Dương Ngọc Linh Đan – 2374802010091

Result:

9 Practical Exercises

9.1 Getting Started with MongoDB

```
>_ mongosh: localhost:27017 +

>_MONGOSH

> use Lab7
< switched to db Lab7
Lab7> db["Restaurant"].find()
```

- The find() function is used to retrieve all documents in a collection.

```
>_ mongosh: localhost:27017 +

>_MONGOSH

> use Lab7
< switched to db Lab7
> db["Restaurant"].find()
< {
    _id: ObjectId('6782122afa4945d74d0aaaef'),
    address: {
      building: '1007',
      coord: [
        -73.856077,
        40.848447
      ],
      street: 'Morris Park Ave',
      zipcode: '10462'
    },
    borough: 'Bronx',
    cuisine: 'Bakery',
    grades: [
      {
        date: 2014-03-03T00:00:00.000Z,
        grade: 'A',
        score: 2
      },
      {
        date: 2013-09-11T00:00:00.000Z,
        grade: 'A',
        score: 6
      },
      {
```

5. Display the fields: restaurant_id, name, borough, and cuisine.

```
mongosh: localhost:27017
>_MONGOSH
}
Type "it" for more
> db["Restaurant"].find({}, {restaurant_id: 1, name: 1, borough: 1, cuisine: 1})
< {
  _id: ObjectId('6782122afa4945d74d0aaaef'),
  borough: 'Bronx',
  cuisine: 'Bakery',
  name: 'Morris Park Bake Shop',
  restaurant_id: '30075445'
}
{
  _id: ObjectId('6782122afa4945d74d0aaaf0'),
  borough: 'Brooklyn',
  cuisine: 'Hamburgers',
  name: "Wendy'S",
  restaurant_id: '30112340'
}
{
  _id: ObjectId('6782122afa4945d74d0aaaf1'),
  borough: 'Manhattan',
  cuisine: 'Irish',
  name: 'Dj Reynolds Pub And Restaurant',
  restaurant_id: '30191841'
}
{
  _id: ObjectId('6782122afa4945d74d0aaaf2'),
  borough: 'Brooklyn',
  cuisine: 'American ',
```

6. Display the same fields but exclude the _id field.

```
mongosh: localhost:27017
>_MONGOSH
Type "it" for more
> db["Restaurant"].find({}, {restaurant_id: 1, name: 1, borough: 1, cuisine: 1, _id: 0})
< {
    borough: 'Bronx',
    cuisine: 'Bakery',
    name: 'Morris Park Bake Shop',
    restaurant_id: '30075445'
  }
  {
    borough: 'Brooklyn',
    cuisine: 'Hamburgers',
    name: "Wendy'S",
    restaurant_id: '30112340'
  }
  {
    borough: 'Manhattan',
    cuisine: 'Irish',
    name: 'Dj Reynolds Pub And Restaurant',
    restaurant_id: '30191841'
  }
  {
    borough: 'Brooklyn',
    cuisine: 'American ',
    name: 'Riviera Caterer',
    restaurant_id: '40356018'
  }
  {
    borough: 'Queens',
```

7. Display the fields name and address of all restaurants.

```
mongosh: localhost:27017
>_MONGOSH
}
Type "it" for more
> db["Restaurant"].find({}, {name: 1, address: 1, _id: 0})
< {
  address: {
    building: '1007',
    coord: [
      -73.856077,
      40.848447
    ],
    street: 'Morris Park Ave',
    zipcode: '10462'
  },
  name: 'Morris Park Bake Shop'
}
{
  address: {
    building: '469',
    coord: [
      -73.961704,
      40.662942
    ],
    street: 'Flatbush Avenue',
    zipcode: '11225'
  },
  name: "Wendy'S"
}
{
```

8. Show all restaurants where borough is **Bronx**.

```
mongosh: localhost:27017
>_MONGOSH
Type "it" for more
> db["Restaurant"].find({borough: "Bronx"})
< {
    _id: ObjectId('6782122afa4945d74d0aaaef'),
    address: {
      building: '1007',
      coord: [
        -73.856077,
        40.848447
      ],
      street: 'Morris Park Ave',
      zipcode: '10462'
    },
    borough: 'Bronx',
    cuisine: 'Bakery',
    grades: [
      {
        date: 2014-03-03T00:00:00.000Z,
        grade: 'A',
        score: 2
      },
      {
        date: 2013-09-11T00:00:00.000Z,
        grade: 'A',
        score: 6
      },
      {
        date: 2013-01-24T00:00:00.000Z,
```

9. how all restaurants where cuisine is **Bakery**.

```
mongosh: localhost:27017
>_MONGOSH
Type "it" for more
> db["Restaurant"].find({cuisine: "Bakery"})
< {
    _id: ObjectId('6782122afa4945d74d0aaaef'),
    address: {
      building: '1007',
      coord: [
        -73.856077,
        40.848447
      ],
      street: 'Morris Park Ave',
      zipcode: '10462'
    },
    borough: 'Bronx',
    cuisine: 'Bakery',
    grades: [
      {
        date: 2014-03-03T00:00:00.000Z,
        grade: 'A',
        score: 2
      },
      {
        date: 2013-09-11T00:00:00.000Z,
        grade: 'A',
        score: 6
      },
      {
        date: 2013-01-24T00:00:00.000Z,
```

10. Show the first 5 restaurants located in **Bronx**.

- Use limit() to limit the number of returned documents.

```
mongosh: localhost:27017
>_MONGOSH
Type "it" for more
> db["Restaurant"].find({borough: "Bronx"}).limit(5)
< {
  _id: ObjectId('6782122afa4945d74d0aaaef'),
  address: {
    building: '1007',
    coord: [
      -73.856077,
      40.848447
    ],
    street: 'Morris Park Ave',
    zipcode: '10462'
  },
  borough: 'Bronx',
  cuisine: 'Bakery',
  grades: [
    {
      date: 2014-03-03T00:00:00.000Z,
      grade: 'A',
      score: 2
    },
    {
      date: 2013-09-11T00:00:00.000Z,
      grade: 'A',
      score: 6
    },
    {
      date: 2013-01-24T00:00:00.000Z,
```

11. Display the next 5 restaurants in **Bronx**, skipping the first 5.

- Use skip() to skip a number of documents.

mongosh: localhost:27017

>_MONGOSH

```
> db["Restaurant"].find({borough: "Bronx"}).skip(5).limit(5)
< {
    _id: ObjectId('6782122afa4945d74d0aab2c'),
    address: {
      building: '658',
      coord: [
        -73.81363999999999,
        40.82941100000001
      ],
      street: 'Clarence Ave',
      zipcode: '10465'
    },
    borough: 'Bronx',
    cuisine: 'American ',
    grades: [
      {
        date: 2014-06-21T00:00:00.000Z,
        grade: 'A',
        score: 5
      },
      {
        date: 2012-07-11T00:00:00.000Z,
        grade: 'A',
        score: 10
      }
    ],
    name: 'Manhem Club',
    restaurant_id: '40364363'
```

12. Display 10 restaurants located in **Manhattan** that have a grade of **A**.

```
mongosh: localhost:27017
>_MONGOSH
> db["Restaurant"].find({
    borough: "Manhattan",
    "grades.grade": "A"
  }).limit(10)
< {
    _id: ObjectId('6782122afa4945d74d0aaaf1'),
    address: {
      building: '351',
      coord: [
        -73.98513559999999,
        40.7676919
      ],
      street: 'West   57 Street',
      zipcode: '10019'
    },
    borough: 'Manhattan',
    cuisine: 'Irish',
    grades: [
      {
        date: 2014-09-06T00:00:00.000Z,
        grade: 'A',
        score: 2
      },
      {
        date: 2013-07-22T00:00:00.000Z,
        grade: 'A',
        score: 11
      },
```

13. Count the number of restaurants grouped by borough

- Use the count() or countDocuments() function.
- The count() function is used to count the number of documents in a collection.

```
mongosh: localhost:27017
>_MONGOSH
> db["Restaurant"].aggregate([
    {
      $group: {
        _id: "$borough",
        total: { $sum: 1 }
      }
    }
  ])
< {
    _id: 'Queens',
    total: 738
  }
  {
    _id: 'Brooklyn',
    total: 684
  }
  {
    _id: 'Bronx',
    total: 310
  }
  {
    _id: 'Manhattan',
    total: 1883
  }
  {
    _id: 'Staten Island',
    total: 158
  }
```

14. Count how many restaurants are in **Queens** (presented using two counting methods).

```
> db["Restaurant"].find({borough: "Queens"}).count()
< 738
Lab7 >
```

15. Display restaurants with at least one score greater than 90.

• Use the find() function and comparison operators.

• Comparison operators:

– Greater than: $gt

– Greater than or equal to: $gte

– Less than: $lt

– Less than or equal to: $lte

mongosh: localhost:27017

>_MONGOSH

```
> db["Restaurant"].find({
    "grades.score": { $gt: 90 }
  })
< {
    _id: ObjectId('6782122afa4945d74d0aac4d'),
    address: {
      building: '65',
      coord: [
        -73.9782725,
        40.7624022
      ],
      street: 'West   54 Street',
      zipcode: '10019'
    },
    borough: 'Manhattan',
    cuisine: 'American ',
    grades: [
      {
        date: 2014-08-22T00:00:00.000Z,
        grade: 'A',
        score: 11
      },
      {
        date: 2014-03-28T00:00:00.000Z,
        grade: 'C',
        score: 131
      },
      {
```

16. Display restaurants with scores greater than 80 and less than 100.

```
mongosh: localhost:27017
>_MONGOSH
> db["Restaurant"].find({
    "grades.score": { $gt: 80, $lt: 100 }
  })
< {
    _id: ObjectId('6782122afa4945d74d0aac4d'),
    address: {
      building: '65',
      coord: [
        -73.9782725,
        40.7624022
      ],
      street: 'West   54 Street',
      zipcode: '10019'
    },
    borough: 'Manhattan',
    cuisine: 'American ',
    grades: [
      {
        date: 2014-08-22T00:00:00.000Z,
        grade: 'A',
        score: 11
      },
      {
        date: 2014-03-28T00:00:00.000Z,
        grade: 'C',
        score: 131
      },
      {
```

17. Display restaurants whose latitude is less than -95.754168.

```
mongosh: localhost:27017
>_MONGOSH
> db["Restaurant"].find({
    "address.coord.0": { $lt: -95.754168 }
  })
< {
    _id: ObjectId('6782122afa4945d74d0ab137'),
    address: {
      building: '3707',
      coord: [
        -101.8945214,
        33.5197474
      ],
      street: '82 Street',
      zipcode: '11372'
    },
    borough: 'Queens',
    cuisine: 'American ',
    grades: [
      {
        date: 2014-06-04T00:00:00.000Z,
        grade: 'A',
        score: 12
      },
      {
        date: 2013-11-07T00:00:00.000Z,
        grade: 'B',
        score: 19
      },
      {
```

18. Display restaurants that have at least one score less than 10.

```
mongosh: localhost:27017
>_MONGOSH
> db["Restaurant"].find({
    "grades.score": { $lt: 10 }
  })
< {
    _id: ObjectId('6782122afa4945d74d0aaaef'),
    address: {
      building: '1007',
      coord: [
        -73.856077,
        40.848447
      ],
      street: 'Morris Park Ave',
      zipcode: '10462'
    },
    borough: 'Bronx',
    cuisine: 'Bakery',
    grades: [
      {
        date: 2014-03-03T00:00:00.000Z,
        grade: 'A',
        score: 2
      },
      {
        date: 2013-09-11T00:00:00.000Z,
        grade: 'A',
        score: 6
      },
      {
```

19. Display restaurants whose cuisine is either **Hamburgers** or **Pizza**.

    - The $in and $nin operators are used to determine whether a value exists within (or does not exist within) a specified array of values

```
mongosh: localhost:27017
>_MONGOSH
> db["Restaurant"].find({
    cuisine: { $in: ["Hamburgers", "Pizza"] }
  })
< {
    _id: ObjectId('6782122afa4945d74d0aaaf0'),
    address: {
      building: '469',
      coord: [
        -73.961704,
        40.662942
      ],
      street: 'Flatbush Avenue',
      zipcode: '11225'
    },
    borough: 'Brooklyn',
    cuisine: 'Hamburgers',
    grades: [
      {
        date: 2014-12-30T00:00:00.000Z,
        grade: 'A',
        score: 8
      },
      {
        date: 2014-07-01T00:00:00.000Z,
        grade: 'B',
        score: 23
      },
      {
```

20. Count how many restaurants have cuisine other than **American** and borough is **Queens**.

- The $ne (not equal) operator returns documents in which the value of a specified field is not equal to the provided value.

```
> db["Restaurant"].countDocuments({
    borough: "Queens",
    cuisine: { $ne: "American " }
  })
< 544
Lab7>
```

21. Display fields restaurant_id, name, borough, cuisine for restaurants where name starts with **"M"** and have a grades.score greater than **100**.

