneshKumar (11014060)

Shyam Lakhani (11014103)

Supritha Prakash(11014125)

Table of Contents

1. Introduction

We won't always be able to celebrate birthday, anniversaries, festivals with our near and dear ones, relatives and friends together because of the separate locations, time constraints or now due to the pandemic situation where practicing social distancing is a rule. So, in these situations or even situations where we want to surprise our dear ones, the best idea is to order a gift and deliver it at their doorstep.

Now when everyone is trying to stay home, a grocery or any product delivery at the doorstep is reducing the stress of public outings, so why not an online gift store to deliver gifts across the globe. And if we are far apart or if we are any situation mentioned above then an Online-Gifting Website is the way to go!!!

The application we have developed called **'Online-Geschenk'** is an online Gift ordering E-commerce website, where people can order gifts by selecting the occasion type for gifting. They can even get recommendations to choose a gift based on age, location, occasion, and person type. They can contact the sellers directly via the chatting option and customise their gift and the sellers or shop owners can also showcase their products.

2. Organization

2.1 Roles and Responsibilities

Tasks		Team Members		
	Akhila	Ashuthosh	Shyam	Supritha
Brainstorming	R	R	R	R
Topic Discussion with features, goals	R	R	R	R
Requirement Gathering	R	R	R	R
UML/Graph structure	R	R	R	R
Schema	R	R	R	R
Neo4j	R	C	I	R
Redis	I	C	R	I
MongoDB	R	R	I	R
Documentation & Report	R	R	R	R

R	A	C	I
Responsible	Accountable	Consulted	Informed

2.2 Meetings

No.	Date	Discussion	Outcome	Attendees
1	May 22. 2021	Brainstorming, researching about topic, features and goals	Finalised topic, features and goals	Akhila, Ashuthosh, Supritha and Shyam
2	May 22, 2021	Usecases and database discussions	Distributed tasks and usecases	Akhila, Ashuthosh, Shyam and Supritha
3	May 27, 2021	Usecase modifications	Changes and modified usecase based on Professor's inputs	Professor Frank Hefter, Akhila, supritha and Ashuthosh
4	June 7th, 2021	Neo4j usecases and Mongo DB combining	Modified nodes in neo4j for few usecases and professor suggested for combining MongoDB Collections together	Professor Frank Hefter, Akhila, supritha Ashuthosh and Shyam
5	June 10th, 2021	MongoDB Combining and discussing about Redis chat error	MongoDB working and cleared chat error in redis and finalized the lines in redis chat	Akhila, Ashuthosh, Shyam and Supritha
6	June 12th, 2021	Report discussions	Finalising draft report	Akhila, Ashuthosh, Shyam and Supritha
7	June 15th, 2021	Status presentation with professor	Finalising Project with changes	Professor Frank Hefter, Akhila, supritha Ashuthosh and Shyam

2.3 Tools Used:

- MongoDB Atlas – For MongoDB use cases.
- JavaScript (ES7), ReactJS – Front end design
- Neo4j Desktop Application – Neo4j Use cases.
- Redis CLI and Redis Server – Redis use cases.
- MS teams – used for Team meetings.
- MS Word – Documentation

3. User Stories

- During these unprecedented pandemic situations where it is impossible to meet near ones, friends and relatives, Mr. Ralf finds it difficult to meet them for any of the occasions and exchange the best wishes or gifts, so, he wants to at least surprise them with best wishes or gifts delivered at their doorstep – **Used Neo4j.**
- Ralf has now chosen a birthday gift for his wife's birthday as a photo frame and he wants to order it, but he would like to customize the photo frame **– Used Redis.**
- Ralf has been invited to his colleague Mathias farewell party and he has to gift him something. So, he needs some ideas to gift his colleague on his farewell day – **Used Neo4j.**
- Ralf liked his gift, and he would want to recommend it to other people as well so that it helps them to decide and buy – **Used MongoDB.**
- John is an owner of a small gift store but due to covid, no customers can come to his shop, so he now wants a platform to tell customers that his store is online -– **Used MongoDB.**
- John has opened another shop in a different location, and he wants it online as well with his previous shop – **Used Mongo DB**
- John has his stores online on the website, and he wants to upload his products and the occasions he sells it for on the website - **Used MongoDB**.
- John has his stores online on the website, and he has uploaded his products details but he now wants photos also to be uploaded and also he wants one of his products to be removed - **Used MongoDB**
- John business is under loss, and he wants to now close the stores which is online – **Used Mongo DB.**
- I, as a user now would like to get some recommendations for gifts for an occassion for my sister's kid who is 2years old and also for my sister of age 30years – **Used Neo4j.**
- I like looking at all the gift stores and the products they offer – **Used MongoDB.**
- I, as a seller now wants to give discounts on certain products – **Using Redis**
- I would like to only see gift stores which is highest popular – **Used Neo4j.**

4. Details

4.1 Akhila Srinath and Supritha Prakash (11014053 & 11014125)

4.1.1 User Story

During these unprecedented pandemic situations where it is impossible to meet near ones, friends and relatives, Mr. Ralf finds it difficult to meet them for any of the occasions and exchange the best wishes or gifts, so, he wants to at least surprise them with best wishes or gifts delivered at their doorstep.

4.1.2 Identified Use Cases – Implemented in Neo4j

1. Search for the occasion category (example: Birthday, Anniversary etc..).
2. Search for gift products based on the occasion sub-category (example: Greeting cards, chocolates etc... for birthday) directly and products are viewed.
3. Search gift stores at a particular location.
4. Search gift items based on gift stores at a particular location.
5. Search for gift items and the gift items based on locations, occasions and gift stores are displayed.

4.1.3 Actors - Users

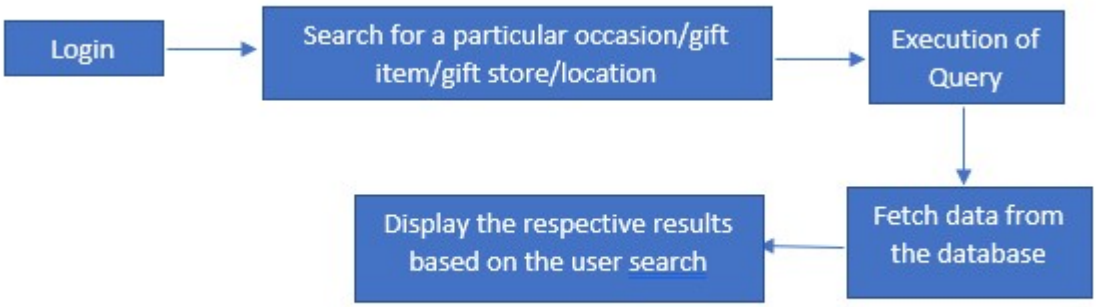
4.1.4 Detailed description

1. The first use case Interaction of the user and the database searches for the different occasions for which gifts are available. When the user opens the website and clicks on view all occasions, then the occasion list is displayed.
   (User → Birthday, Anniversary, Wedding, Parties etc…)
2. The second use case Interaction of the user and the database searches for the gift products based on a particular occasion selected by the user and displays the gift products based on the occasion selected by the user.
   (User → Birthday → Coffee Mugs, Greeting Cards, Chocolates, Photo Frames)
3. The third use case Interaction of the user and the database searches for the gift stores based on a particular location desired and typed by the user and displays

the gift stores based on the location.
(User →Mannheim → Tedi, Muller, Rossmann etc…)

4. The fourth use case Interaction of the user and the database displays the gift items based on a particular store desired by the user and displays the gift products.
(User → Tedi, Muller, Rossmann → Coffee Mugs, Greeting Cards etc…)
5. The fifth use case Interaction of the user and the database displays the gift items which the user searches and wants with the gift stores and the location it is based.
(User → Coffee Mugs → Tedi → Heidelberg)

4.1.5 Data Flow



4.1.6 Databases – Neo4j

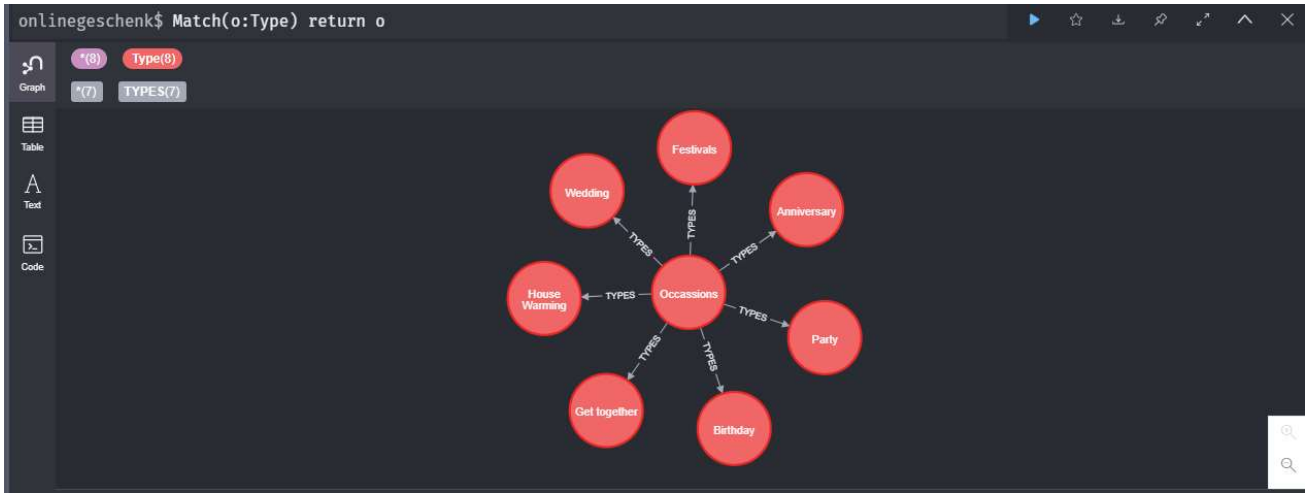
4.1.6.1 Databases used and why?

- Neo4j is a scalable, fast graph database, and we get high speeds results with the graph traversals from the nodes and the relationships connected. Hence, Neo4j can be easily used for searching and displaying the results.

4.1.6.2 Expressions used:

- **Use Case 1:**

Create (n:Type{name:'Occassions'})
Create (n:Type{name:'Birthday'})
Match(o:Type) return o

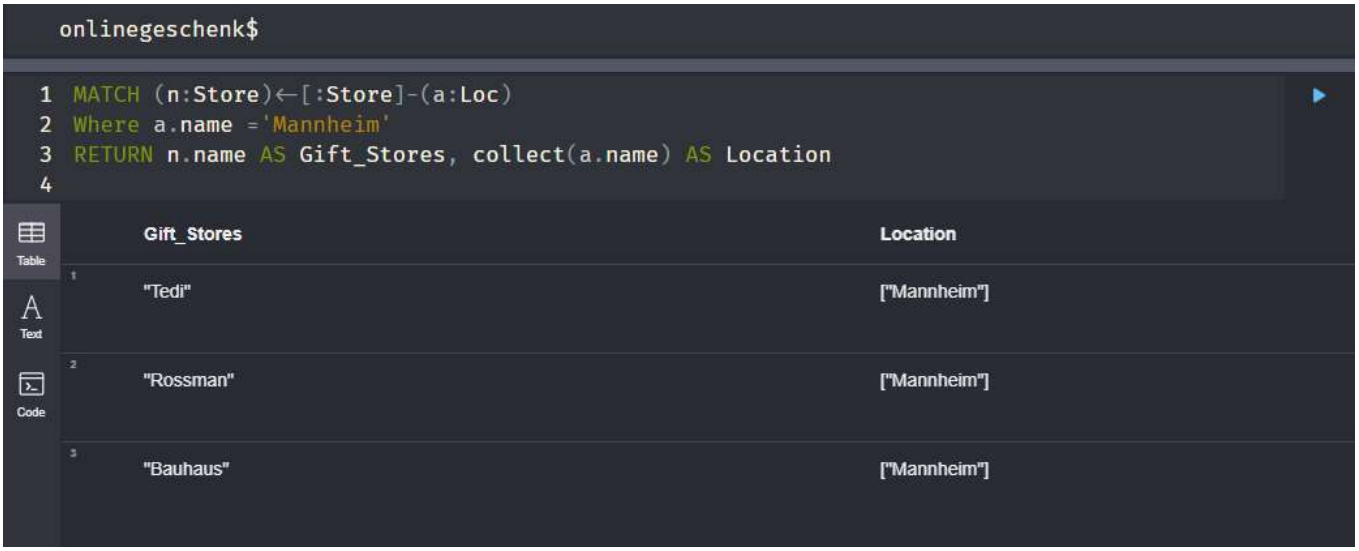


- **Use Case 2:**

Create (n:BirthdayGift{name:'Coffee Mugs'})
Match (n:BirthdayGift) return n

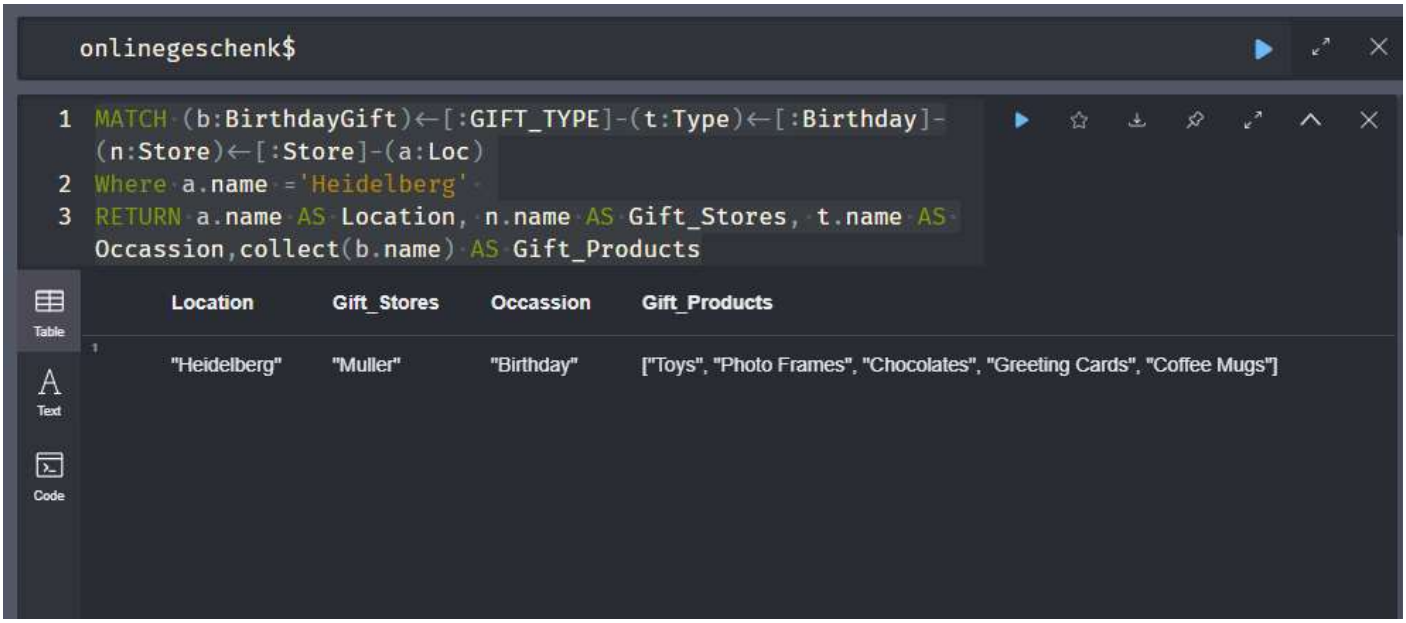
- **Use Case 3:**

MATCH (n:Store)<-[:Store]-(a:Loc)
Where a.name ='Mannheim'
RETURN n.name AS Gift_Stores, collect(a.name) AS Location