- Use find() with regular expressions and comparison operators.
- Regular expressions are used to filter string values.

```
> db["Restaurant"].countDocuments({
    borough: "Queens",
    cuisine: { $ne: "American " }
  })
< 544
> db["Restaurant"].find(
    {
      name: { $regex: /^M/ },
      "grades.score": { $gt: 100 }
    },
    {
      restaurant_id: 1,
      name: 1,
      borough: 1,
      cuisine: 1,
      _id: 0
    }
  )
< {
    borough: 'Manhattan',
    cuisine: 'American ',
    name: "Murals On 54/Randolphs'S",
    restaurant_id: '40372466'
  }
Lab7>
```

9.2 Building A Web Application With Flask & MongoDB(Local)

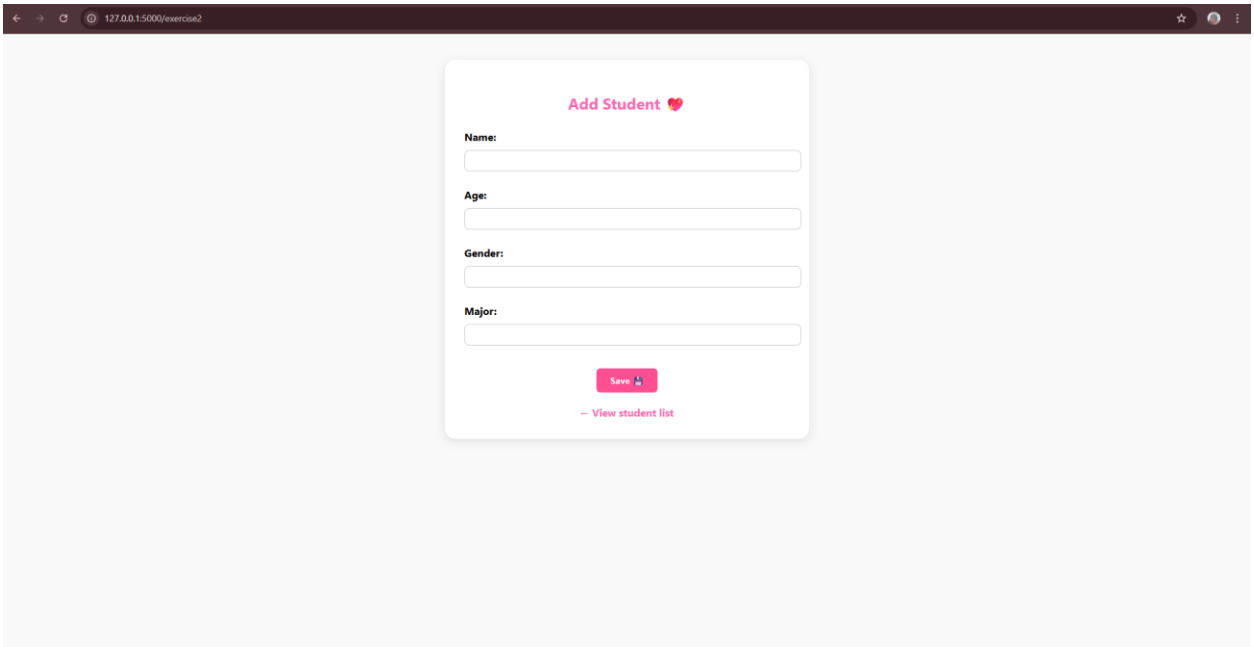
Exercise 1: Connecting Flask to MongoDB

- Set up the initial connection between Flask and MongoDB.
- Create the students_db database and the Chapter6_students collection.

Exercise 2: Add Student Information

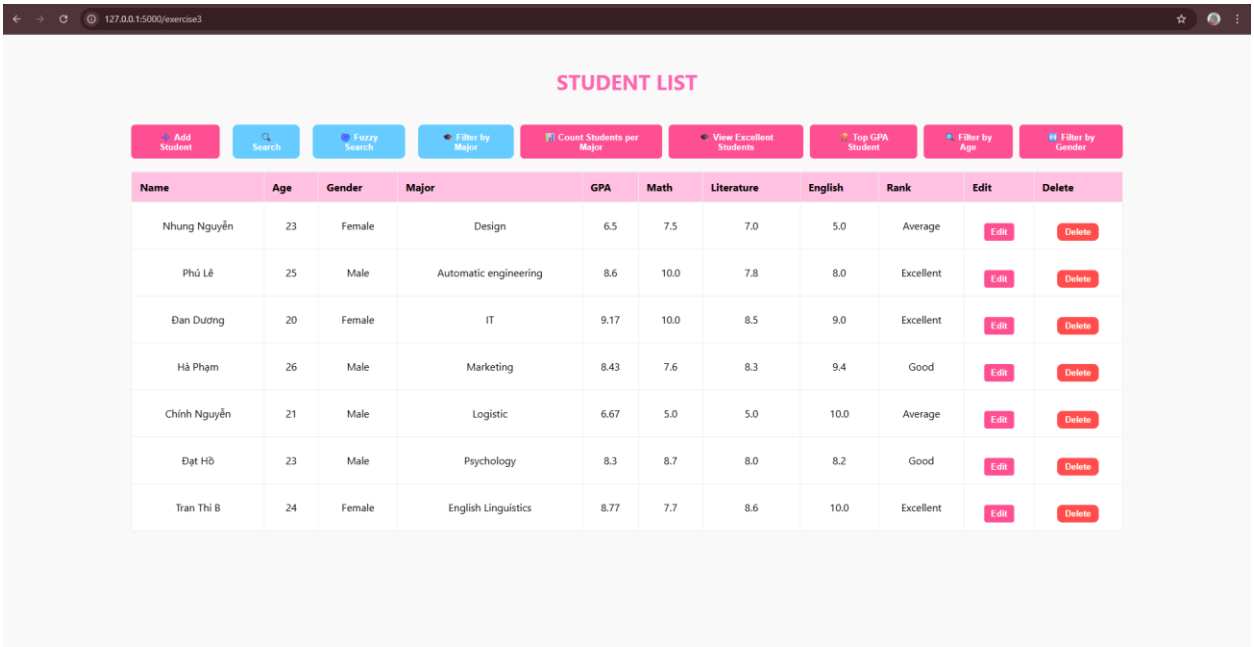
- Create a form to add student data and store it in MongoDB. Input name, age, gender, and major.

Note: Insert between 5 to 10 student documents, each with different information.



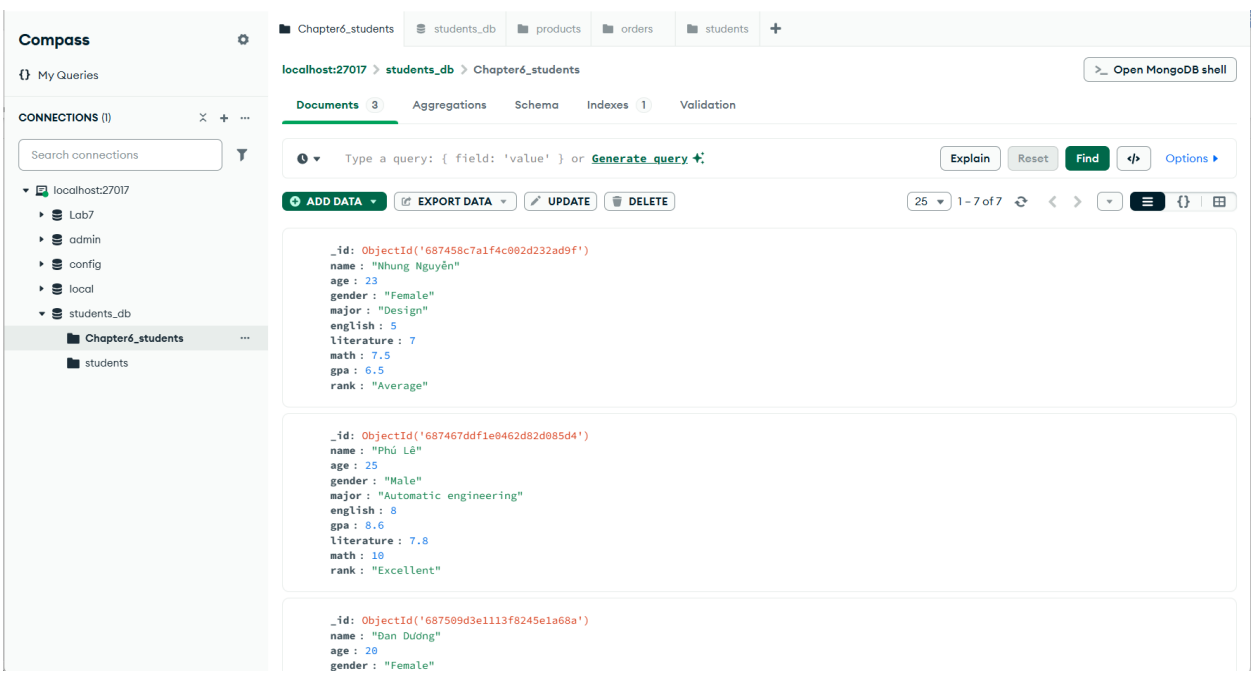
Exercise 3: Display Student List

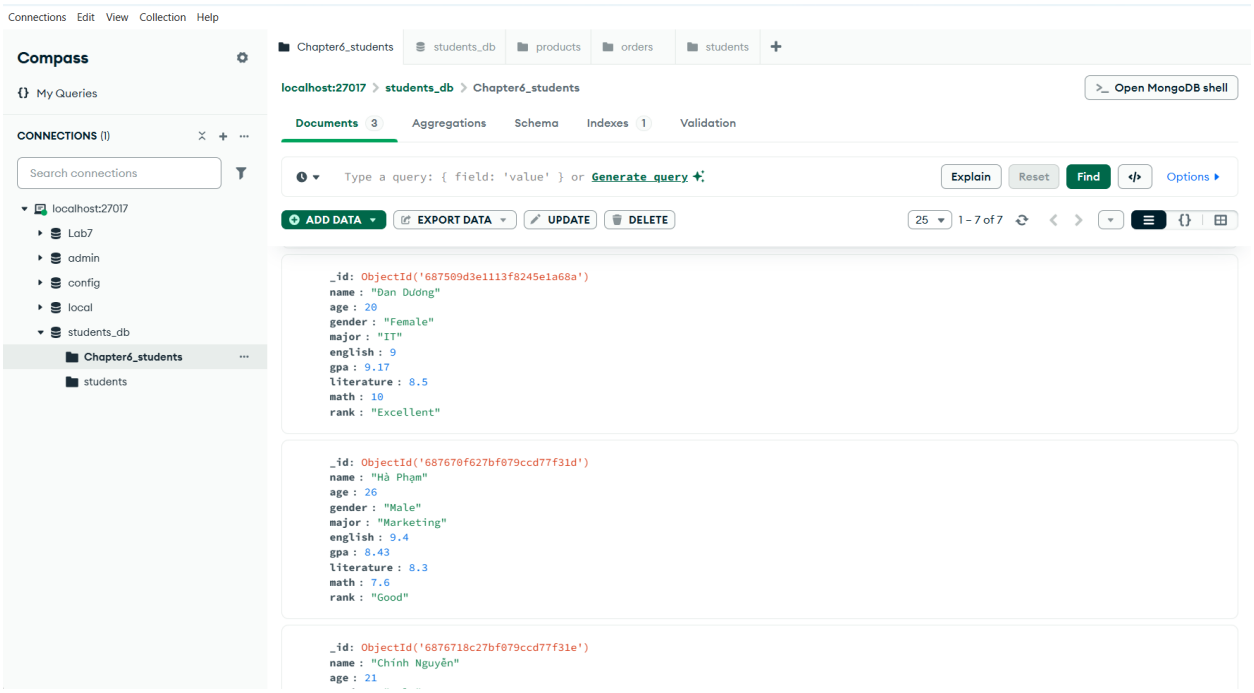
- Display all student records from MongoDB.



Name	Age	Gender	Major	GPA	Math	Literature	English	Rank	Edit	Delete
Nhung Nguyễn	23	Female	Design	6.5	7.5	7.0	5.0	Average	Edit	Delete
Phú Lê	25	Male	Automatic engineering	8.6	10.0	7.8	8.0	Excellent	Edit	Delete
Đan Dương	20	Female	IT	9.17	10.0	8.5	9.0	Excellent	Edit	Delete
Hà Phạm	26	Male	Marketing	8.43	7.6	8.3	9.4	Good	Edit	Delete
Chính Nguyễn	21	Male	Logistic	6.67	5.0	5.0	10.0	Average	Edit	Delete
Đạt Hồ	23	Male	Psychology	8.3	8.7	8.0	8.2	Good	Edit	Delete
Tran Thi B	24	Female	English Linguistics	8.77	7.7	8.6	10.0	Excellent	Edit	Delete

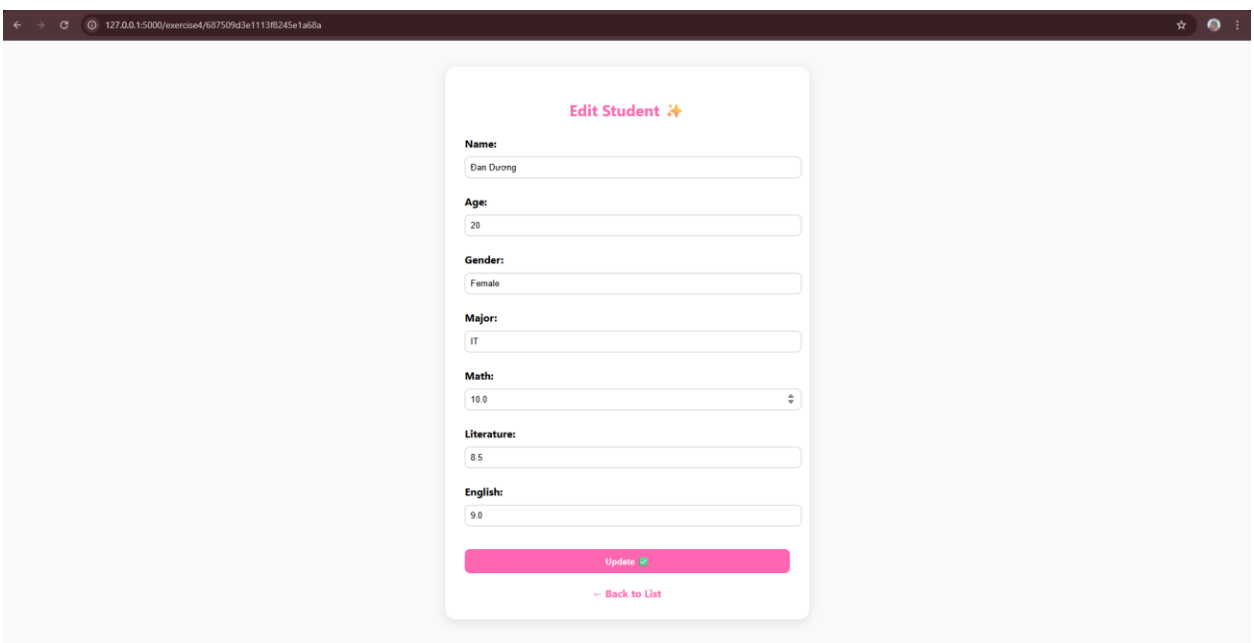
- MongoDB:





Exercise 4: Edit Student Information

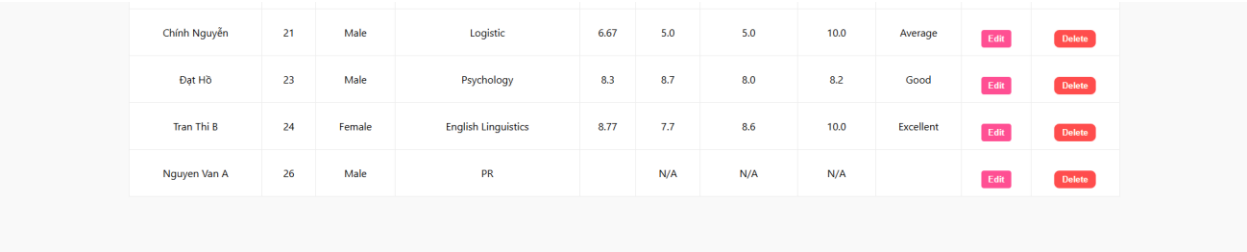
- Allow users to update student data.
- GET and POST methods, update_one().



Exercise 5: Delete Student

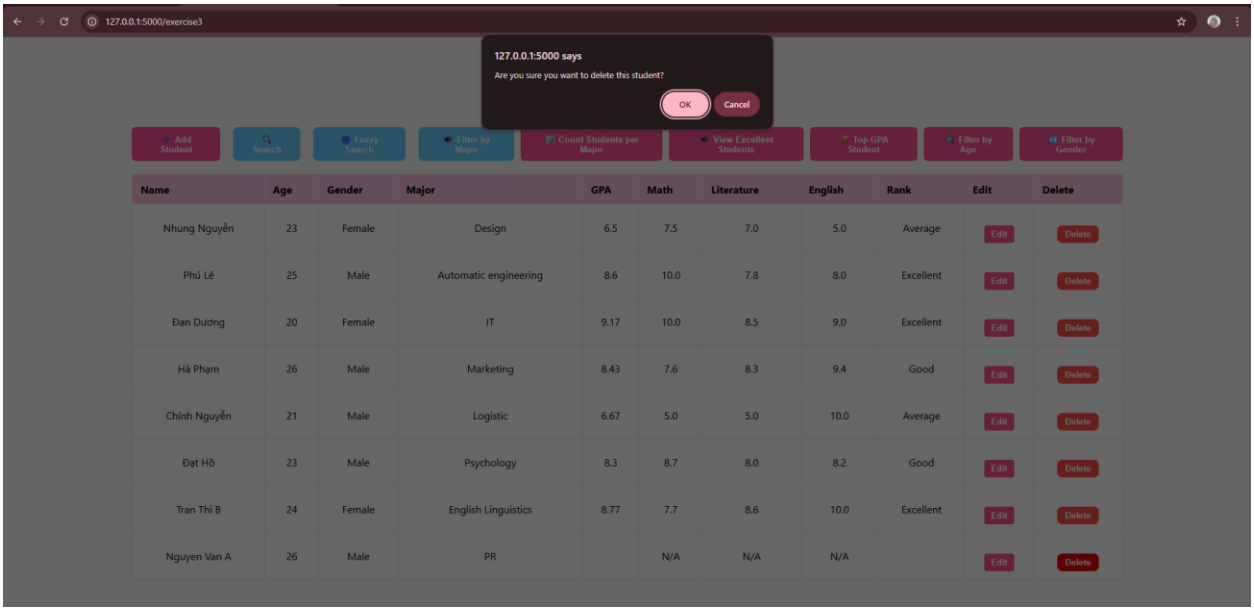
- Delete a student by ID. **Delete** button per row, confirm and remove.
- delete_one(), ObjectId.

- At first, there was a person named Nguyen Van A.

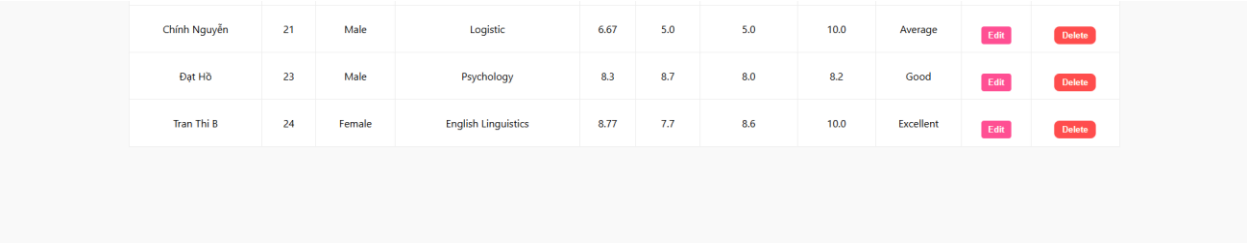


Chính Nguyễn	21	Male	Logistic	6.67	5.0	5.0	10.0	Average	Edit	Delete
Đạt Hồ	23	Male	Psychology	8.3	8.7	8.0	8.2	Good	Edit	Delete
Tran Thi B	24	Female	English Linguistics	8.77	7.7	8.6	10.0	Excellent	Edit	Delete
Nguyen Van A	26	Male	PR		N/A	N/A	N/A		Edit	Delete

- When I click Delete:



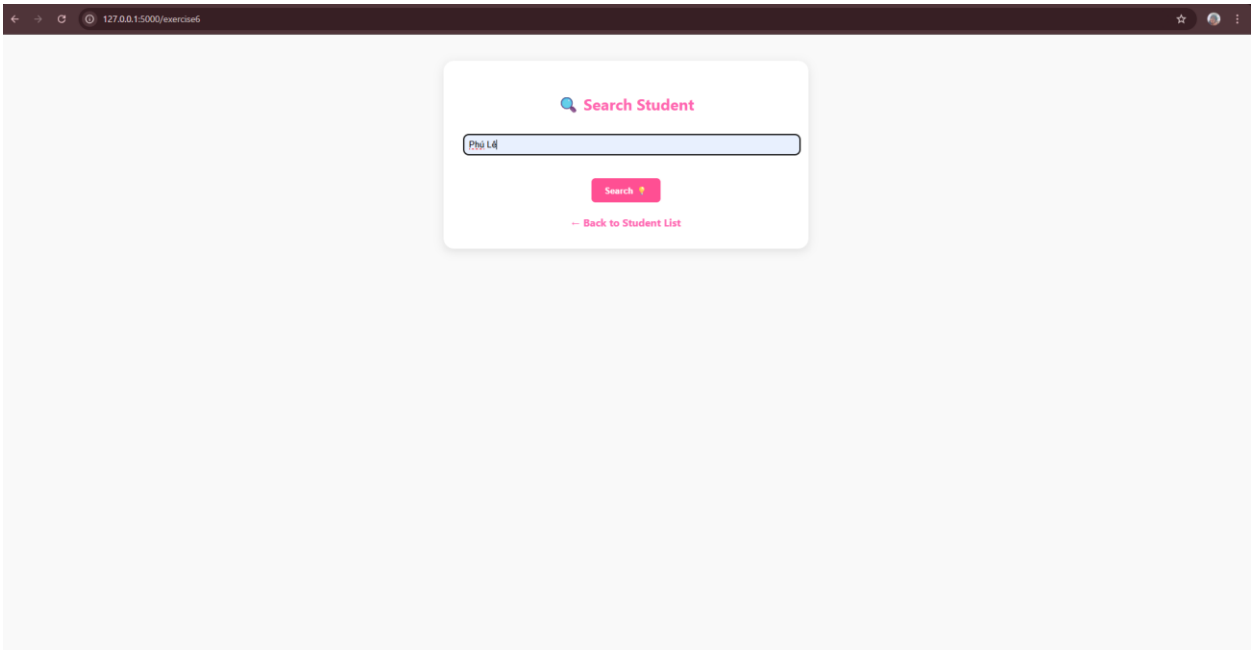
- After I deleted, Nguyen Van A has disappeared