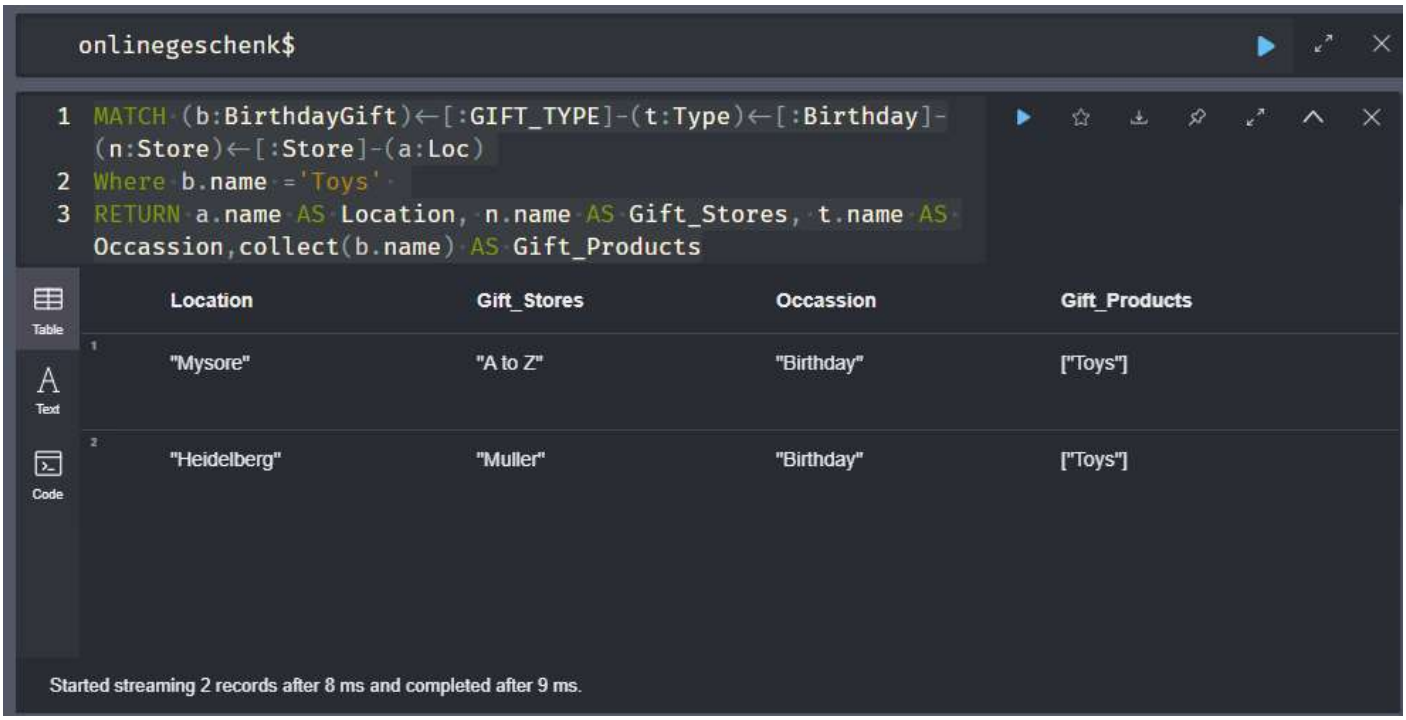
- **Use Case 4:**

MATCH (b: BirthdayGift) <-[:GIFT_TYPE]-(t:Type)<-[:Birthday]-(n:Store)<-[:Store]-(a:Loc) Where a.name ='Heidelberg'
RETURN a.name AS Location, n.name AS Gift_Stores, t.name AS Occassion, collect(b.name) AS Gift_Products



- **Use Case 5:**

MATCH (b:BirthdayGift)<-[:GIFT_TYPE]-(t:Type)<-[:Birthday]-(n:Store)<-[:Store]-(a:Loc)
Where b.name ='Toys'
RETURN a.name AS Location, n.name AS Gift_Stores, t.name AS Occassion,collect(b.name) AS Gift_Products



4.1.7 Outcome (what did you learn?)

Learning the largest and most vibrant community of graph database – Neo4j to deep search and get results for use cases.

4.2 Shyam Lakhani (11014103)

4.2.1 User Story

Ralf has now chosen a birthday gift for his wife's birthday as a photo frame and he wants to order it, but he would like to customize the photo frame.

4.2.1 Identified Use Cases – Implemented in Redis

1. Customers can contact the product seller directly via chat and provide him the details of their product customizations.

4.2.3 Actors

1. **Redis Chat** – Users & Seller

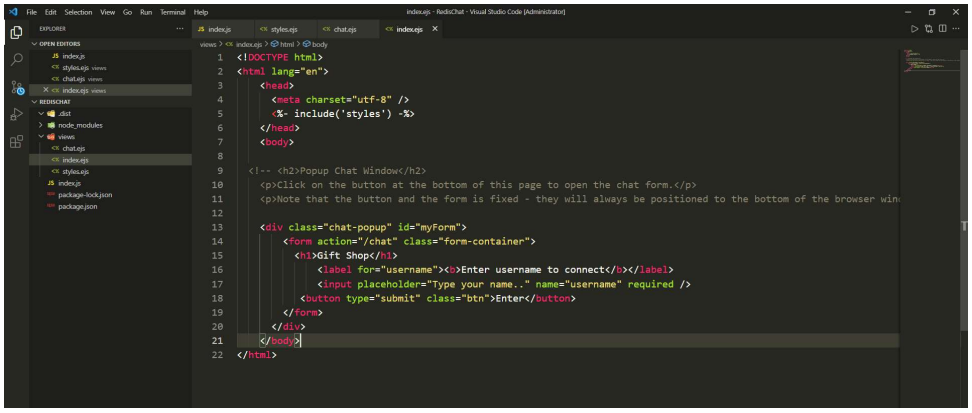
4.2.4 Detailed description

1. **Redis Chat**: A Customer needs to talk directly Seller about product he wants some customization so easily interact by real-time chat. A customer needs to enter his/her username and when seller enter by his/her username. Customer also notified that gift seller is connected and chat as real time. Both can interact so easily and discussed.

4.2.5 Optional Frontend used / Command lines to reproduce execution.

Redis Chat

❖ User Enter to chat by username.



❖ Notified by gift-shop enters & Real time Chat

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <%- include('styles') -%>
</head>
<body>
    <h1>Redis Chat</h1>
    <div id="messages">
        <input id="message" type="text" name="message" value="" placeholder="Message" />
        <img />
        <br />
        <br />
        <button onClick="emitData().value=''">Send</button>
    </div>
    <div id="joined"></div>
</body>

<script src="/socket.io/socket.io.js"></script>
<script>
    const socket = io("http://localhost:5000");

    function emitData(){
        const message = document.querySelector("#message").value;
        if (message.length > 0) {
            socket.emit("message",{ message, from:"<%= username %>"});
        }
    }

    socket.on("message", ({ from, message}) => {
        const messageElement = document.createElement("h5");
        messageElement.innerText = `${from}: ${message}`;
```

4.2.6: Data Flow

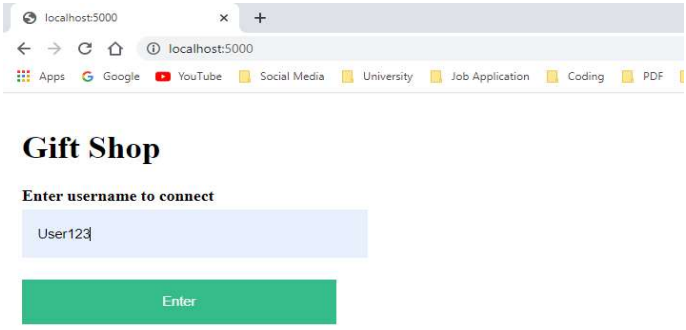
Redis Chat:

❖ Type code in Gitbash. (Make sure that Redis Server and Redis CLI is opened before type this code)

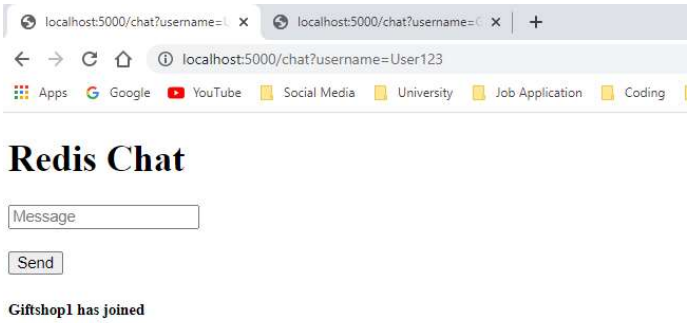
node index.js and will indicate to run code by localhost 5000