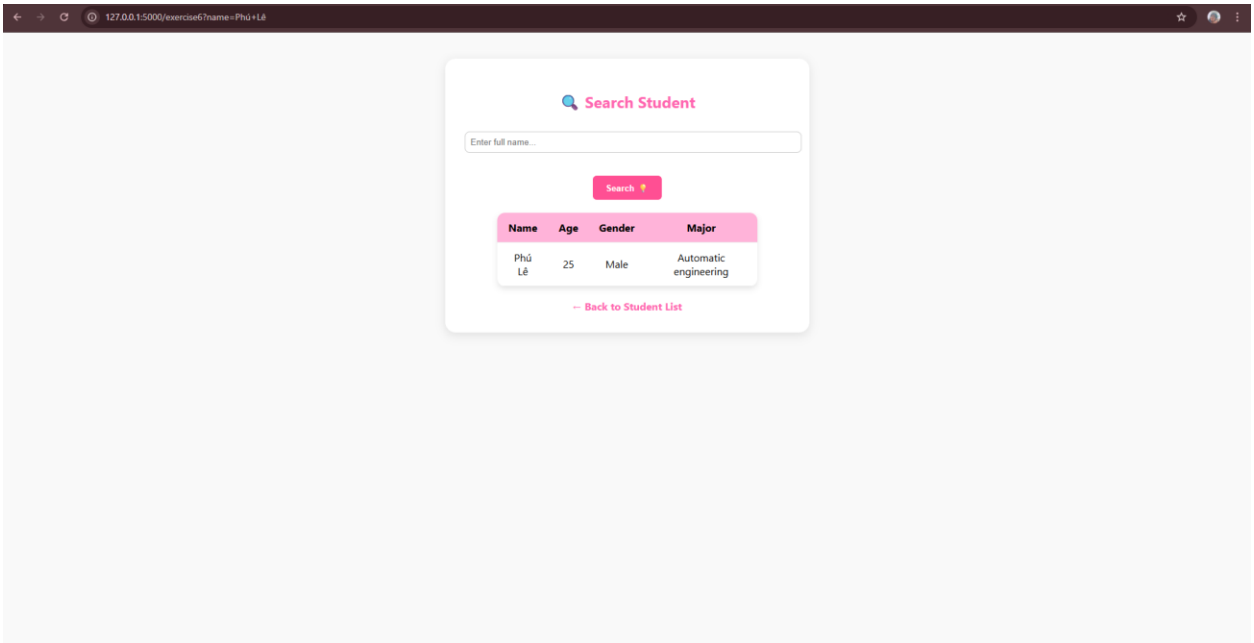


Chính Nguyễn	21	Male	Logistic	6.67	5.0	5.0	10.0	Average	Edit	Delete
Đạt Hồ	23	Male	Psychology	8.3	8.7	8.0	8.2	Good	Edit	Delete
Tran Thi B	24	Female	English Linguistics	8.77	7.7	8.6	10.0	Excellent	Edit	Delete

Exercise 6: Search Student by Exact Name

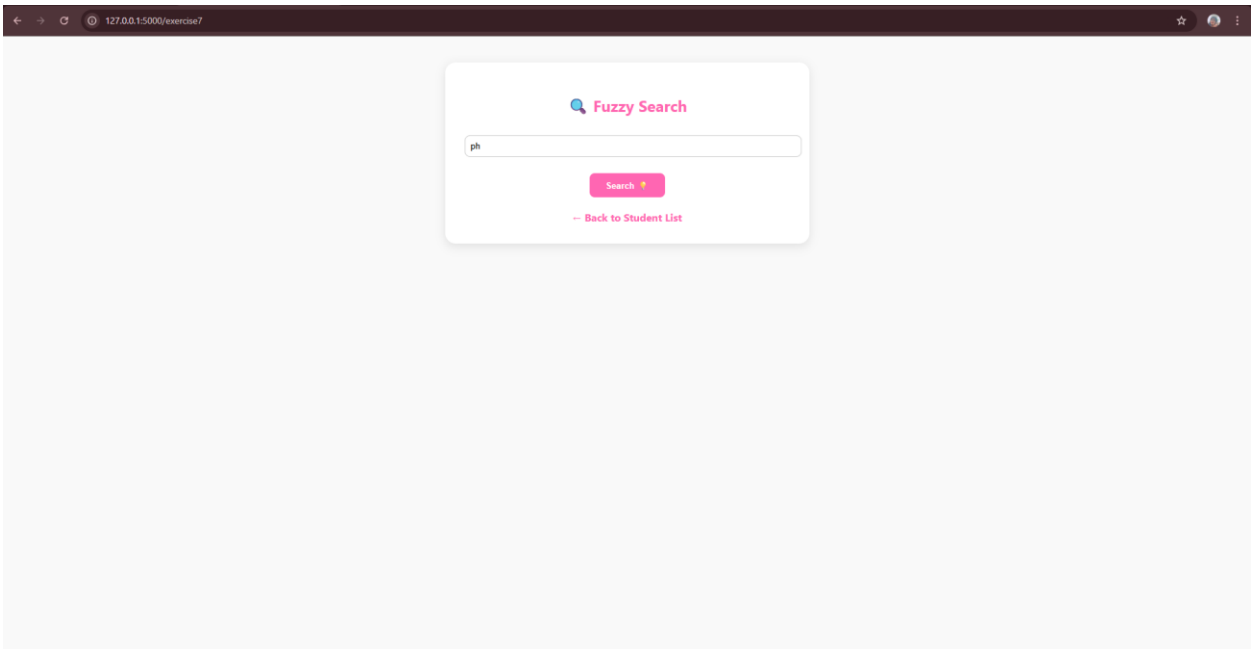
- Search students by full name. Create a search form and display the matching results.
- request.args, find().

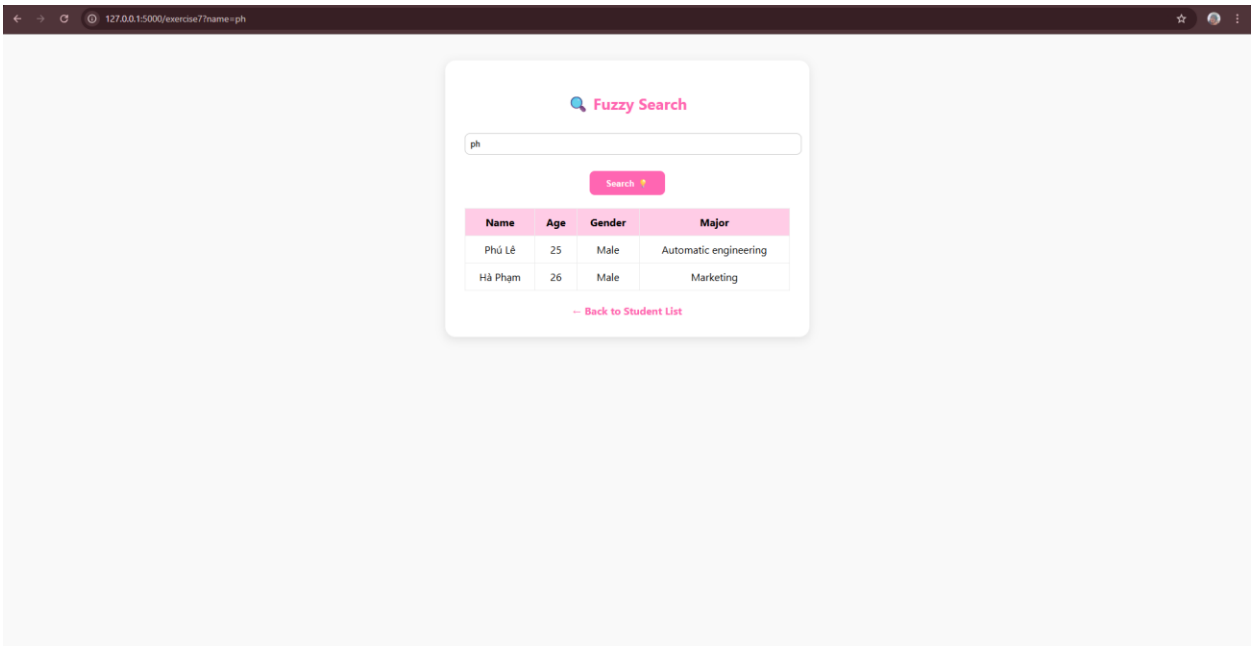




Exercise 7: Fuzzy Search by Name

- Search with approximate name using $regex. Input "an" → matches "An", "Van", etc.
- find() + $regex, case-insensitive option i.

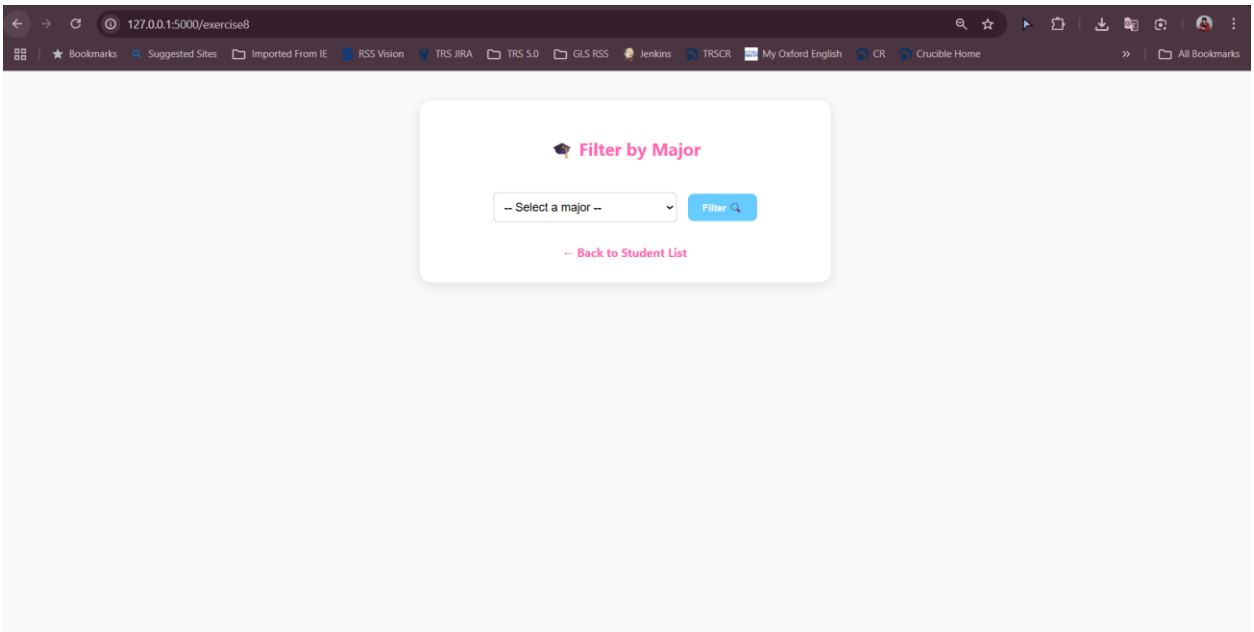


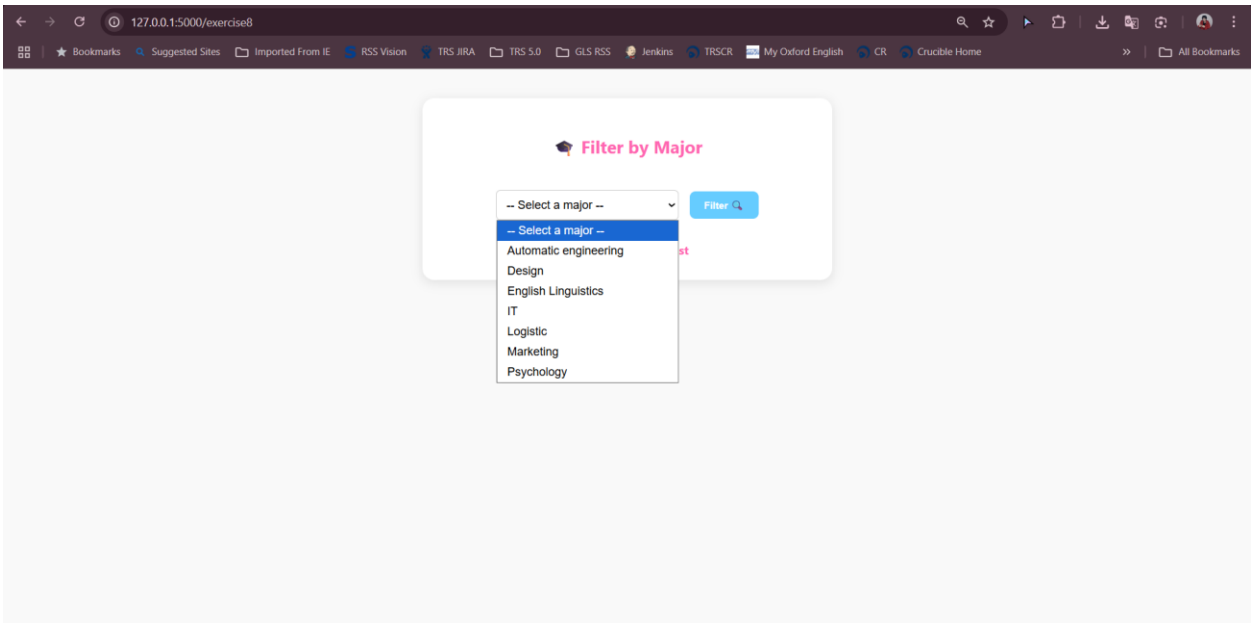


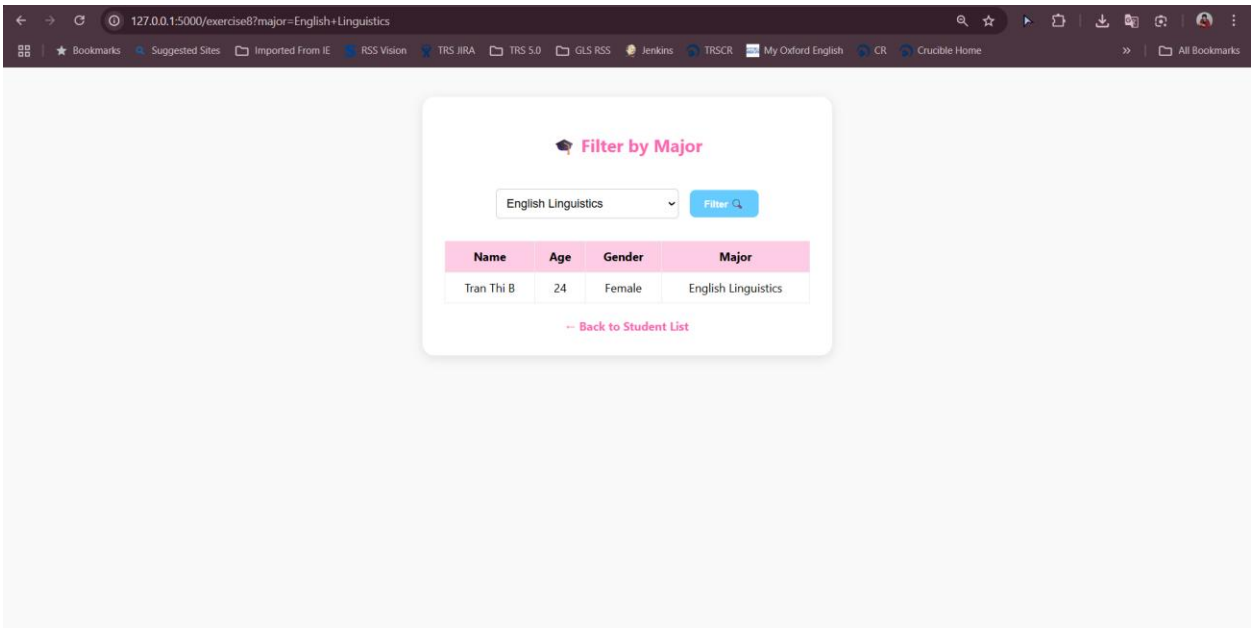
Exercise 8: Filter by Major

• Filter students based on their major. Use a dropdown menu to select a major and display the list of matching students.

• find() with specific field.

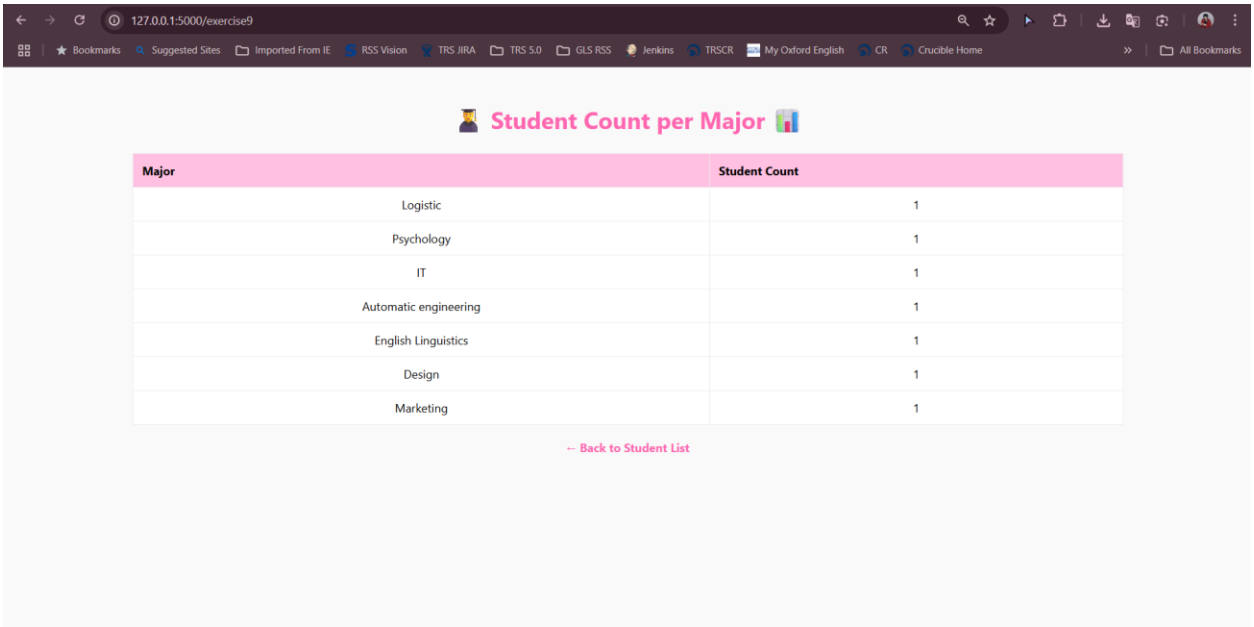






Exercise 9: Count Students per Major

- Count the number of students in each major. Table showing major with student count.
- aggregate() wor manual grouping.

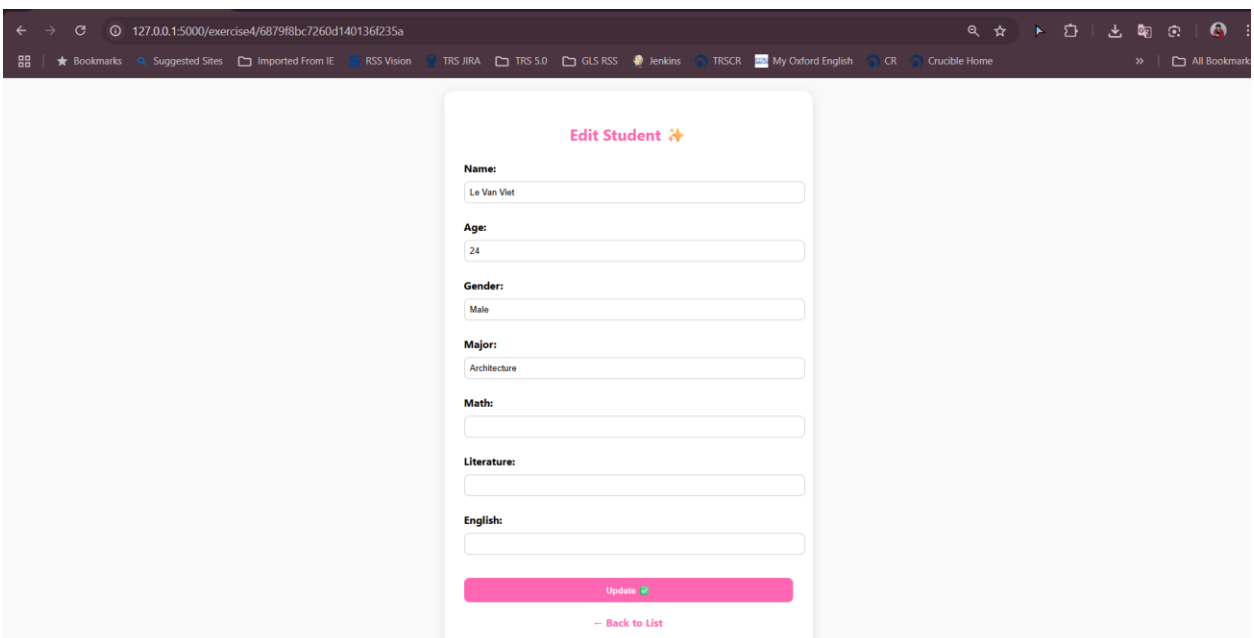


Exercise 10: Add Subject Scores

- Add Math, Literature, and English scores to each student. Edit form to include subject scores.
- update_one()

Name	Age	Gender	Major	GPA	Math	Literature	English	Rank	Edit	Delete
Nhung Nguyễn	23	Female	Design	6.5	7.5	7.0	5.0	Average	Edit	Delete
Phú Lê	25	Male	Automatic engineering	8.6	10.0	7.8	8.0	Excellent	Edit	Delete
Đan Dương	20	Female	IT	9.17	10.0	8.5	9.0	Excellent	Edit	Delete
Tran Thi B	24	Female	English Linguistics	8.77	7.7	8.6	10.0	Excellent	Edit	Delete
Le Van Viet	24	Male	Architecture		N/A	N/A	N/A		Edit	Delete

- When I click “Edit”, it shows “Edit Student” to update Math, Literature, English scores.



Exercise 11: Calculate GPA

- Calculate average of 3 subjects. Display GPA in student table.
- Python math operations, update field.

- Column: GPA

Name	Age	Gender	Major	GPA	Math	Literature	English	Rank	Edit	Delete
Nhung Nguyễn	23	Female	Design	6.5	7.5	7.0	5.0	Average	Edit	Delete
Phú Lê	25	Male	Automatic engineering	8.6	10.0	7.8	8.0	Excellent	Edit	Delete
Đan Dương	20	Female	IT	9.17	10.0	8.5	9.0	Excellent	Edit	Delete

Exercise 12: Academic Ranking

- Classify students based on GPA. Add column "Rank": Excellent, Good, Average.
- Python math operations, update field.