```
MINGW64:/c/Users/Administrator/Desktop/RedisChat
Administrator@DESKTOP-2RI2FNP MINGW64 ~/Desktop/RedisChat (master)
$ node index.js
Server at 5000
```

❖ User will enter by his/her username.

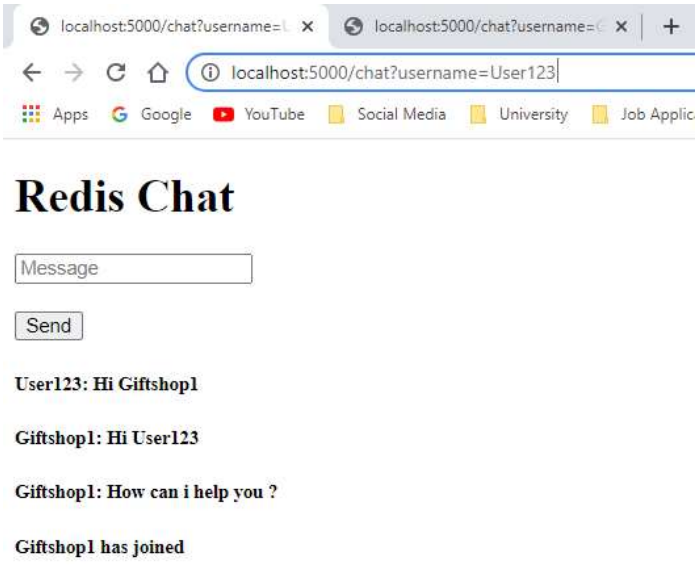


❖ User will be notified when Gift-shop seller joined to chat.

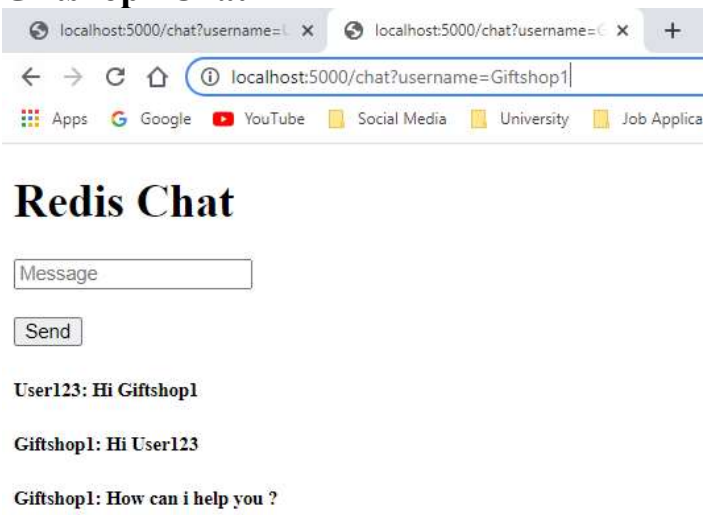


❖ Real-time Chat messages between User and Seller

User123 Chat



Giftshop1 Chat



❖ **<u>Redis Database storing and showing by Redis cli</u>**

Command: lrange messages 0 -1

4.2.7: Databases

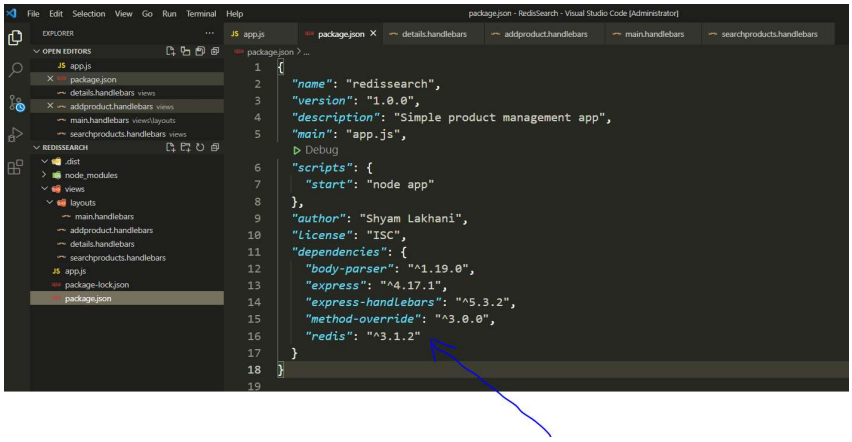
4.2.7.1 Why Redis database for Chat?

Redis is a pretty good choice as a database for a chat as it provides a couple of data structures that are not only very handy for various chat use cases but also processed in a really performant way.

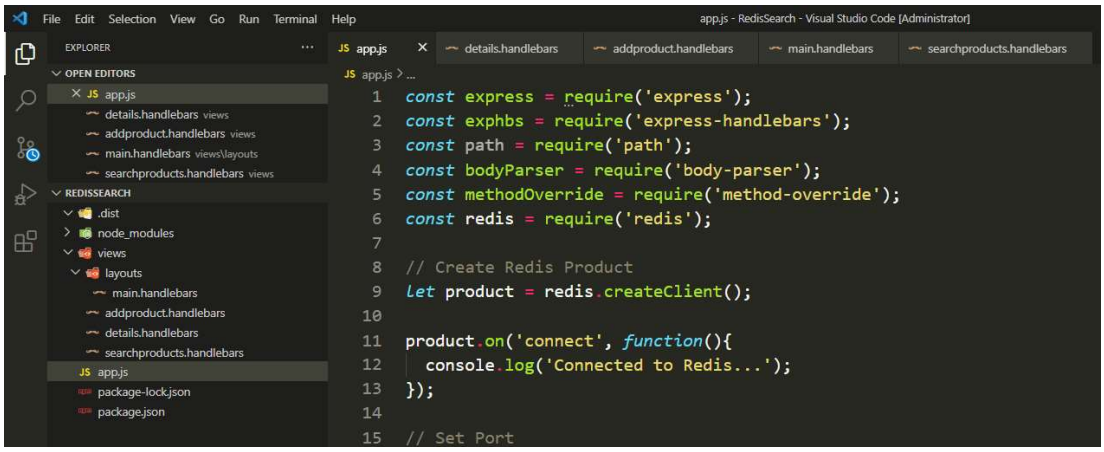
Redis is a very useful data service for tying microservices together and following the 12 factor app principles. For workloads focusing on rapidly changing ephemeral data sets where privilege control is not a concern (i.e. apps that you trust enough or less sensitive data) Redis is a strong choice for database.

4.2.7.2 Expressions used for this use case:

Before start entire coding, I install Redis and you can find redis along with version in package.json



Here I used in node.js and also create connection to show in console.log that connected to Redis…



4.2.8: Outcomes - What did you learn?

This powerful database is perfect for high performance jobs such as caching. Redis is a fast database for many different functions including as a cache or a message broker. I learned everything through Redis tutorial, which is the best place to progress from a newbie to an advanced user of Redis, the basic fundamentals of Redis such as the different data structures, various clients that work with Redis, different key-value pair commands (scan, config, commands, and client), how to persist data to disks and even the different methods of persisting data. After that, I build a functional working task how to actually work with Redis in a real-world example. I built a task manager using NodeJS and Redis. I also learnt how to incorporate Twitter Bootstrap for designing the manager.

4.3 Akhila Srinath and Supritha Prakash (11014053 & 11014125)

4.3.1 User Story

Ralf has been invited to his colleague Mathias farewell party and he has to gift him something. So, he needs some ideas to gift his colleague on his farewell day.

4.3.2 Identified Use Cases – Implemented in Neo4j

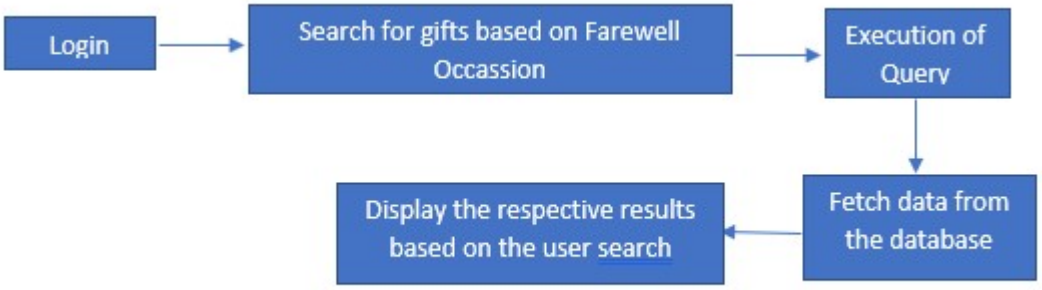
1. Searching for gift items based on Occasions and gift items are displayed.

4.3.3 Actors - Users

4.3.4 Detailed description

1. The first use case Interaction of the user and the database searches for the gift products based on the occasion selected 'farewell' by the user and displays the gift products based for Farewell.
   (User → Farewell → Bags)

4.3.5 Data Flow



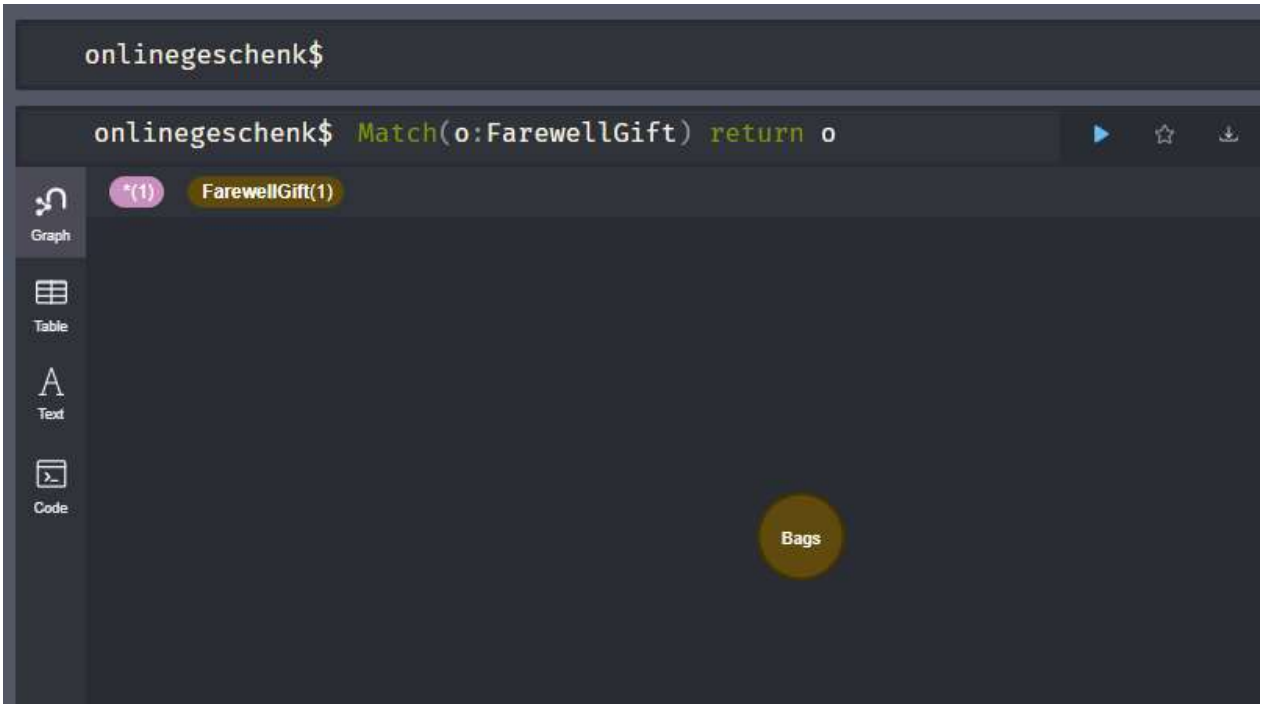
4.3.6 Databases – Neo4j

4.3.6.1 Databases used and why?

- Neo4j is a scalable, fast graph database, and we get high speeds results with the graph traversals from the nodes and the relationships connected. Hence, Neo4j can be easily used for searching and displaying the results.

4.3.6.2 Expressions used:

- Usecase1:
  Match (o: FarewellGift) return o



4.3.7 Outcome (what did you learn?)

Learning the largest and most vibrant community of graph database – Neo4j to deep search and get results for use cases.

4.4 Supritha Prakash (11014125)

4.4.1 User Story

John is an owner of a small gift store but due to covid, no customers can come to his shop, so he now wants a platform to tell customers that his store is online.

4.4.2 Identified Use Cases – Implemented in MongoDB

1. Registering the store on the website and creating the User /store profile.
2. Users can Login after they register on the website.

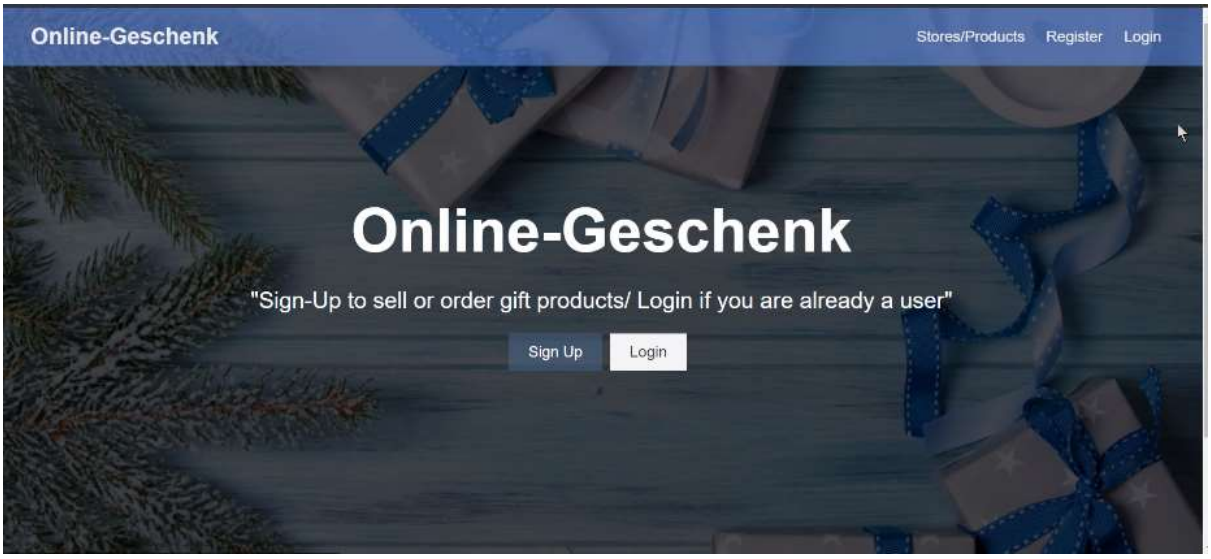
4.4.3 Actors - Users

4.4.4 Detailed description

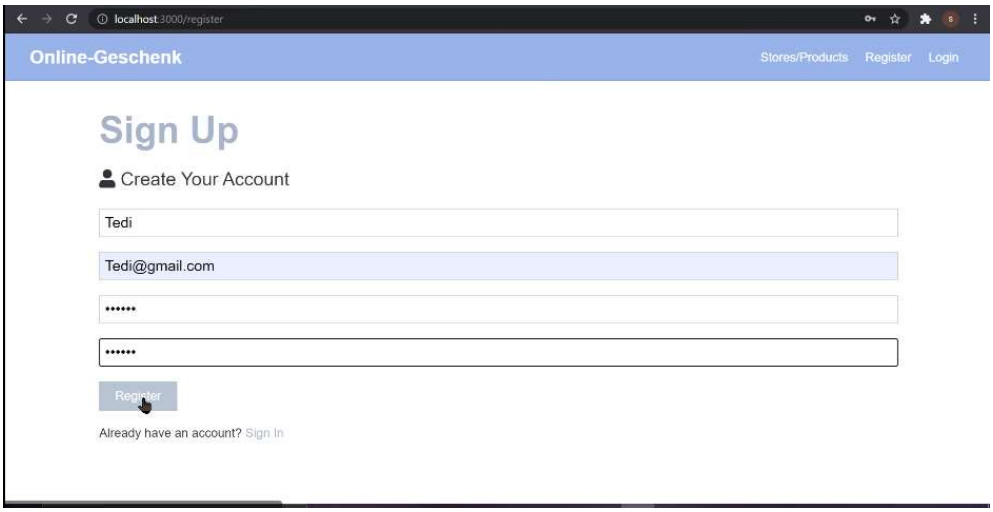
1. The first use case Interaction of the user and the database registers the user that is the store profile on the website with email, password, and username of the store.
2. The second use case logins the user/customer who is already registered on the website using his credentials that is email and password.

4.4.5 Optional: Frontend used / Command lines to reproduce execution:

Landing Page

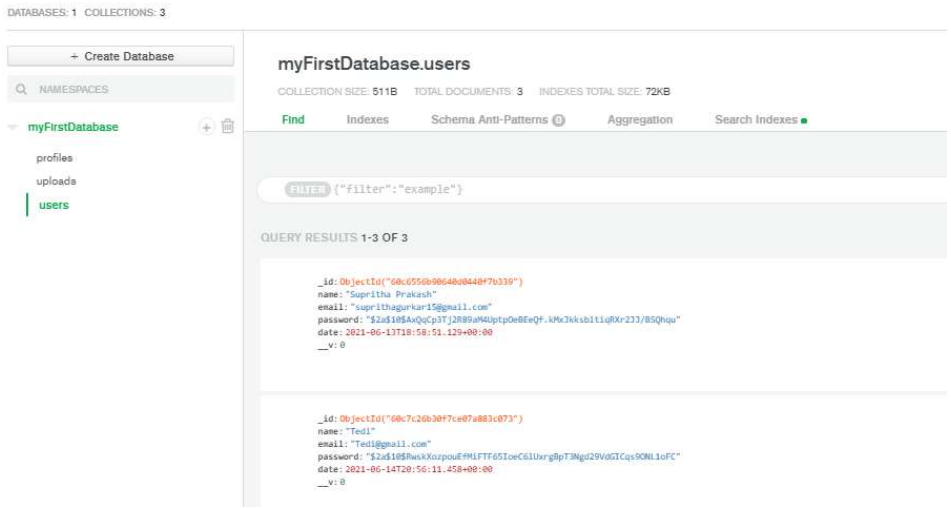


SignUp Page

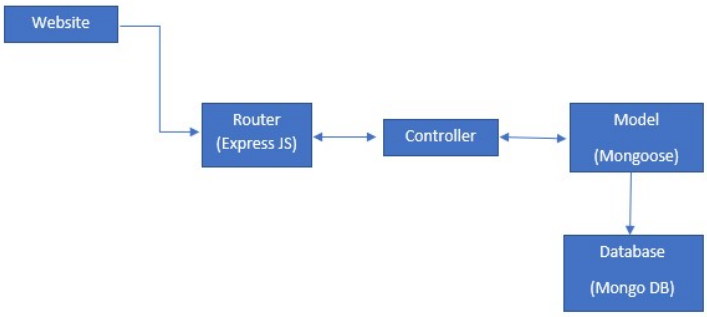


Login Page

MongoDB registered Users collections



4.4.6 Data Flow



4.4.7 Databases - MongoDB

4.4.7.1 Databases used and why?

- Used Mongodb to store passwords in hash version instead of storing it in user's plain text. If the database gets hacked in this case, the hacker will be left with random hash strings instead of plain text ones they could use to easily exploit the user's accounts. Addition to this, it helps to "salt" the password as well. as this is the additional data added to the hash function that will further secure the password.
- Mongodb returns a right response for inserting & deleting single or multiple documents into/from a collection.

4.4.7.2 Expressions used:

```
// @route    POST api/auth
// @desc     Authenticate user & get token
// @access   Public
router.post(
  "/",
  check("email", "Please include a valid email").isEmail(),
  check("password", "Password is required").exists(),
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const { email, password } = req.body;

    try {
      let user = await User.findOne({ email });

      if (!user) {
        return res
          .status(400)
          .json({ errors: [{ msg: "Invalid Credentials" }] });
      }

      const isMatch = await bcrypt.compare(password, user.password);
```

4.4.8 Outcome (What did you learn?)

- I learnt how powerful NOSQL databases are and the different API services that can be collabarated with Mongodb.
- Mongodb uses different servers to store information and retreive data which allows it to store large amount of information.
- Mongodb stores data in JSON format which enables users to work with different types of data in contrast with MYSQL, ORACLE databases which store information in rows and columns.

4.5 Supritha Prakash (11014125)

4.5.1 User Story

John has opened another shop in a different location, and he wants it online as well with his previous shop.

4.5.1 Identified Use Cases – Implemented in MongoDB

1. Adding stores with the all the details on the profile registered on the website.
2. Updating Store profile on the website or adding stores if there are any new stores opened by the user.

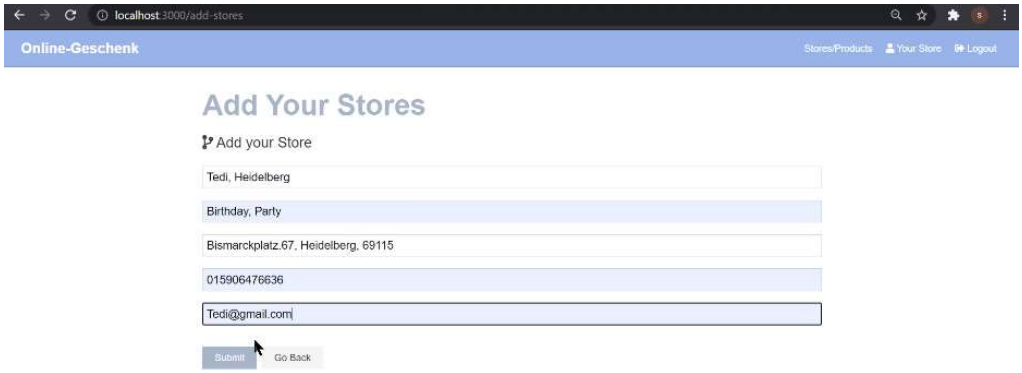
4.5.2 Actors - Users

4.5.3 Detailed description

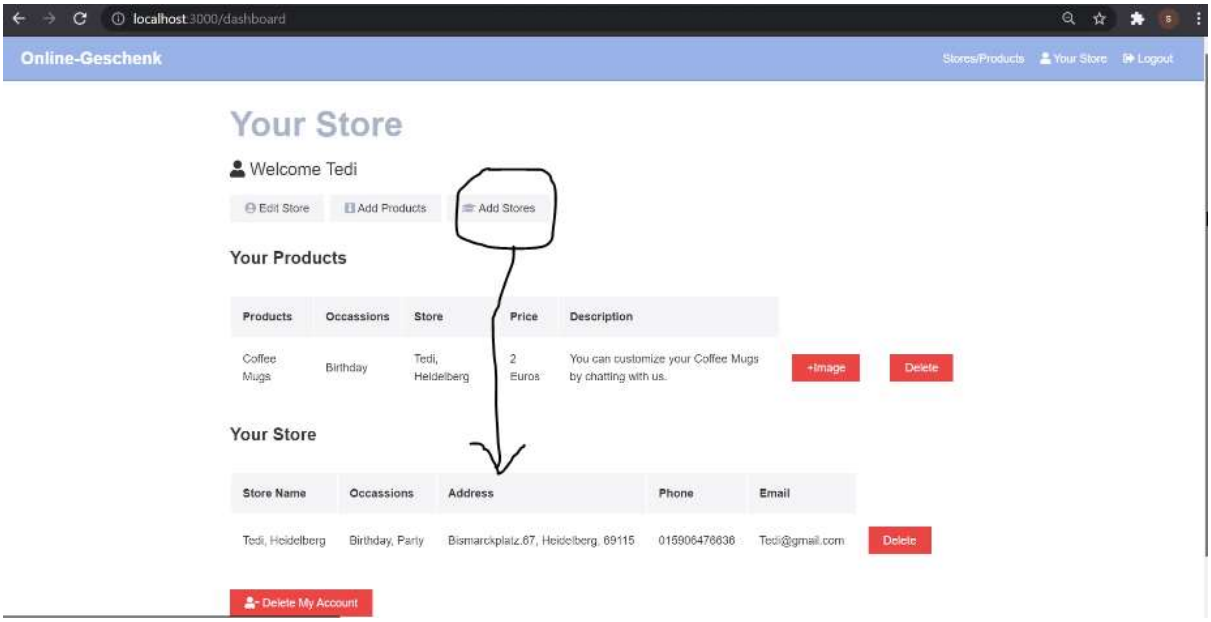
1. The first use case Interaction of the user and the database adds stores of the registered user on the stores page.
2. The second use case updates the user's stores if there is a new information added or the user can even add new stores for his profile.

4.5.4 Optional: Frontend used / Command lines to reproduce execution:

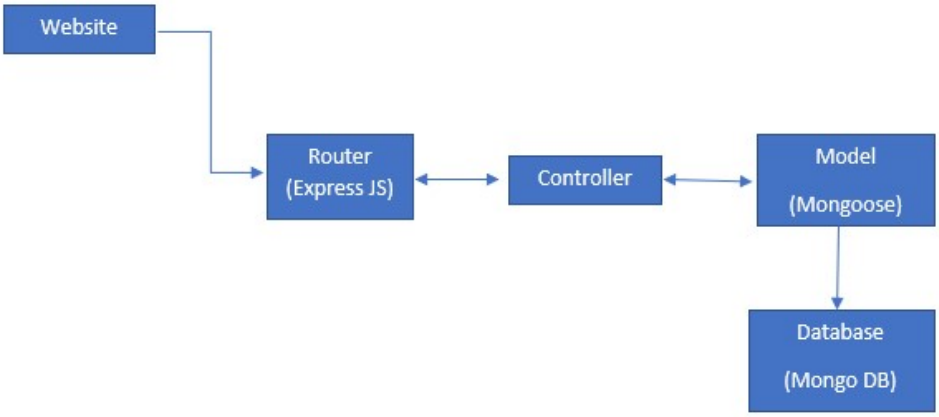
Adding Stores Page



Added Store is displayed on the Store Profile Page



4.5.6 Data Flow



4.5.7 Databases - MongoDB

4.5.7.1 Databases used and why?

- It's easy to store and retrieve model instances from a mongodb database using mongoose which automatically inherits a number of methods (such as create, save, remove and find).

4.5.7.2 Expressions used:

```
router.put(
  "/stores",
  [
    auth,
    [
      check("Occassions", "Occassions is required").not().isEmpty(),
      check("Address", "Address is required").not().isEmpty(),
    ],
  ],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const { SName, Occassions, Address, Phone, Email } = req.body;

    const newEdu = {
      SName,
      Occassions,
      Address,
      Phone,
      Email,
    };
```

4.5.8 Outcome (What did you learn?)

- Mongodb can also be easily deployed to the cloud when compared to other database.
- MongoDB has a serverless platform called Stitch that removes the need for web hosting and application services. With a Stitch SDK, we can run JavaScript code, access data stored in Atlas, authenticate end users, and more.

4.6 Ashuthosh Manika GaneshKumar (11014060)

4.6.1 User Story

John has his stores online on the website, and he wants to upload his products and the occasions he sells it for on the website.

4.6.2 Identified Use Cases – Implemented in MongoDB

1. User that is the seller can add Products online on his registered store profile under the section Products page, the seller can also add details (like store where he sells it, name of the product) of his products along with adding products.

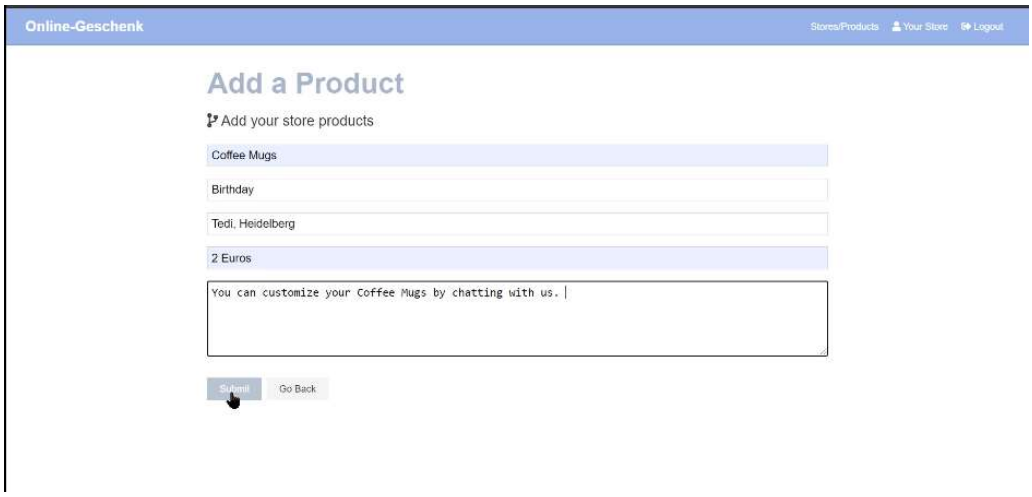
4.6.3 Actors - Users

4.6.4 Detailed description

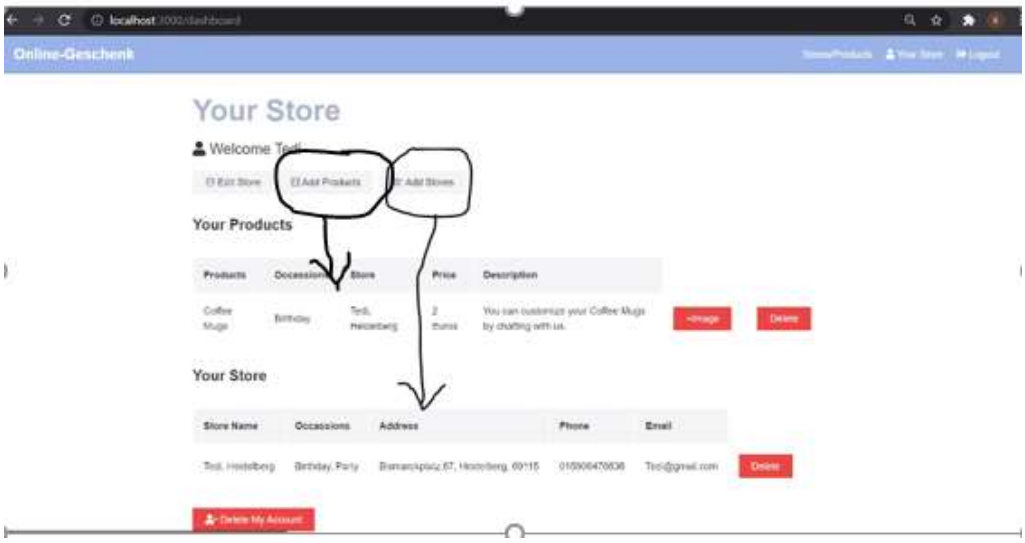
1. The first use Interactions of the user and the database describes the user registered on the website adding his products online on his store profile and the products he added will be displayed in his store profile. Here, he can give details about his product/s.

4.6.5 Optional: Frontend used / Command lines to reproduce execution:

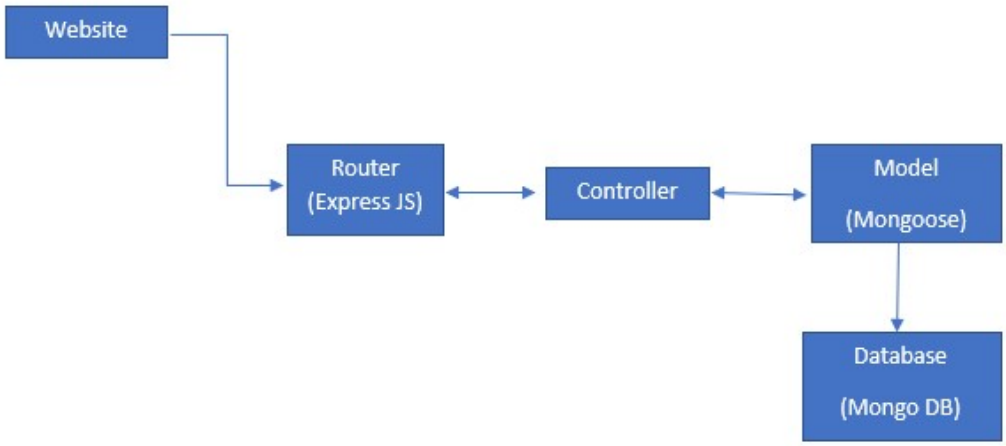
Adding Products Page



Added Products is displayed on the Store Profile Page



4.6.6 Data Flow



4.6.7 Databases - MongoDB

4.6.7.1 Databases used and why?

- MongoDB's dynamic schema capability allows for product documents to only contain attributes that are relevant to that product.

4.6.7.2 Expressions used:

```
router.put(
  "/products",
  [auth, [check("PName", "PName is required").not().isEmpty()]],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const { PName, Occassions, Store, Price, description } = req.body;

    const newExp = {
      PName,
      Occassions,
      Store,
      Price,
      description,
    };

    try {
```

4.6.8 Outcome (What did you learn?)

- Most mobile application development companies are dealing with varying data structures coming from multiple sources and potentially highly dynamic growth. The flexibility and scalability provide a great database solution for dealing with this type of environment. With schemas that can evolve over time mobile application developers don't have to spend time adjusting the database. Instead developers can focus on enhancing the customer experience.

4.7 Ashuthosh Manika GaneshKumar (11014060)

4.7.1 User Story

John has his stores online on the website, and he has uploaded his products details but he now wants photos also to be uploaded and also he wants one of his products to be removed.

4.7.2 Identified Use Cases – Implemented in MongoDB

1. User that is the seller can add Products online on his registered store profile under the section Products page, along with it he can also add images of his products on to his products page.
2. Deleting products which are out of stock or which are not available anymore.

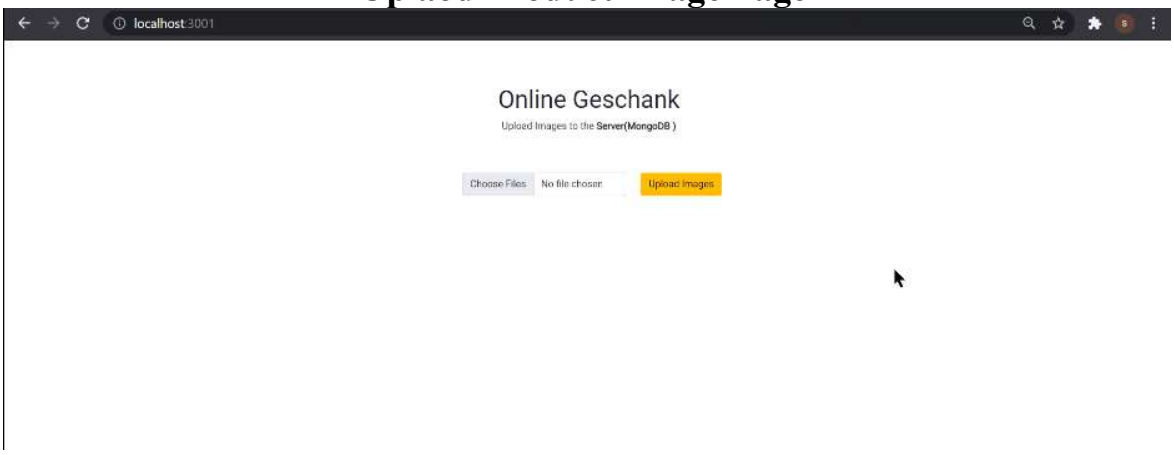
4.7.3 Actors - Users

4.7.4 Detailed description

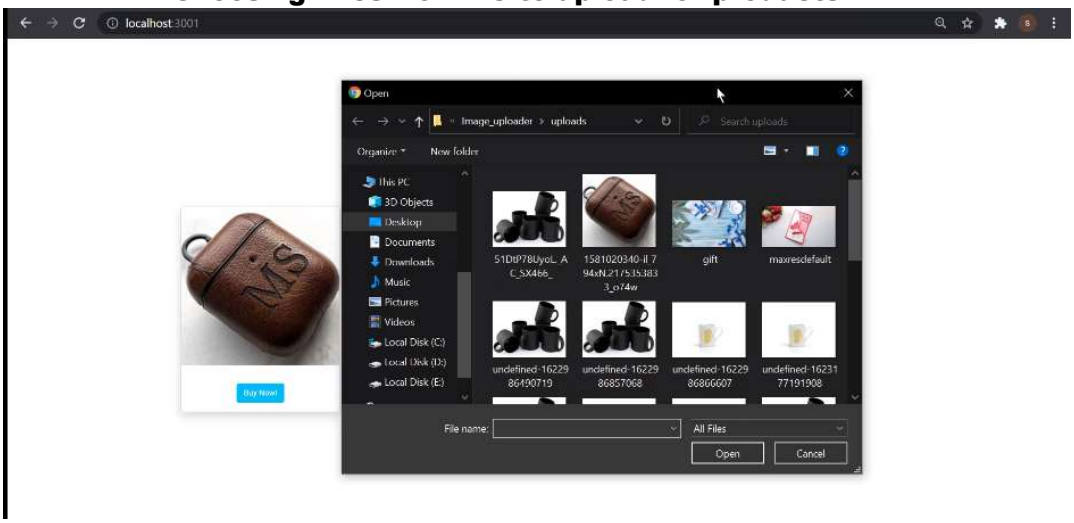
1. The first use Interactions of the user and the database describes the user registered on the website adding his products images online on his products profile and the product images he added will be displayed in his store profile.
2. The first use Interactions of the user and the database describes the user deleting his products uploaded already on the store.

4.7.5 Optional: Frontend used / Command lines to reproduce execution:

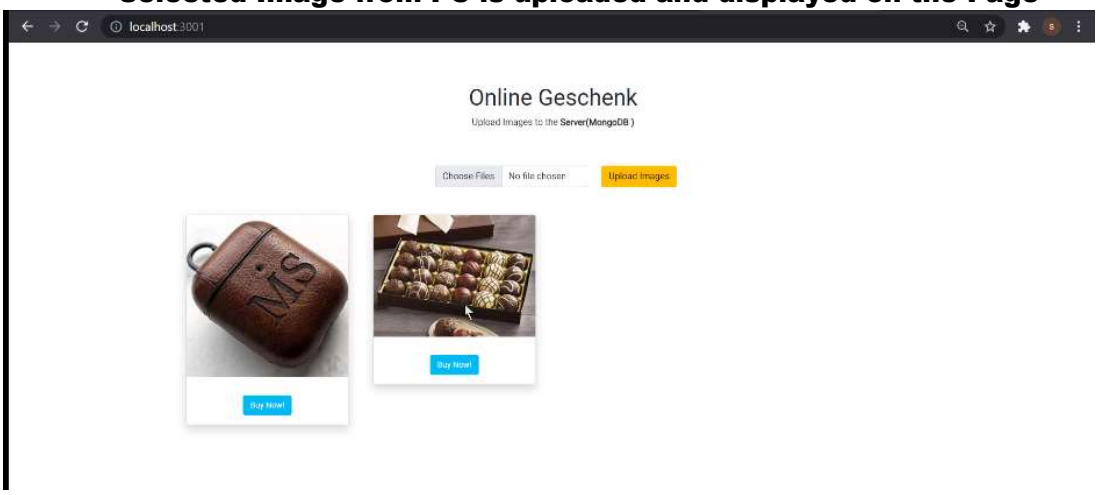
Uplaod Product Image Page



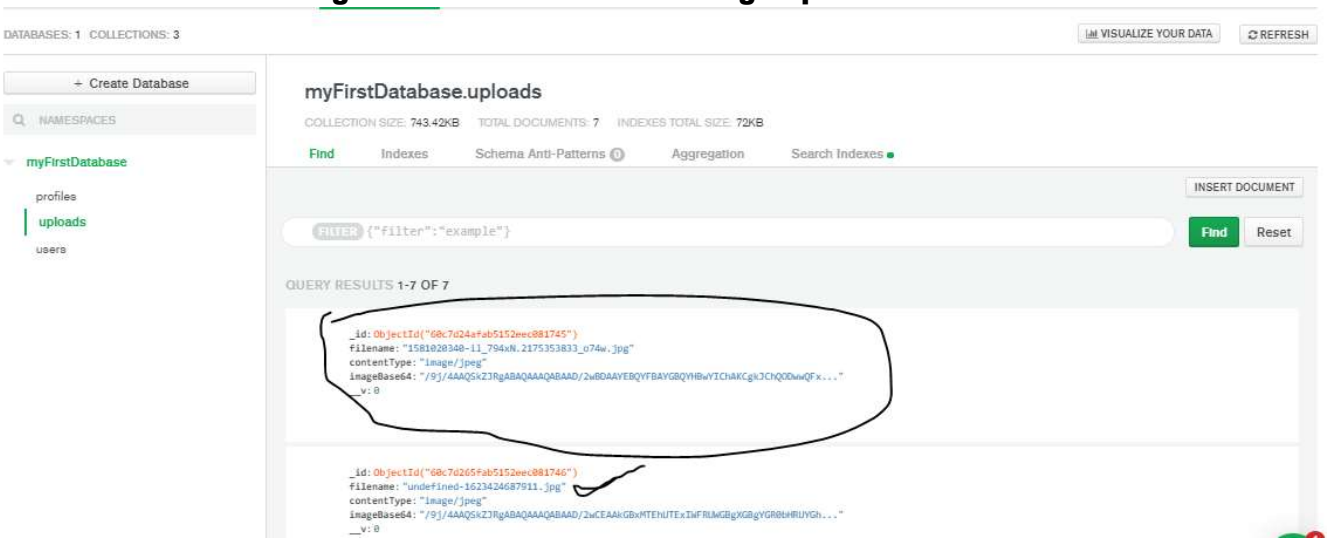
Choosing Files from PC to upload for products



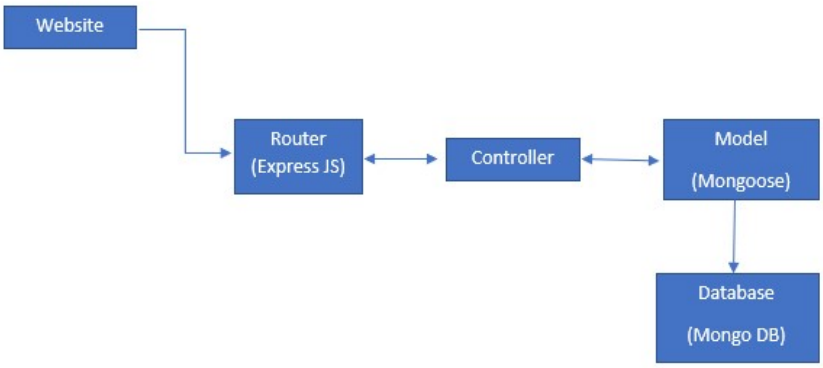
Selected Image from PC is uploaded and displayed on the Page



MongoDB Collections of the Image uplaods



4.7.6 Data Flow



4.7.7 Databases - MongoDB

4.7.7.1 Databases used and why?

- We used MongoDB because MongoDB provides a great tool to store many different types of objects with different sets of attributes.
- MongoDB users can very quickly and easily make changes to their catalogs providing better experience for developers and customers.

4.7.7.2 Expressions used:

```
router.delete("/products/:exp_id", auth, async (req, res) => {
  try {
    const profile = await Profile.findOne({ user: req.user.id });

    const removeIndex = profile.products
      .map((item) => item.id)
      .indexOf(req.params.exp_id);

    profile.products.splice(removeIndex, 1);
    await profile.save();

    res.json(profile);
  } catch (err) {
    console.error(err.message);
    res.status(500).send("Server Error");
  }
});
```

4.7.8 Outcome (What did you learn?)

- I leant how powerful NOSQL databases are, this can enhance the developer and user experience at a great level because of its horizontal scaling, document human easily readable JSON models, speed, accessibility, flexibility, transactions, handling Big Data.
- How MongoDB can be used for customer analytics, product data management, content management, mobile development, real time data Integration, broad objectives with evolving data requirements.

4.8 Akhila Srinath (11014053)

4.8.1 User Story

I like looking at all the gift stores and the products they offer – Used MongoDB.

4.8.2 Identified Use Cases – Implemented in MongoDB

1. Users who want to just view all the gift stores, can view it on the dashboard page on the website without even registering or login.

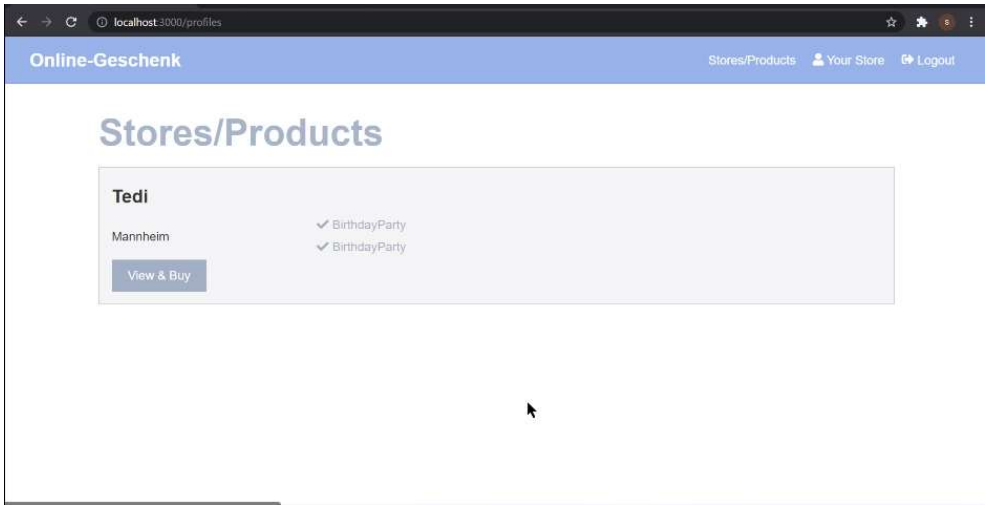
4.8.3 Actors - Users

4.8.4 Detailed description

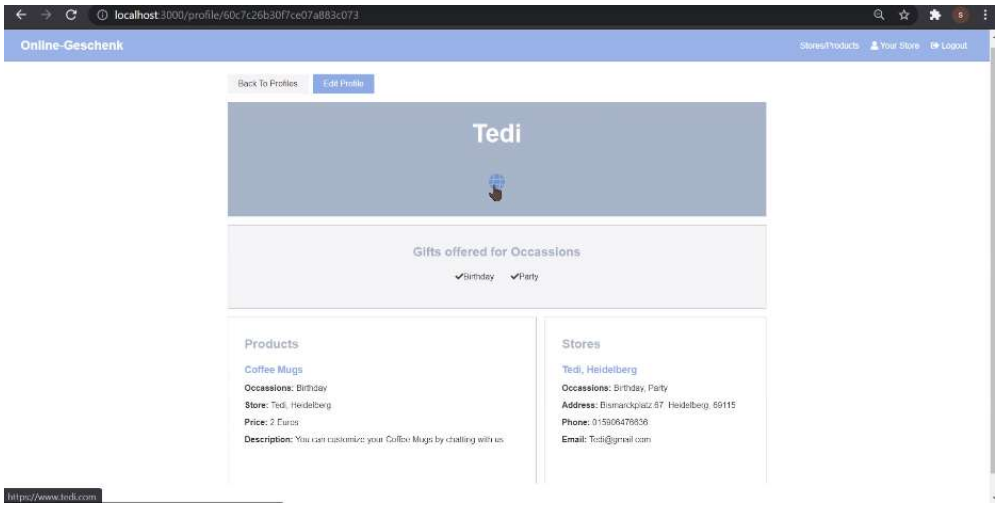
1. The first use case Interaction of the database and the user views all the gift stores, products the gift stores offer, the location of their stores on the dashboard page on the website called 'Stores/Products'. Clicking on the View and Buy button on each and every store on the dashboard will take user to the more detailed description page of the store.
   In the description page - He/ she can find all the information about the products, seller, stores and the website of the store for more information. This Dashboard page can be viewed even by users who don't want to register at first.

4.8.5 Optional: Frontend used / Command lines to reproduce execution:

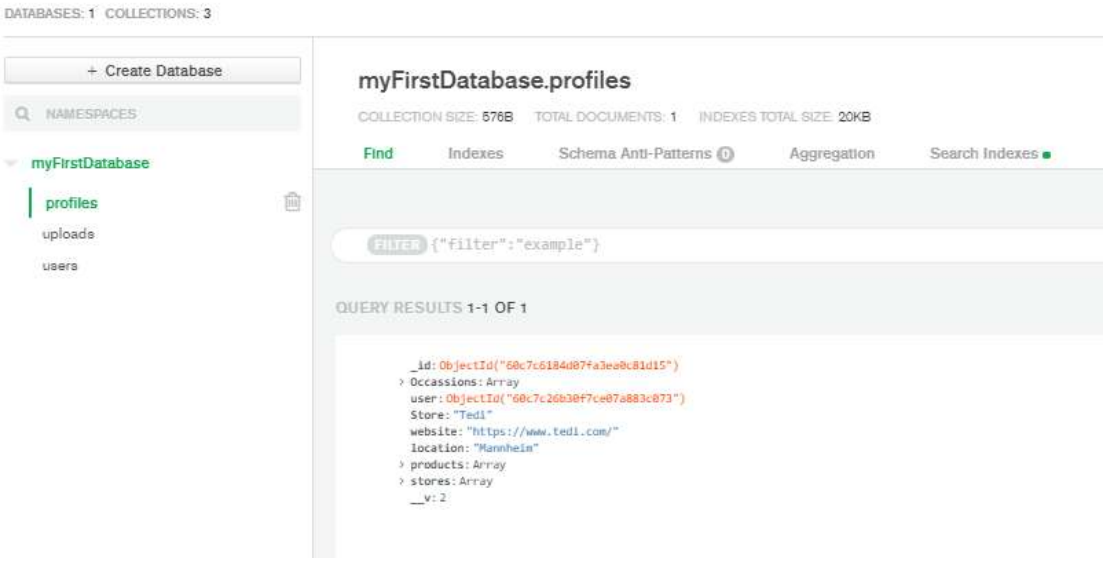
Dashboard page which displays all the stores registered as a user on the website



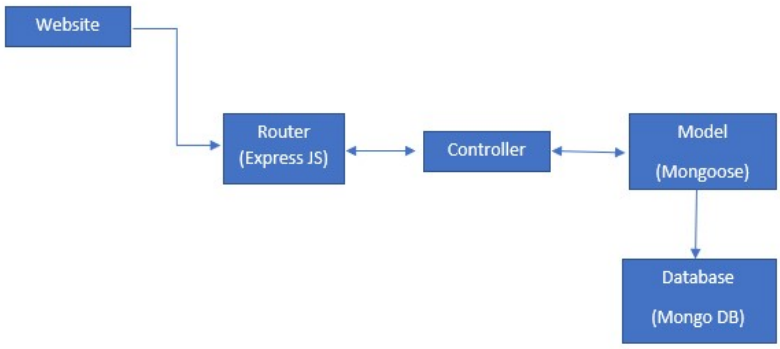
View and Buy Button clicking leads to the more detailed description Page of the store Profile



MongoDB Collection of the User Profiles



4.8.6 Data Flow



4.8.7 Databases - MongoDB

4.8.7.1 Databases used and why?

- The document data model - MongoDB is a powerful way to store and retrieve data very fast of the store profiles along with their store info, products attached with their profile ID.
- Supports JavaScript
- MongoDB Integrates large amounts of data and the flexibility and power of MongoDB document model made it easy for a unified single view.

4.8.7.2 Expressions used:

```
import React from "react";
import { Link } from "react-router-dom";

const DashboardActions = () => {
  return (
    <div className='dash-buttons'>
      <Link to='/edit-profile' className='btn btn-light'>
        <i className='fas fa-user-circle text-primary' /> Edit Store
      </Link>
      <Link to='/add-products' className='btn btn-light'>
        <i className='fab fa-black-tie text-primary' /> Add Products
      </Link>
      <Link to='/add-stores' className='btn btn-light'>
        <i className='fas fa-graduation-cap text-primary' /> Add Stores
      </Link>
    </div>
  );
};

export default DashboardActions;
```

```
const ProfileItem = ({
  profile: {
    user: { _id, name },
    Store,
    location,
    Occassions,
  },
}) => {
  return (
    <div className='profile bg-light'>
      <div>
        <h2>{name}</h2>
        {/* <p>
          {Store} {location && <span> at {location}</span>}
        </p> */}
        <p className='my-1'>{location && <span>{location}</span>}</p>
        <Link to={`/profile/${_id}`} className='btn btn-primary'>
          View & Buy
        </Link>
      </div>
      <ul>
        {Occassions.slice(0, 4).map((Occassion, index) => (
```

```
const Profiles = ({ getProfiles, profile: { profiles, loading } }) => {
  useEffect(() => {
    getProfiles();
  }, [getProfiles]);

  return (
    <Fragment>
      {loading ? (
        <Spinner />
      ) : (
        <Fragment>
          <h1 className='large text-primary'>Stores/Products</h1>

          <div className='profiles'>
            {profiles.length > 0 ? (
              profiles.map((profile) => (
                <ProfileItem key={profile._id} profile={profile} />
              ))
            ) : (
```

4.8.8 Outcome (What did you learn?)

- JSON Document model of MongoDB, very easy to understand.
- Storing, managing and retrieving data using MongoDB
- Learnt about MongoDB Key-Value pairs i.e Object ID attached to each object.

4.9 Akhila Srinath and Supritha Prakash (11014053)

4.9.1 User Story

I, as a user now would like to get some recommendations for gifts for a birthday party occasion for my sister's kid who is 2years old and also for my sister on her farewell day of age 30years – Used Neo4j.

4.9.2 Identified Use Cases – Implemented in Neo4j

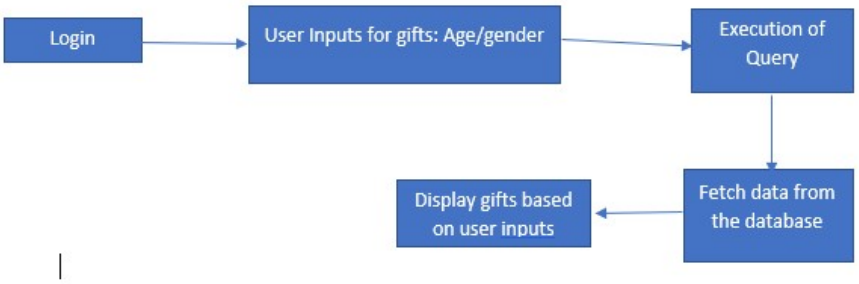
1. Users can get recommendations for gifts for any occasion based on age.
2. Users can get recommendations for gifts for any occasion based on gender and also can get recommendations on age and gender together.

4.9.3 Actors - Users

4.9.4 Detailed description

1. The first use case Interaction of the database and the user describes the user Interaction with the online website, here in this use case the user provides his/her desired occasion: birthday type and inputs the age as '2' and the gifts for the birthday occasion for a 2 year kid are displayed.
2. The second use case Interaction of the database and the user describes the user Interaction with the online website, here in this use case the user provides his/her desired occasion: farewell type and inputs the gender as 'female' and age as '30' and the gifts for the farewell occasion for a 30-year-old female are displayed.

4.9.5 Data Flow



4.9.6 Databases – Neo4j

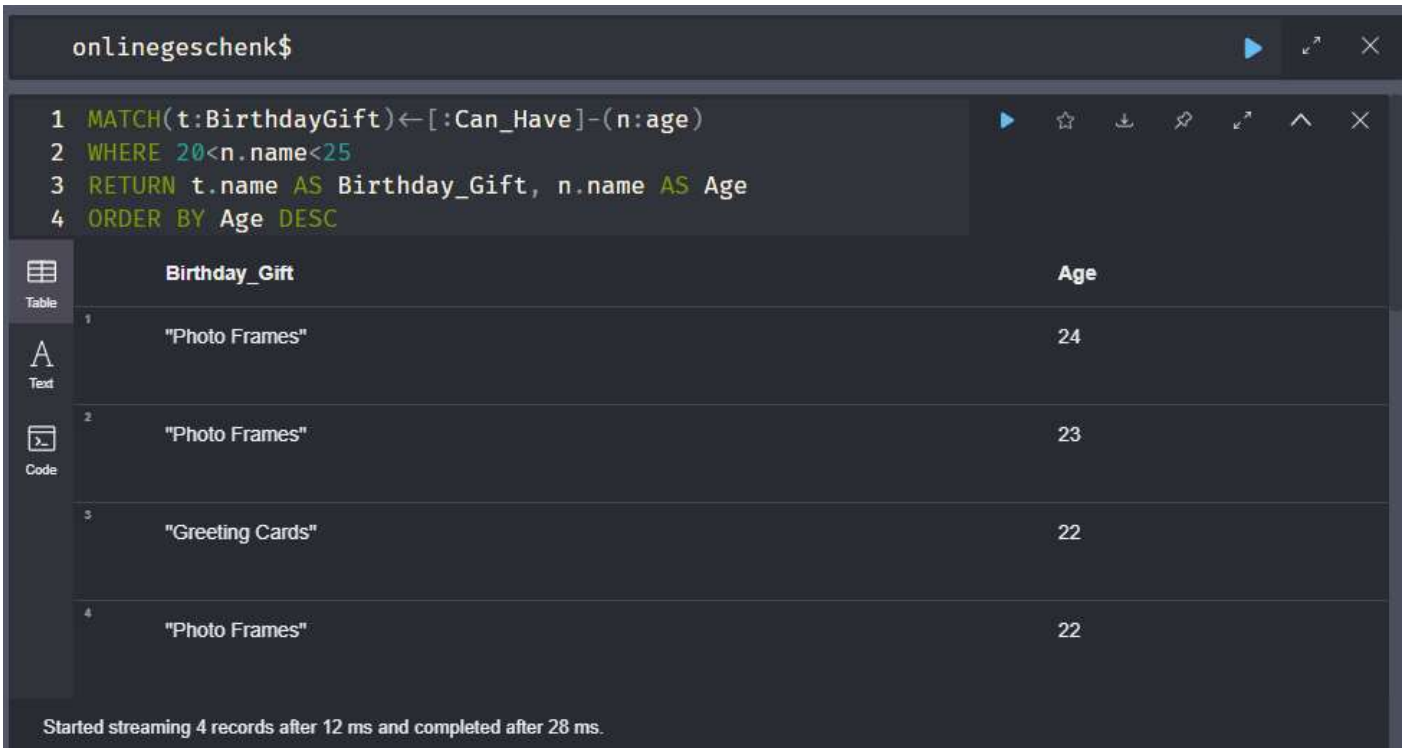
4.9.6.1 Databases used and why?

- Index-free adjacency shortened our read time and got better as the data complexity grew. Got reliably fast transactions with high throughput.
- To get recommendations Neo4j was the best as it could handle the data complexity and provide us with the recommendation query results.

4.9.6.2 Expressions used:

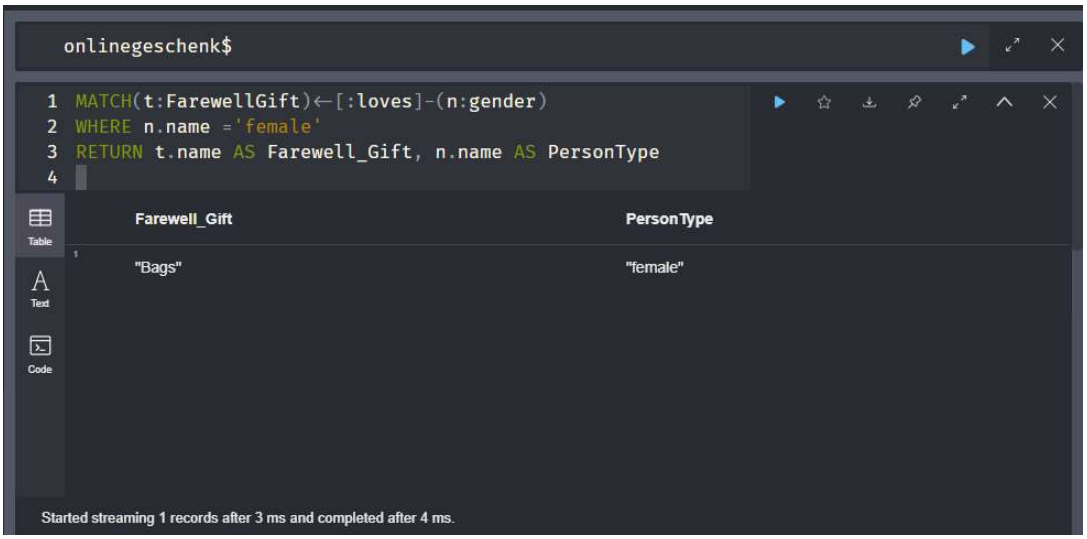
- **Usecase 1:**

```
MATCH(t:BirthdayGift)<-[:Can_Have]-(n:age)
WHERE 20<n.name<25
RETURN t.name AS Birthday_Gift, n.name AS Age
ORDER BY Age DESC
```



- **Usecase 2:**

MATCH(t:FarewellGift)<-[:loves]-(n:gender) WHERE n.name ='female' RETURN t.name AS Farewell_Gift, n.name AS PersonType



- **Usecase 3:**

MATCH(t:BirthdayGift)<-[:Can_Have]-(n:age)<-[:AGED]-(s:gender) WHERE s.name ='female' RETURN t.name AS Birthday_Gift, n.name AS Age, s.name AS gender

4.9.8 Outcome (what did you learn?)

- Learnt Cypher – Neo4j's graph query language.
- ACID compliance to ensure predictability of relationship-based queries.

4.10 Akhila Srinath and Supritha Prakash (11014053)

4.10.1 User Story

I would like to only see gift stores which is highest popular.

4.10.2 Identified Use Cases – Implemented in Neo4j

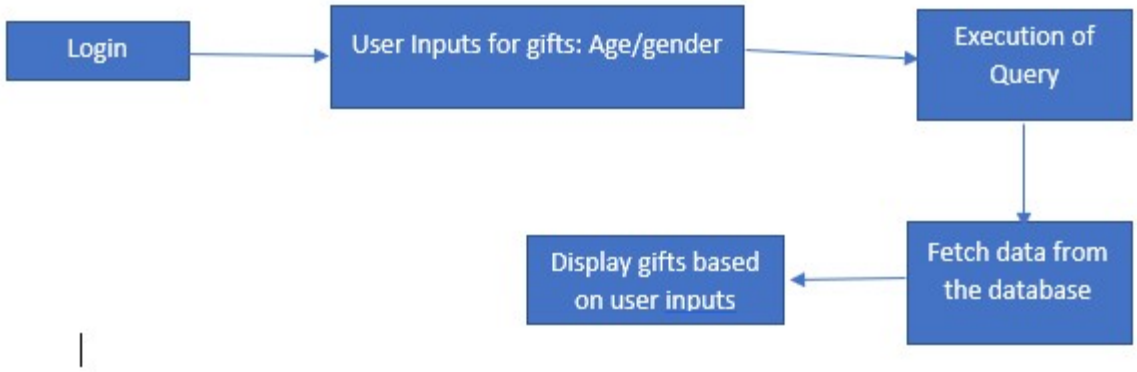
1. Users can get stores which are highest popular.

4.10.3 Actors - Users

4.10.4 Detailed description

1. The first use case Interaction of the database and the user displays the results for the user of the highest popular store that is it displays the average highest ratings given by the user to the stores. Here, the user can also see and set the store and the location.

4.10.5 Data Flow



4.10.6 Databases – Neo4j

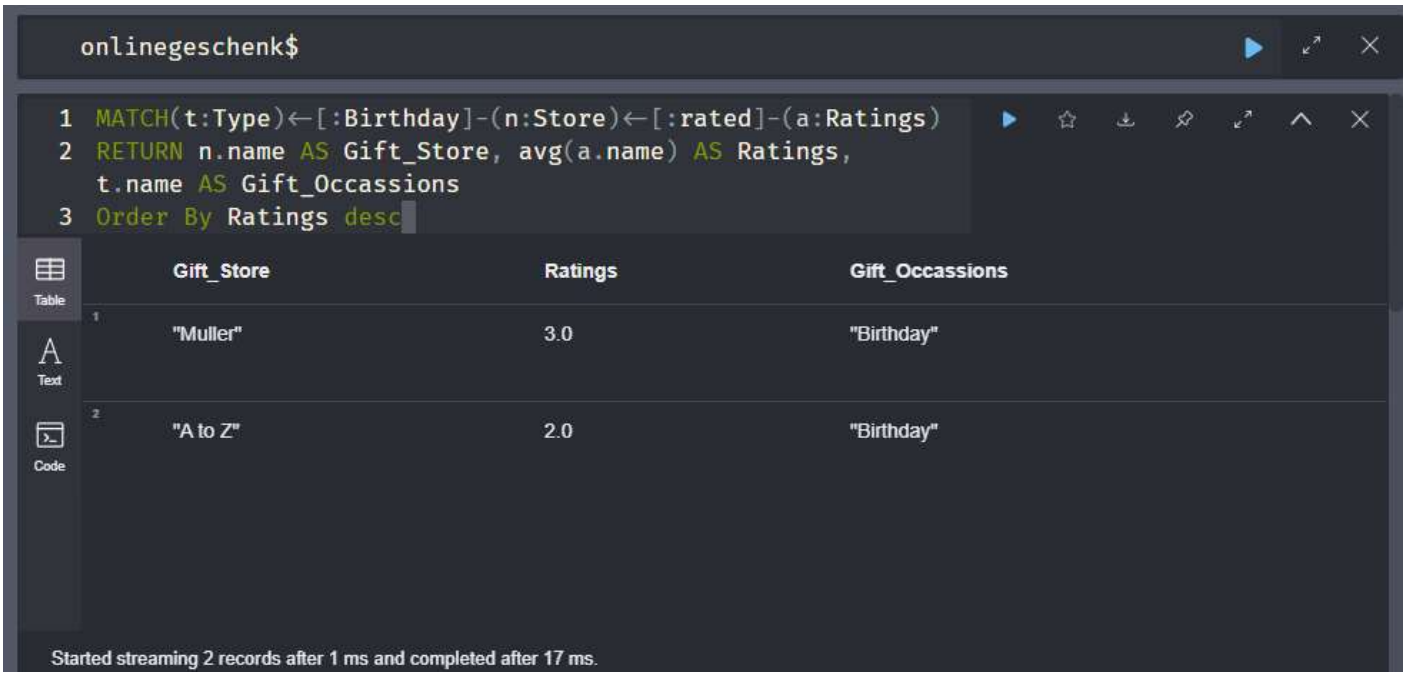
4.10.6.1 Databases used and why?

- Neo4j is a scalable, fast graph database, and we get high speeds results with the graph traversals from the nodes and the relationships connected. Hence, Neo4j can be easily used for searching and displaying the results.
- To get recommendations Neo4j was the best as it could handle the data complexity and provide us with the recommendation query results.

4.10.6.2 Expressions used:

- Usecase 1:

```
MATCH(t:Type)<-[:Birthday]-(n:Store)<-[:rated]-(a:Ratings)
RETURN n.name AS Gift_Store, avg(a.name) AS Ratings, t.name AS Gift_Occassions
Order By Ratings desc
```



4.10.7 Outcome (what did you learn?)

- Learnt Cypher – Neo4j's graph query language.
- ACID compliance to ensure predictability of relationship-based queries.

4.11 Shyam Lakhani (11014103)

4.11.1 User Story

I, as a seller now wants to give discounts on certain products.

4.11.2 Identified Use Cases – Implemented in Redis

1. Sellers can put discounts on their products already uploaded on their profile.

4.11.3 Actors

1. Search Product & Add discount on products - Seller

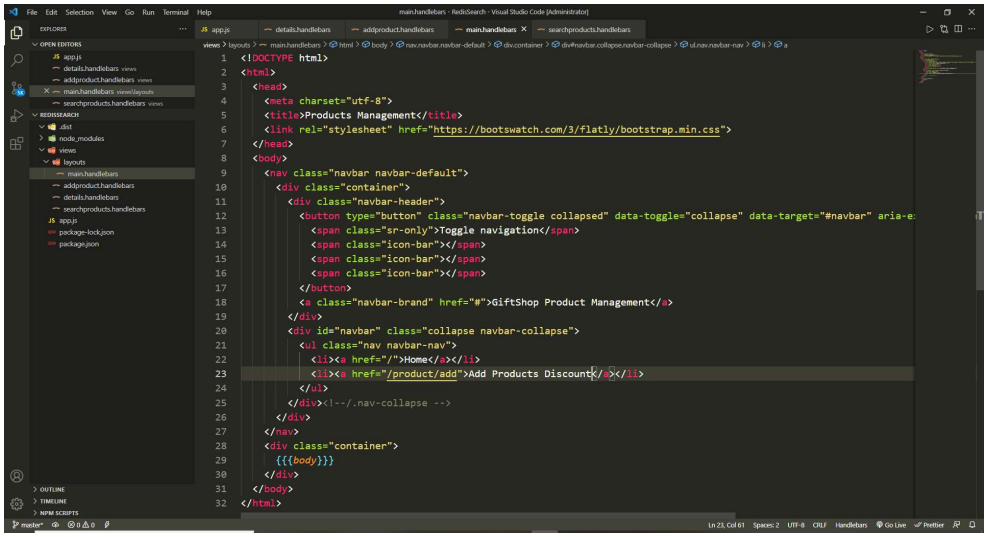
4.11.4 Detailed description

1. **Search Product & Add discount on products**: A seller can search by product id and identified about product details (Discount percentage & Valid discount till date). Seller can add new product and enter as per discount details (product name, Percentage of discount & availability for discount). Apart from that, seller can delete any product to manage status for product.

4.11.5 Optional Frontend used / Command lines to reproduce execution.

<u>Search Product & Add discount on products</u>:

❖ Home Page :



❖ Add discount Page.

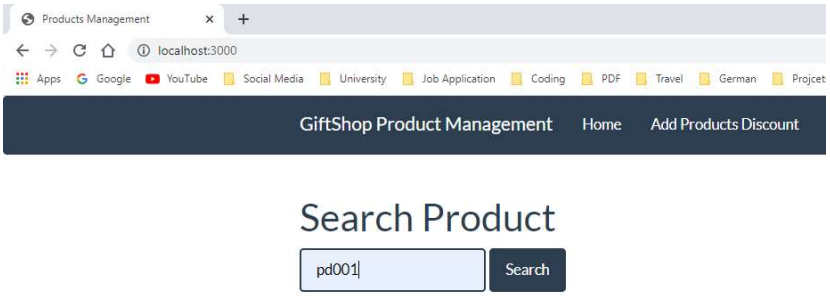
```
<h1>Add Discount on your Products</h1>
{{#if error}} <span>{{error}}</span>{{/if}}
<form method="POST" action="/product/add">
  <div class="form-group">
    <label>ID</label>
    <input type="text" name="id" class="form-control" placeholder="Add Product ID">
  </div>
  <div class="form-group">
    <label>Product Name</label>
    <input type="text" name="product_name" class="form-control" placeholder="Product Name">
  </div>
  <div class="form-group">
    <label>Discount</label>
    <select name="discount" class="form-control">
      <option value="discount">Discount</option>
        <option>10%</option>
        <option>20%</option>
        <option>30%</option>
        <option>35%</option>
    </select>
  </div>
  <div class="form-group">
    <label>Valid Till</label>
      <input type="date" name="product_date" class="form-control" id="date-picker">
  </div>
  <script>
        var date = new Date();
        var year = date.getFullYear();
        var month = date.getMonth()+1;
        var todayDate = String(date.getDate()).padStart(2,'0');
        var datePattern = year + '-' + month + '-' + todayDate;
        document.getElementById("date-picker").value = datePattern;
```

4.11.6 Data Flow

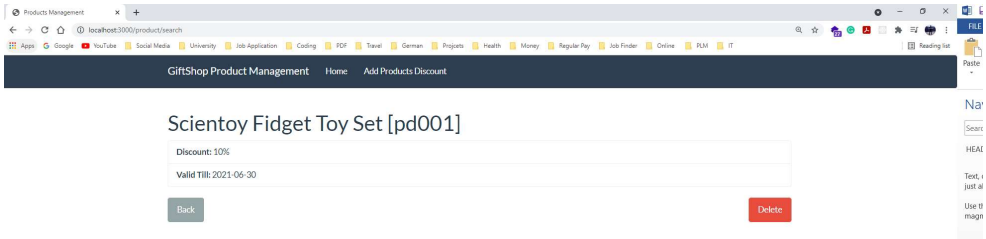
❖ Type code in Gitbash. (Make sure that Redis Server and Redis CLI is opened before type this code) - **npm start and will indicate to run code by localhost 3000**