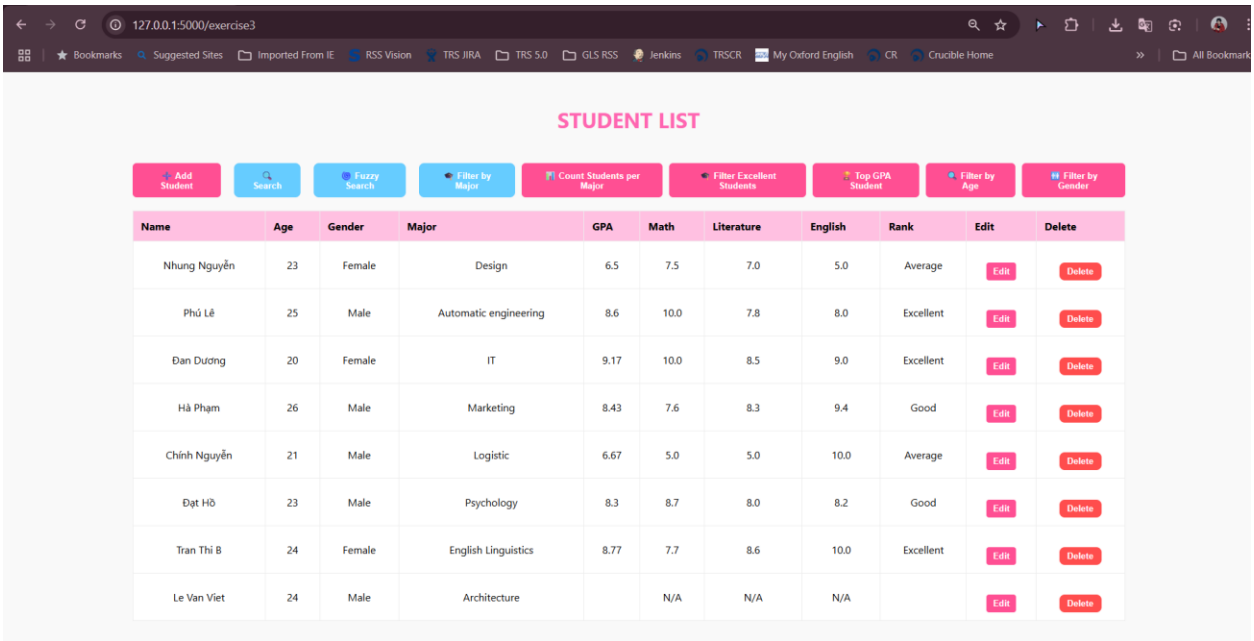
- Column: Rank

Name	Age	Gender	Major	GPA	Math	Literature	English	Rank	Edit	Delete
Nhung Nguyễn	23	Female	Design	6.5	7.5	7.0	5.0	Average	Edit	Delete
Phú Lê	25	Male	Automatic engineering	8.6	10.0	7.8	8.0	Excellent	Edit	Delete
Đan Dương	20	Female	IT	9.17	10.0	8.5	9.0	Excellent	Edit	Delete

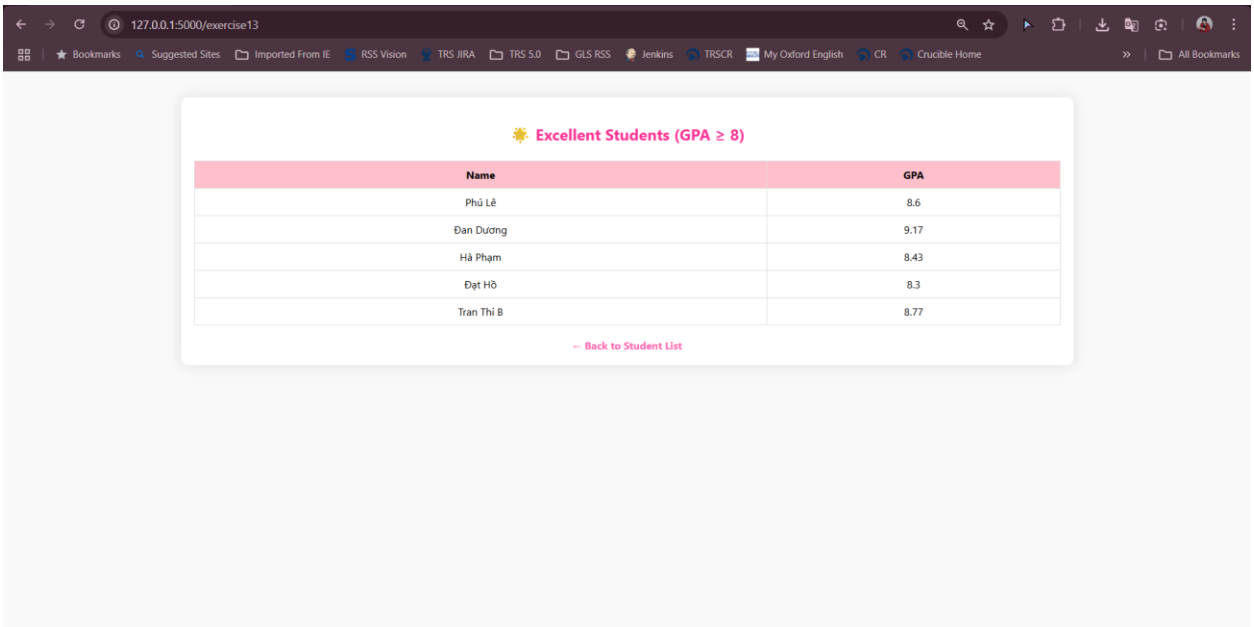
Exercise 13: Filter Excellent Students

- Show students with GPA ≥ 8. Create a popup to display high-achieving students.
- find() with condition.

- Click “Filter Excellent Student” to see students with GPA >= 8



Name	Age	Gender	Major	GPA	Math	Literature	English	Rank	Edit	Delete
Nhung Nguyễn	23	Female	Design	6.5	7.5	7.0	5.0	Average	Edit	Delete
Phú Lê	25	Male	Automatic engineering	8.6	10.0	7.8	8.0	Excellent	Edit	Delete
Đan Dương	20	Female	IT	9.17	10.0	8.5	9.0	Excellent	Edit	Delete
Hà Phạm	26	Male	Marketing	8.43	7.6	8.3	9.4	Good	Edit	Delete
Chính Nguyễn	21	Male	Logistic	6.67	5.0	5.0	10.0	Average	Edit	Delete
Đạt Hồ	23	Male	Psychology	8.3	8.7	8.0	8.2	Good	Edit	Delete
Tran Thi B	24	Female	English Linguistics	8.77	7.7	8.6	10.0	Excellent	Edit	Delete
Le Van Viet	24	Male	Architecture		N/A	N/A	N/A		Edit	Delete

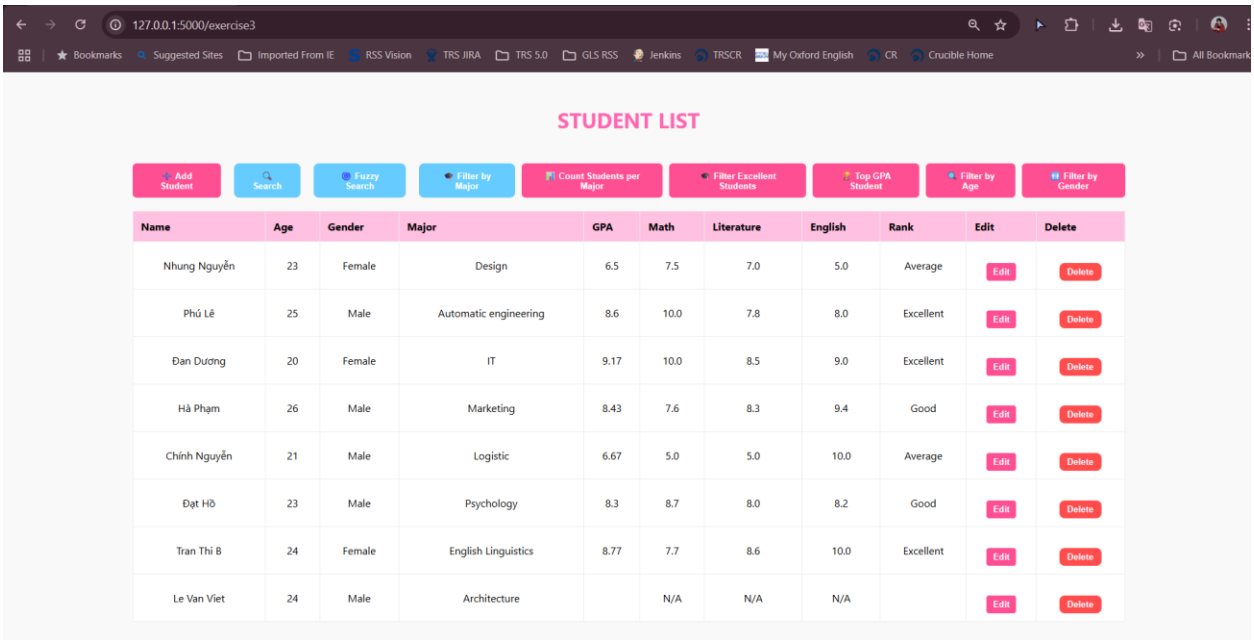


Name	GPA
Phú Lê	8.6
Đan Dương	9.17
Hà Phạm	8.43
Đạt Hồ	8.3
Tran Thi B	8.77

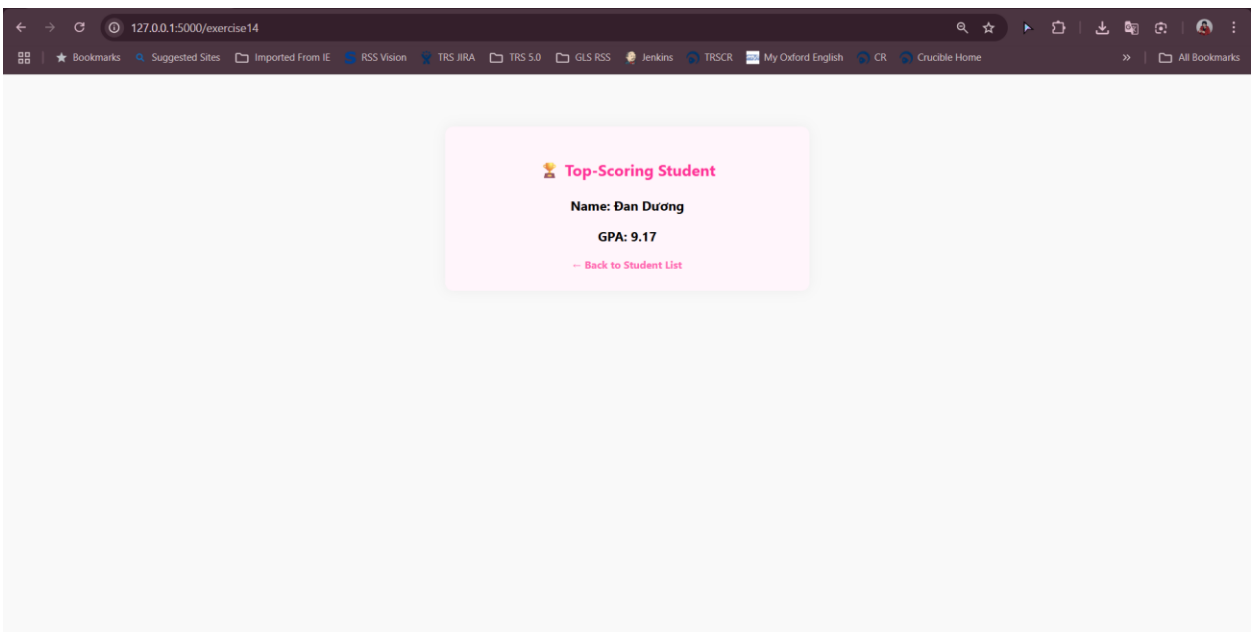
Exercise 14: Find Top-Scoring Student

- Identify student with highest GPA. Display name and GPA of top student.
- sort().limit(1) or max().

- Click “Top-Scoring Student” to see student with highest GPA.