```
MINGW64:/c/Users/Administrator/Desktop/RedisSearch

Administrator@DESKTOP-2RI2FNP MINGW64 ~/Desktop/RedisSearch (master)
$ npm start

> redissearch@1.0.0 start C:\Users\Administrator\Desktop\RedisSearch
> node app

Server started on port 3000
Connected to Redis...
```

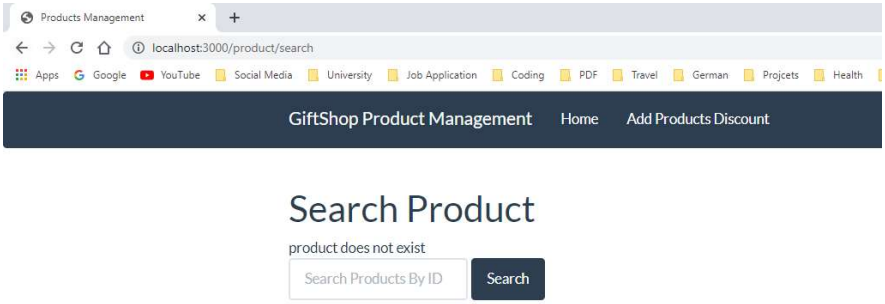
❖ Using Search Functionality (Type by product ID)



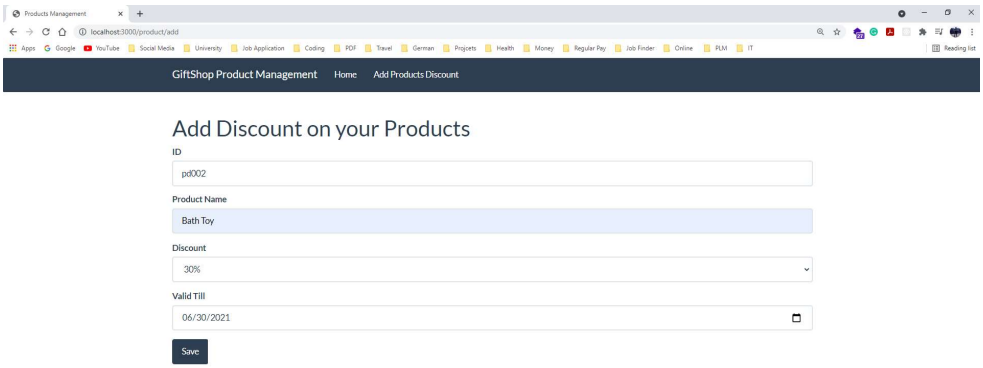
❖ Result (Showing details of Product Name, Discount and Valid date)



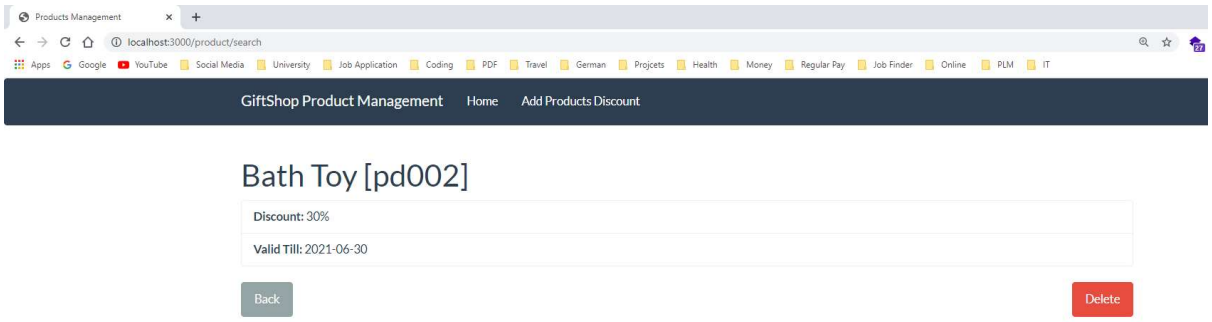
❖ Result (Product does not exist)



❖ **Now Using Add discount on products:**



❖ **After addition outcomes : (pd002 ID is available with details)**

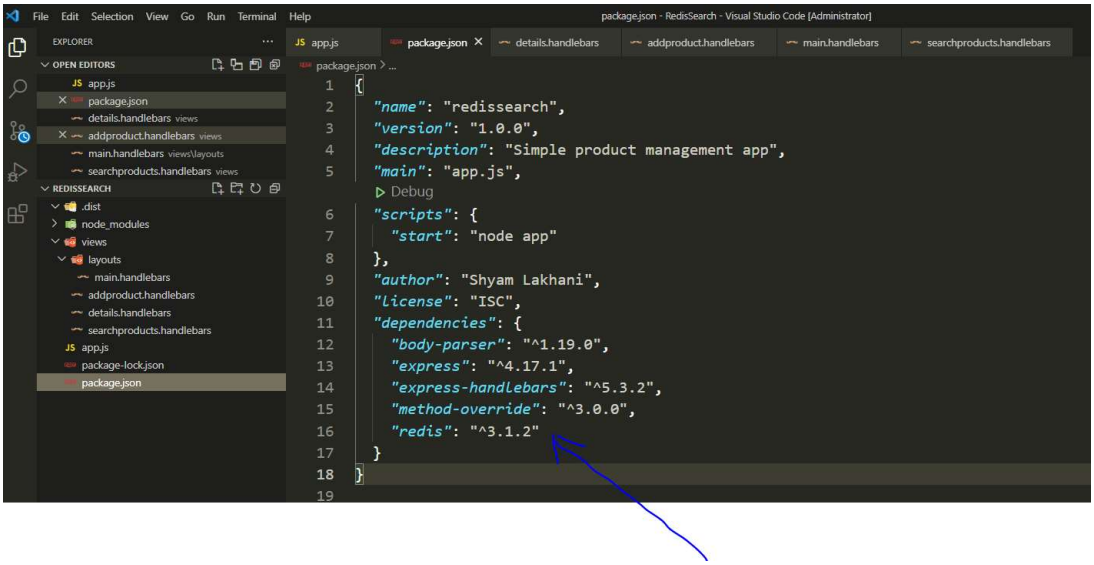


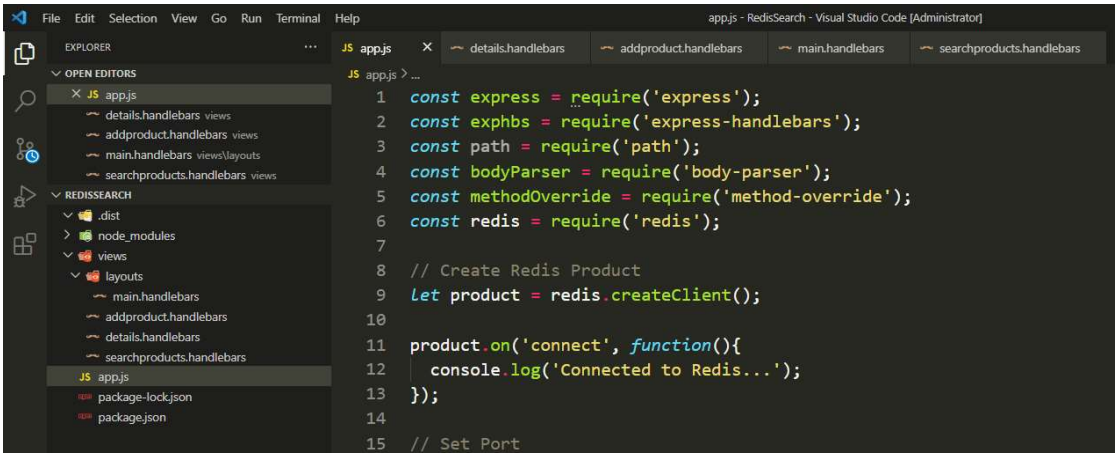
4.11.7 Databases - Redis

4.11.7.1 Databases used and why?

Redis is a very useful data service for tying microservices together and following the 12 factor app principles. For workloads focusing on rapidly changing ephemeral data sets where privilege control is not a concern (i.e. apps that you trust enough or less sensitive data) Redis is a strong choice for database.

4.11.7.2 Expressions used:



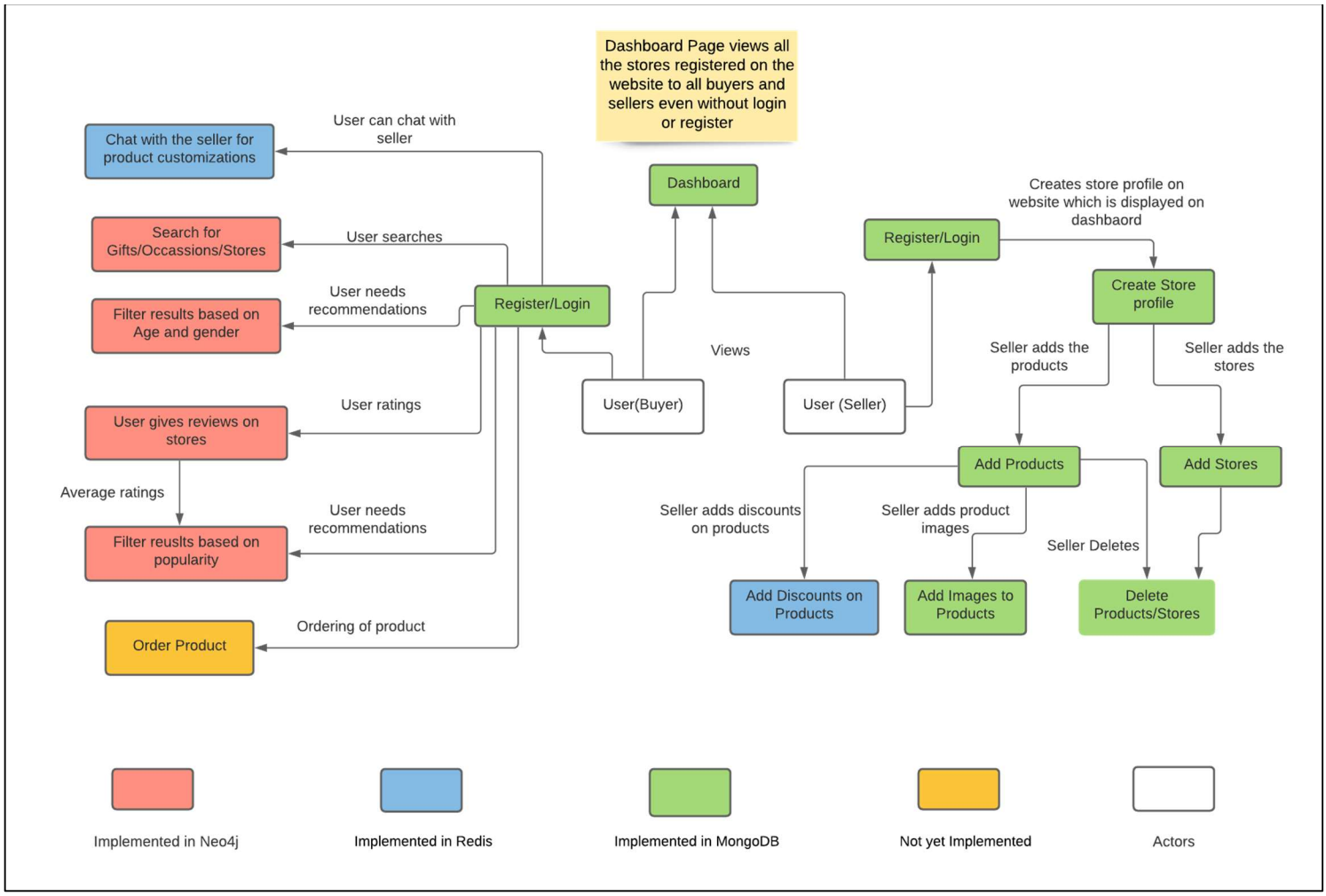


4.11.8 Outcome (what did you learn?)

This powerful database is perfect for high performance jobs such as caching. Redis is a fast database for many different functions including as a cache or a message broker. I learned everything through Redis tutorial, which is the best place to progress from a newbie to an advanced user of Redis, the basic fundamentals of Redis such as the different data structures, various clients that work with Redis, different key-value pair commands (scan, config, commands, and client), how to persist data to disks and even the different methods of persisting data. After that, I build a functional working task how to actually work with Redis in a real-world example. I built a task manager using NodeJS and Redis. I also learnt how to incorporate Twitter Bootstrap for designing the manager.

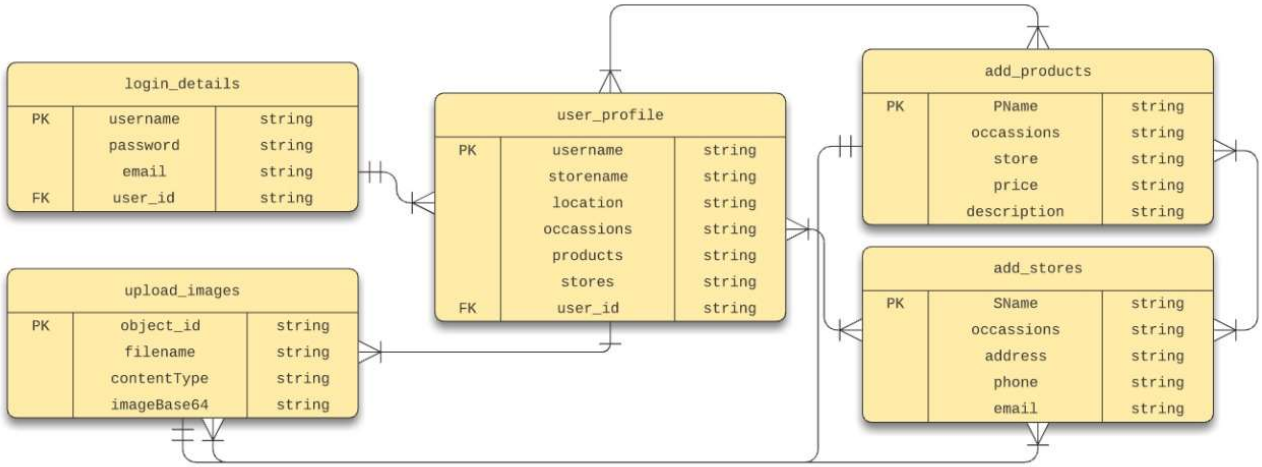
5. Databases

5.1 Overall Structure:

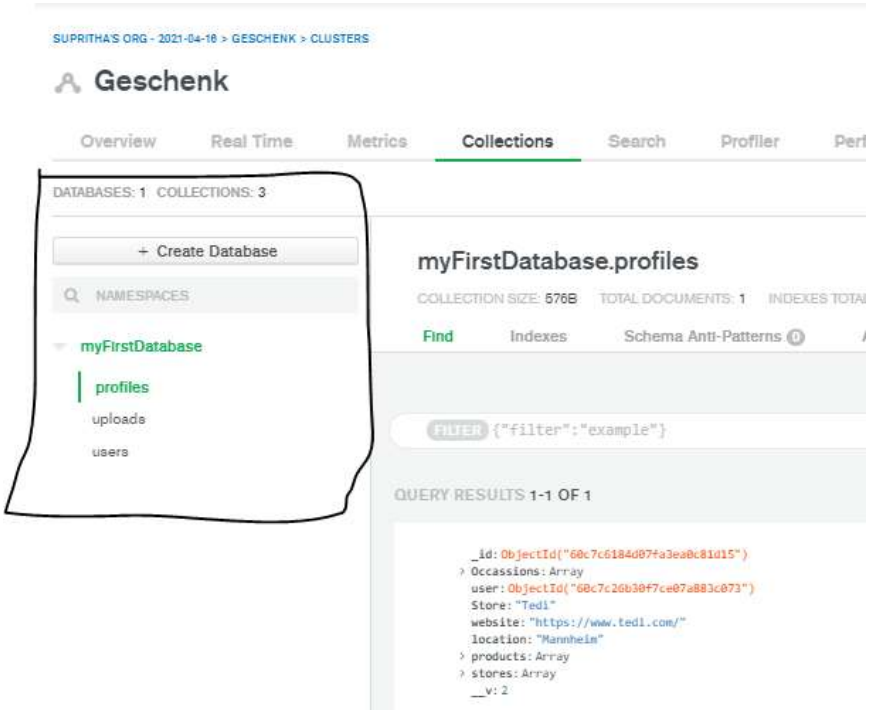


5.2 Data Model Overview all Databases

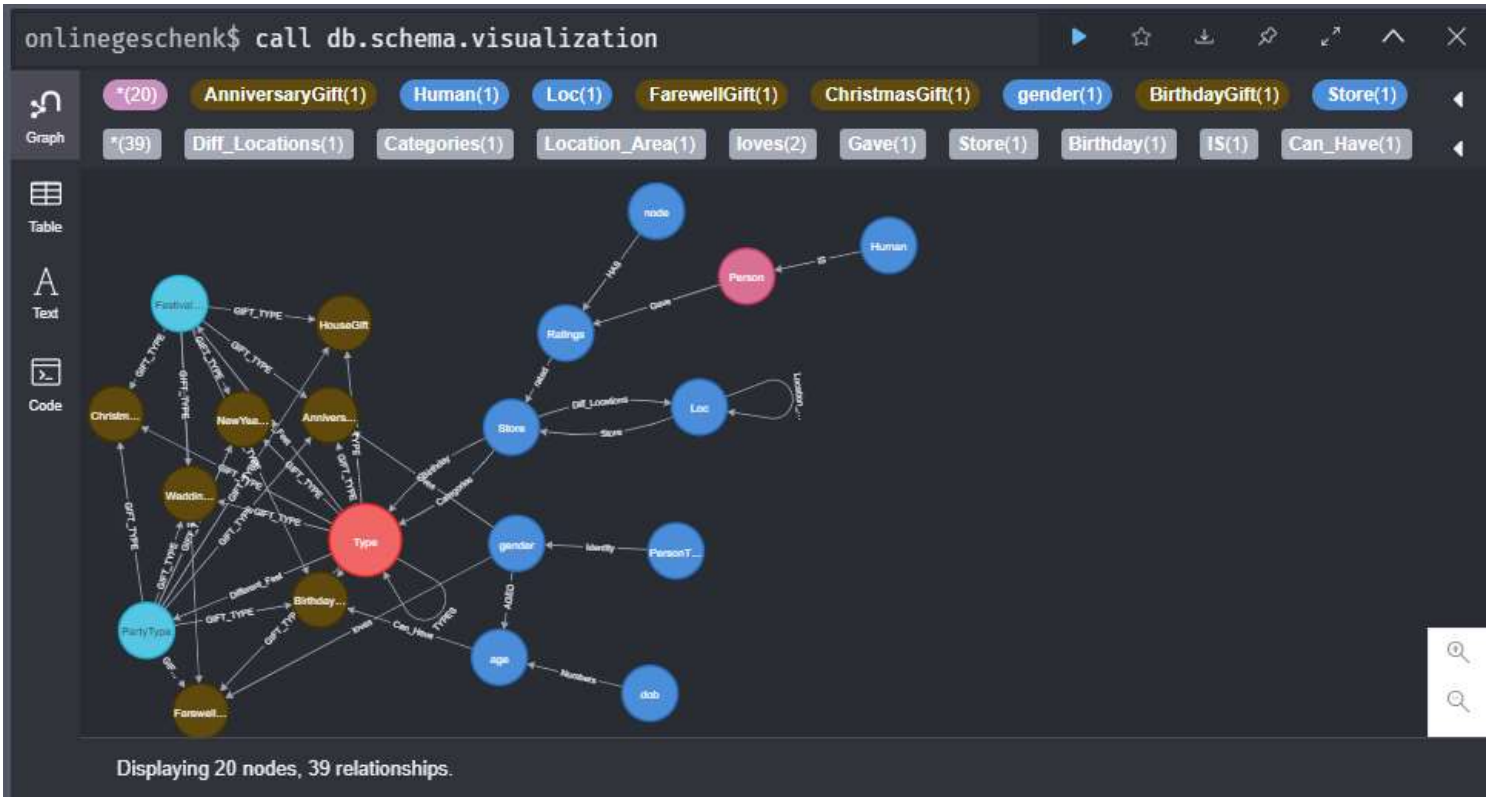
5.2.1 MongoDB (UML)



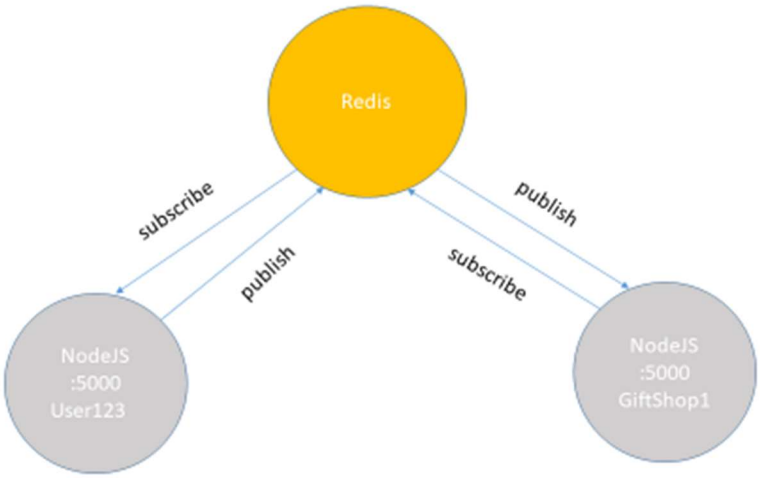
- Collections in MongoDB:



5.2.2 Neo4j (Image)



5.2.3 Redis key Value (UML)



5.3 Used Expressions:

5.3.1 MongoDB

Creating MongoDB Models:

1. User Model schema
2. Profile Model Schema
3. Product Model Schema
4. Store Model Schema
5. Image Upload Model Schema

```
const mongoose = require("mongoose");
const UserSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },

  email: {
    type: String,
    required: true,
    unique: true,
  },

  password: {
    type: String,
    required: true,
  },
  date: {
    type: Date,
    default: Date.now,
  },
});

module.exports = User = mongoose.model("user", UserSchema);
```

```
const mongoose = require("mongoose");
const ProfileSchema = new mongoose.Schema({
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "user",
  },
  Store: {type: String,},
  location: {type: String,},
  website: {type: String,},
  Occassions: {type: [String],},

  products: [{PName: {type: String,},
    Occassions: {type: String,},
    Store: {type: String,},
    Price: {type: String,},
    description: {type: String,},},],

  stores: [{SName: {type: String,},
    Occassions: {type: String,},
    Address: {type: String,},
    Phone: {type: String,default: false,},
    Email: {type: String,},},],
});

module.exports = Profile = mongoose.model("profile", ProfileSchema);
```

5.3.2 Neo4j

- **Creating Nodes:**

create (n:Type {name: 'Birthday'})

- **Creating relationships:**

MATCH (a:Type), (b:Type)
WHERE a.name = 'Occassions' AND b.name = 'Birthday'
CREATE (a)-[r:TYPES]->(b) RETURN type(r)

- **Deleting nodes:**

Match (n) WHERE id(n)=3 DELETE n

- **Deleting relationships:**

MATCH (:Loc)-[r:Gift_Store]-(:LocationStore) DELETE r

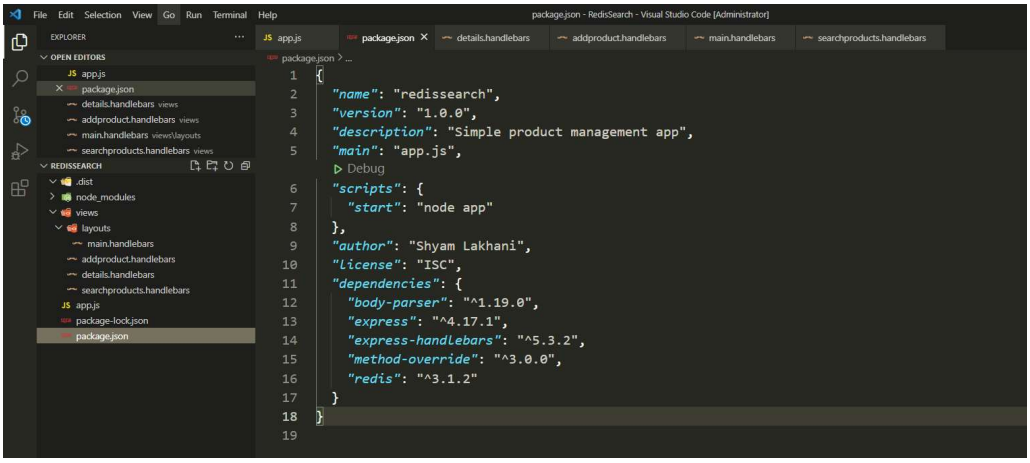
- **Updating properties:**

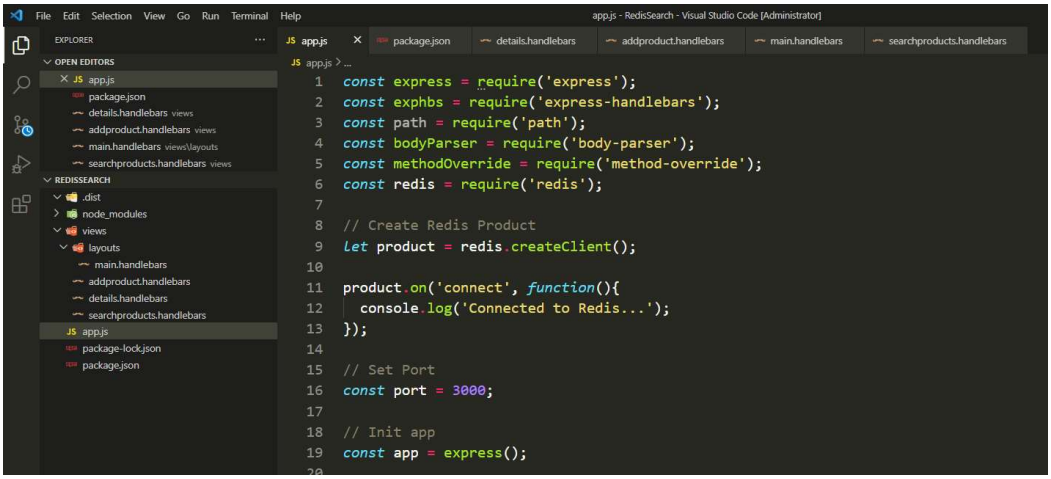
MATCH (p {name: 'Saturn'}) SET p = {name: 'Saturn', Status: 'Open'}
RETURN p.name, p.Status

- **Returning results:**

MATCH (n:Type)<-[:Gifts_for]-(a:Store)
RETURN n.name AS Gift_Categories, collect(a.name) AS Gift_Store

5.3.3 Redis




File Edit Selection View Go Run Terminal Help
app.js - RedisSearch - Visual Studio Code [Administrator]
EXPLORER
OPEN EDITORS
app.js
package.json
details.handlebars views
addproduct.handlebars views
main.handlebars views\layouts
searchproducts.handlebars views
REDISSEARCH
.dist
node_modules
views
layouts
main.handlebars
addproduct.handlebars
details.handlebars
searchproducts.handlebars
app.js
package-lock.json
package.json
app.js
package.json
details.handlebars
addproduct.handlebars
main.handlebars
searchproducts.handlebars
app.js > ...
const express = require('express');
const exphbs = require('express-handlebars');
const path = require('path');
const bodyParser = require('body-parser');
const methodOverride = require('method-override');
const redis = require('redis');
// Create Redis Product
let product = redis.createClient();
product.on('connect', function(){
console.log('Connected to Redis...');
});
// Set Port
const port = 3000;
// Init app
const app = express();

6. Application

6.1 Language Used (and why?)

- **HTML and CSS:** HTML is used for the structure of the web pages designed and CSS for design, look and layout.
- **ReactJS:** For rendering reusable UI components, for the view on mobile and web applications, fast and simple.
- **JavaScript:** Used as the main language for the functionalities used in MongoDB, it's the best fit with reactjs, client-side, robust and responsive web applications.
- **NodeJS:** Goes well with JavaScript and for backend implementation in MongoDB and Redis.
- **Express:** Middleware for the backend server and frontend
- **Mongoose:** Creating schema models

6.2 GitHub Path –

https://github.com/SuprithaPrakash/OnlineGeschenk

6.3 Methods/Functions:

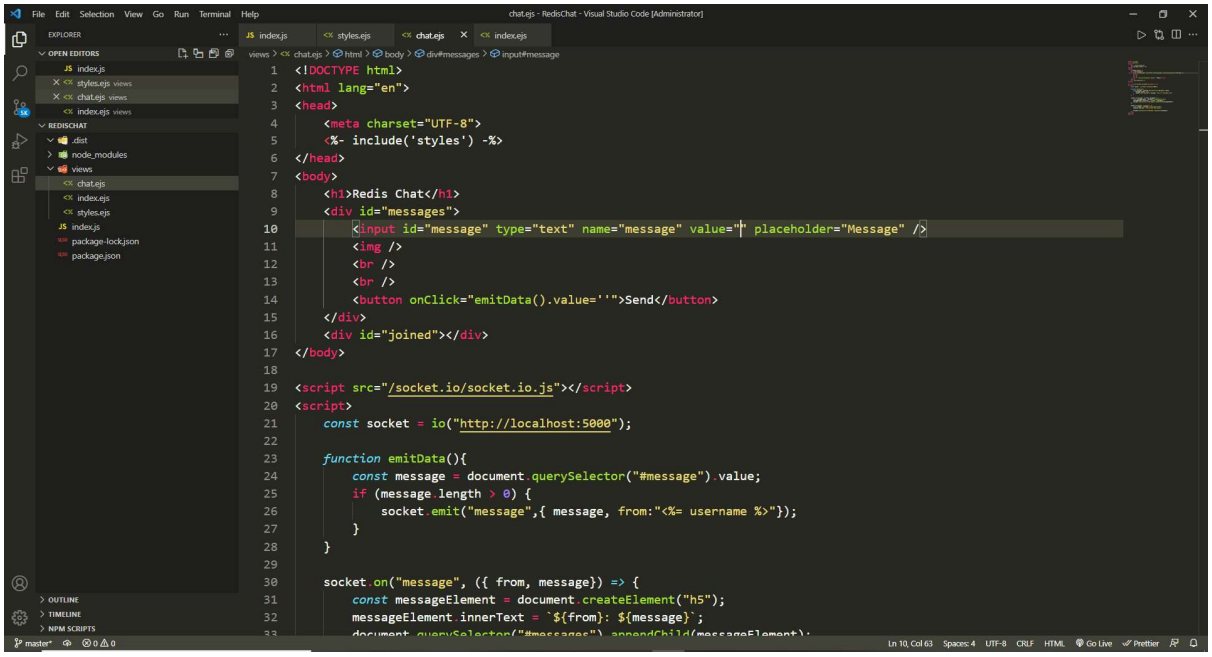
1. Upload image in MongoDB:

```
const UploadModel=require('../model/schema');
const fs=require('fs');

exports.home=async(req,res) =>{
    const all_images=await UploadModel.find()
    res.render('main',{images:all_images});
}

exports.uploads=(req,res,next)=> {
    const files=req.files;

    if(!files){
        const error=new Error('Please choose files');
        error.httpStatusCode=400;
        return next(error)
    }

//convert images into base64 encoding
let imgArray=files.map((file)=>{
    let img=fs.readFileSync(file.path)

    return encode_image= img.toString('base64')
})

let result=imgArray.map((src,index)=>{
    //create object to store datain the collection
    let finalImg={
        filename:files[index].originalname,
```

2. Recommendations in Neo4j:

MATCH(t:Type)<-[:Birthday]-(n:Store)<-[:rated]-(a:Ratings)
RETURN n.name AS Gift_Store, avg(a.name) AS Ratings, t.name AS Gift_Occassions
Order By Ratings desc

3. Live Chat with seller in Redis:



7. API:

- **Post Method:** To upload image in MongoDB and to post data in the forms.
- **Get Method:** To get the images, user profiles and also to get the user data on the profile.
- **Put Method:** To update the store or product or store profile data in MongoDB.
- **Delete Method:** To delete the product, store or user using the ID.

8. Evaluation

Outlook:

- This application Online-geschenk uses three different databases that is MongoDB, Neo4j and Redis to develop the different user stories. Each and every user story selected is appropriate for each and every character of the databased used.
- Identified Usecases for the user stories are implemented fully.
- Four team members in the project contributed equally for the three different databases respectively.

Lessons Learned:

- Learnt about NOSQL databases, Redis, MongoDB and Neo4j from the documentation provided by the professor.
- Learnt querying in all three different databases and learnt the difference and the importance of choosing the database for the appropriate user story tasks.

Extensions:

- Combining the front end and expanding the project.
- Including option for the users to buy on the website itself.
- Likes and dislikes for the products and stores.
- Live delivery tracking of the products ordered.

9. References

- https://neo4j.com/docs/cypher-manual/current/syntax/
- https://neo4j.com/docs/pdf/neo4j-getting-started-4.2.pdf
- https://mongoosejs.com/docs/
- https://www.w3schools.com/js/default.asp
- https://www.geeksforgeeks.org/upload-and-retrieve-image-on-mongodb-using-mongoose/
- https://stackoverflow.com/questions/62943150/uploading-images-with-mongodb-express-node-and-react
- https://docs.atlas.mongodb.com/getting-started/
- https://www.w3schools.com/nodejs/nodejs_mongodb_create_db.asp
- https://www.youtube.com/watch?v=7CqJlxBYj-M
- https://www.objectrocket.com/blog/mongodb/top-use-cases-for-mongodb/
- https://www.mongodb.com/use-cases
- https://www.w3schools.com/nodejs/nodejs_mongodb_create_db.asp
- https://youtu.be/ZS_kXvOeQ5Y
- https://youtu.be/l8aGNhOD91k
- https://youtu.be/4rhKKFbbYT4
- https://youtu.be/3GHZd0zv170
- https://youtu.be/ktjafK4SgWM
- https://youtu.be/3f5Q9wDePzY
- https://www.youtube.com/watch?v=9S-mphgE5fA
- https://www.youtube.com/watch?v=kNIhN5g1A_A
- https://redis.io/commands/KEYS