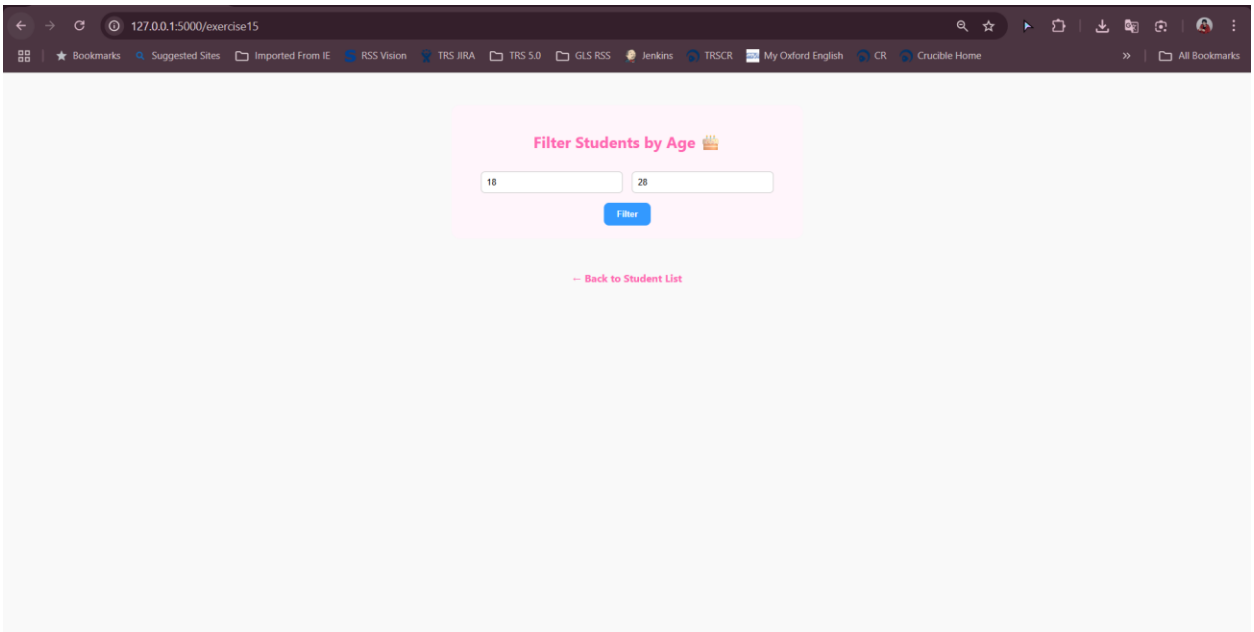


Name	Age	Gender	Major	GPA	Math	Literature	English	Rank	Edit	Delete
Nhung Nguyễn	23	Female	Design	6.5	7.5	7.0	5.0	Average	Edit	Delete
Phú Lê	25	Male	Automatic engineering	8.6	10.0	7.8	8.0	Excellent	Edit	Delete
Đan Dương	20	Female	IT	9.17	10.0	8.5	9.0	Excellent	Edit	Delete
Hà Phạm	26	Male	Marketing	8.43	7.6	8.3	9.4	Good	Edit	Delete
Chính Nguyễn	21	Male	Logistic	6.67	5.0	5.0	10.0	Average	Edit	Delete
Đạt Hồ	23	Male	Psychology	8.3	8.7	8.0	8.2	Good	Edit	Delete
Tran Thi B	24	Female	English Linguistics	8.77	7.7	8.6	10.0	Excellent	Edit	Delete
Le Van Viet	24	Male	Architecture		N/A	N/A	N/A		Edit	Delete

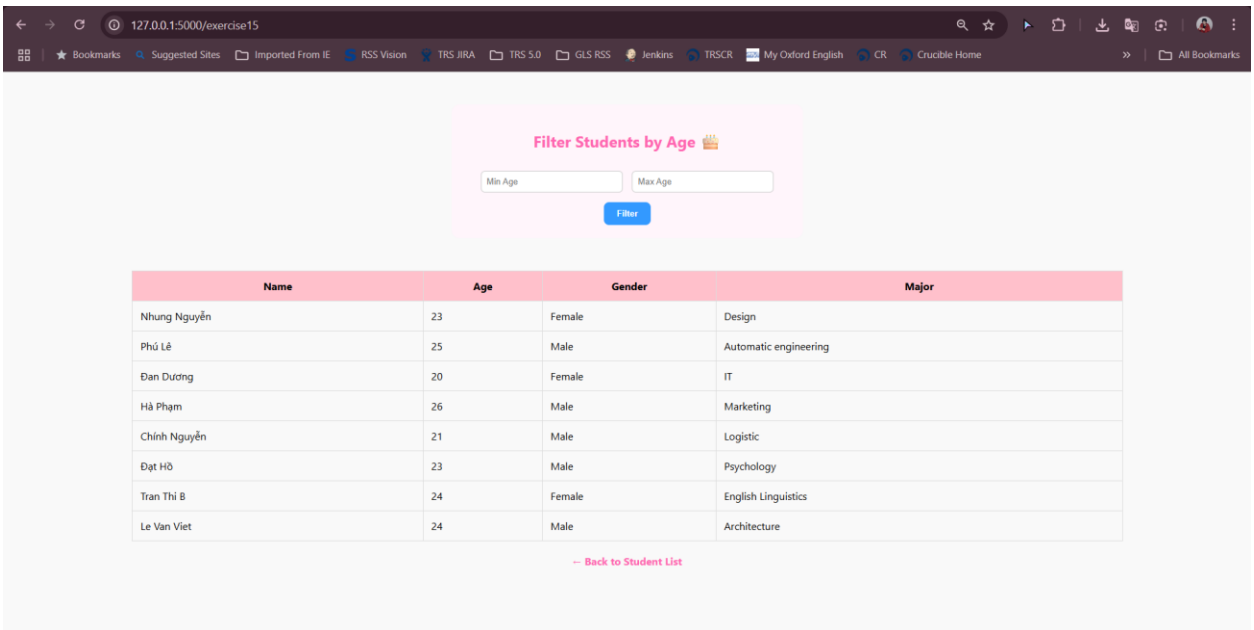


Exercise 15: Filter by Age

- Filter students based on age range. Input min & max age, return matching results.
- find() with range query.



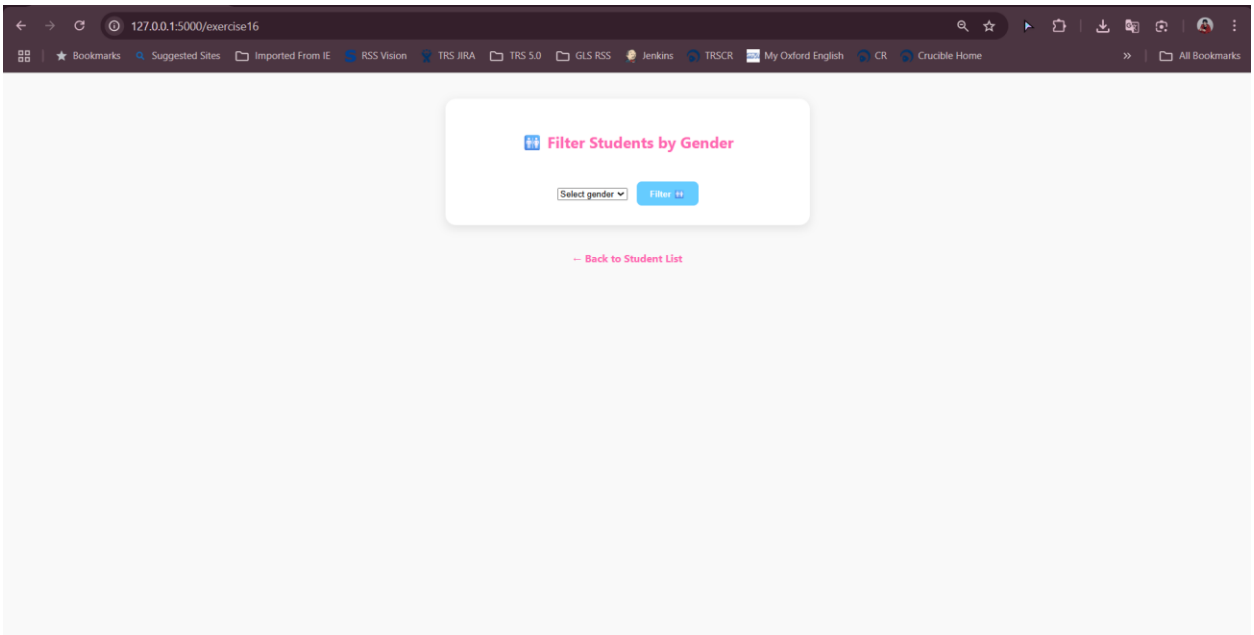
- When I enter the age between 18 and 28, it shows the result below:

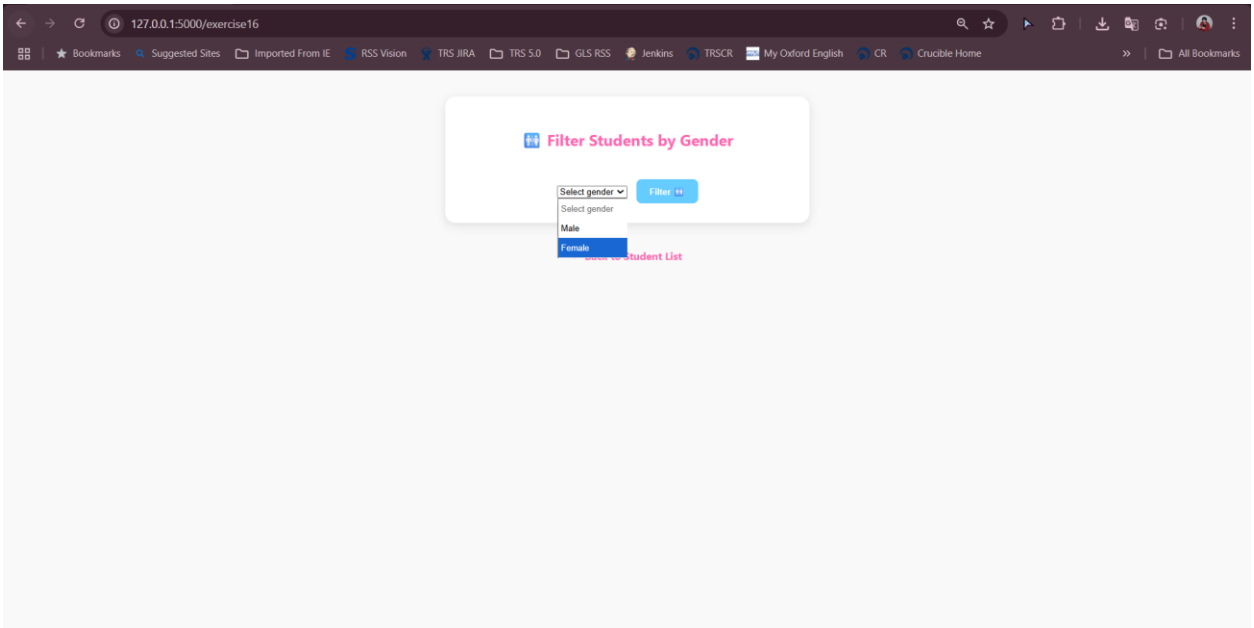


Name	Age	Gender	Major
Nhung Nguyễn	23	Female	Design
Phú Lê	25	Male	Automatic engineering
Đan Dương	20	Female	IT
Hà Phạm	26	Male	Marketing
Chính Nguyễn	21	Male	Logistic
Đạt Hồ	23	Male	Psychology
Tran Thi B	24	Female	English Linguistics
Le Van Viet	24	Male	Architecture

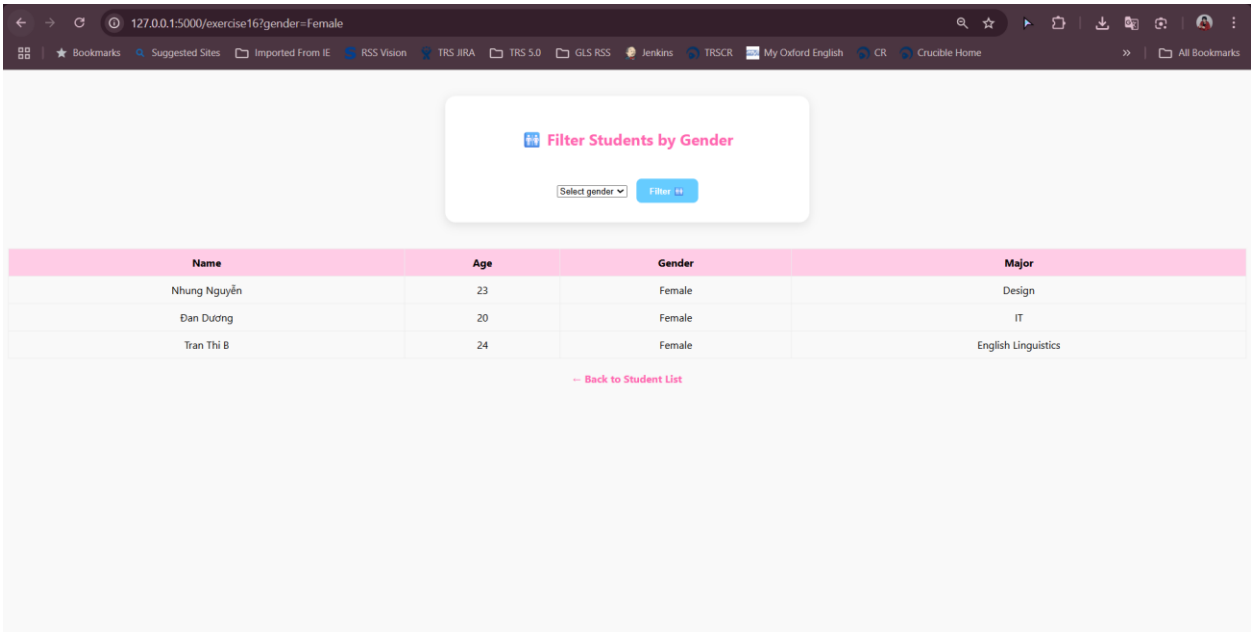
Exercise 16: Filter by Gender

- Filter students by gender (male/female). Gender selection form, show results.
- find() by gender field.





- When I choose “Female”, it shows list of female students below:

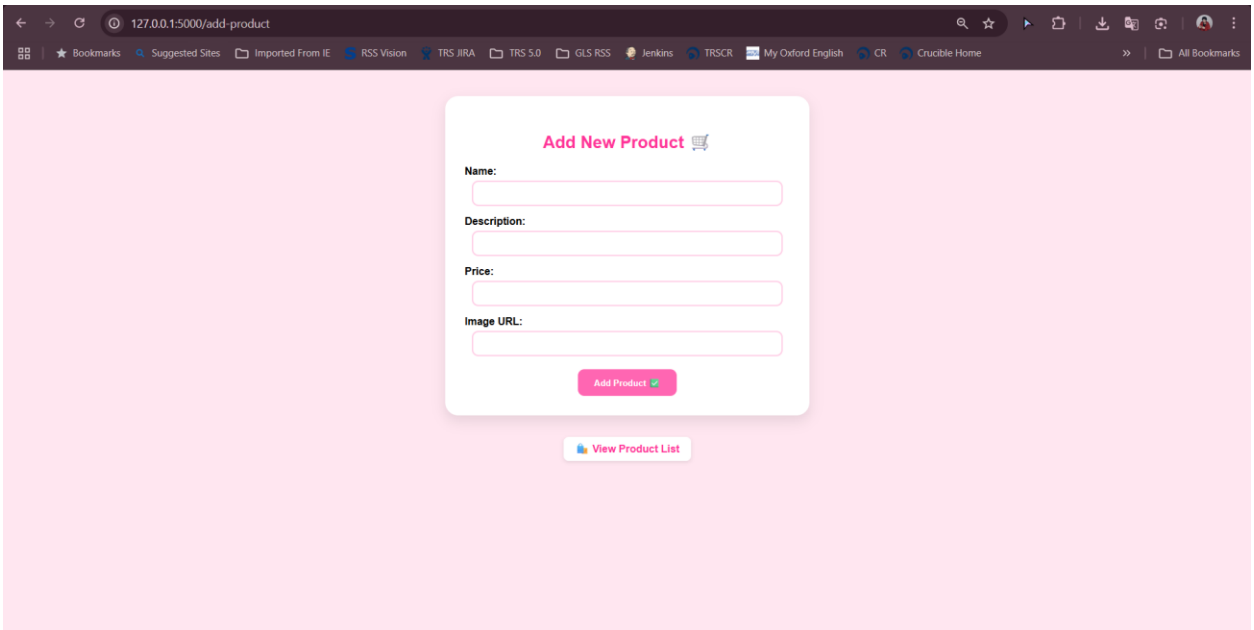


11 Additional Exercises

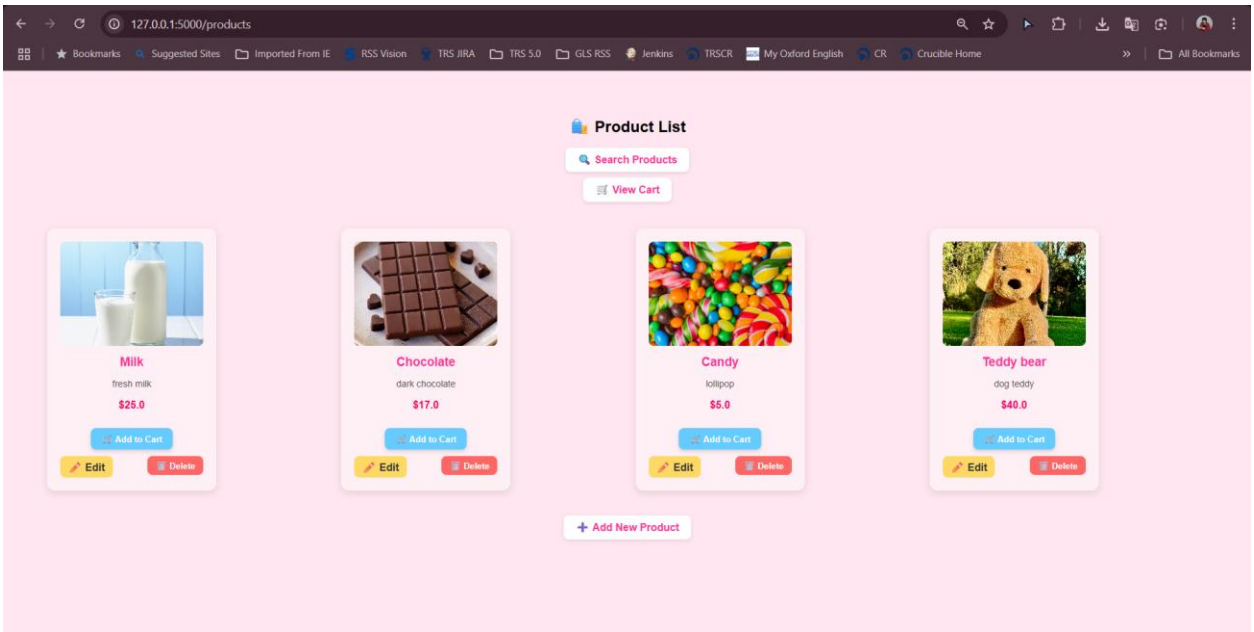
Build Online Store Website with Flask and MongoDB

- Simple web application for selling products online.
- Users browse products, add to cart, and place orders.
- Admins manage products: add, edit, delete. Features include:

1. Connecting Flask to MongoDB
2. Add New Product

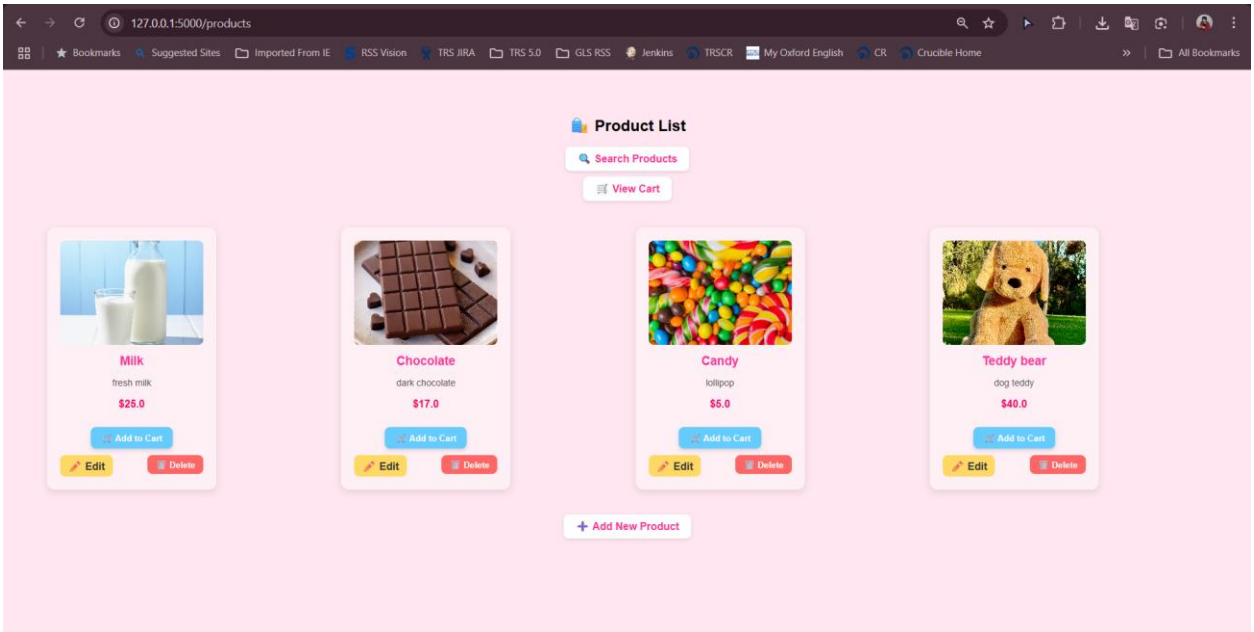


3. Display Product List

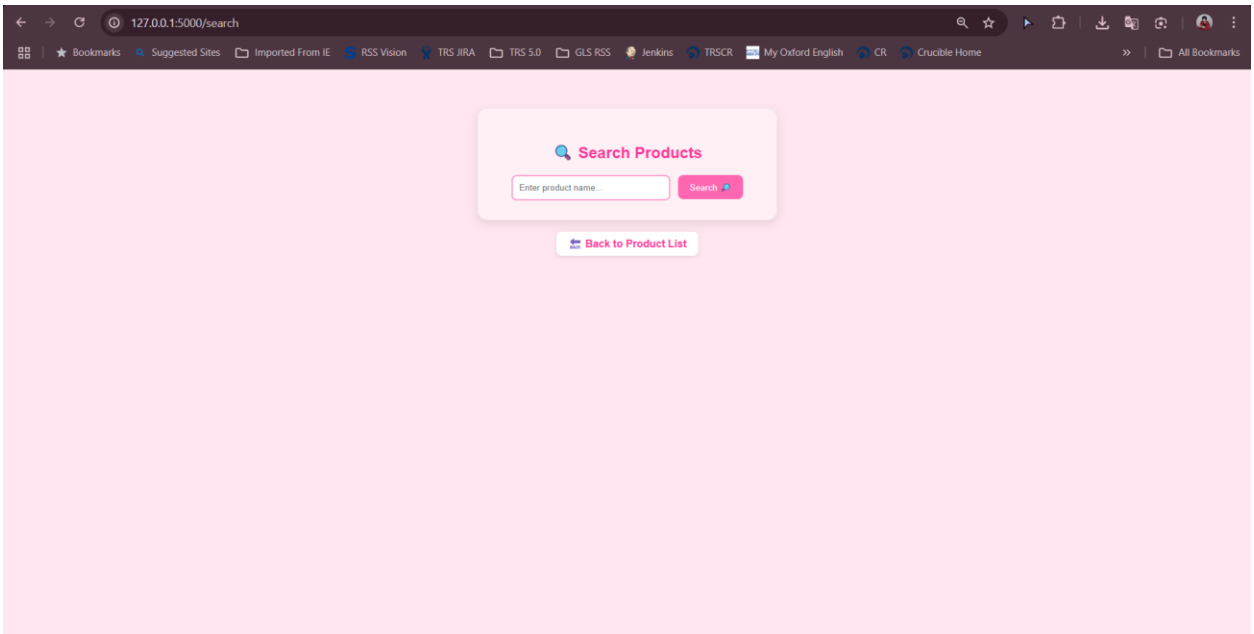


4. Admin Product Page

- Here, I integrate the admin interface with the product list display. This is the interface that displays the product list, and the admin can also manage the products.

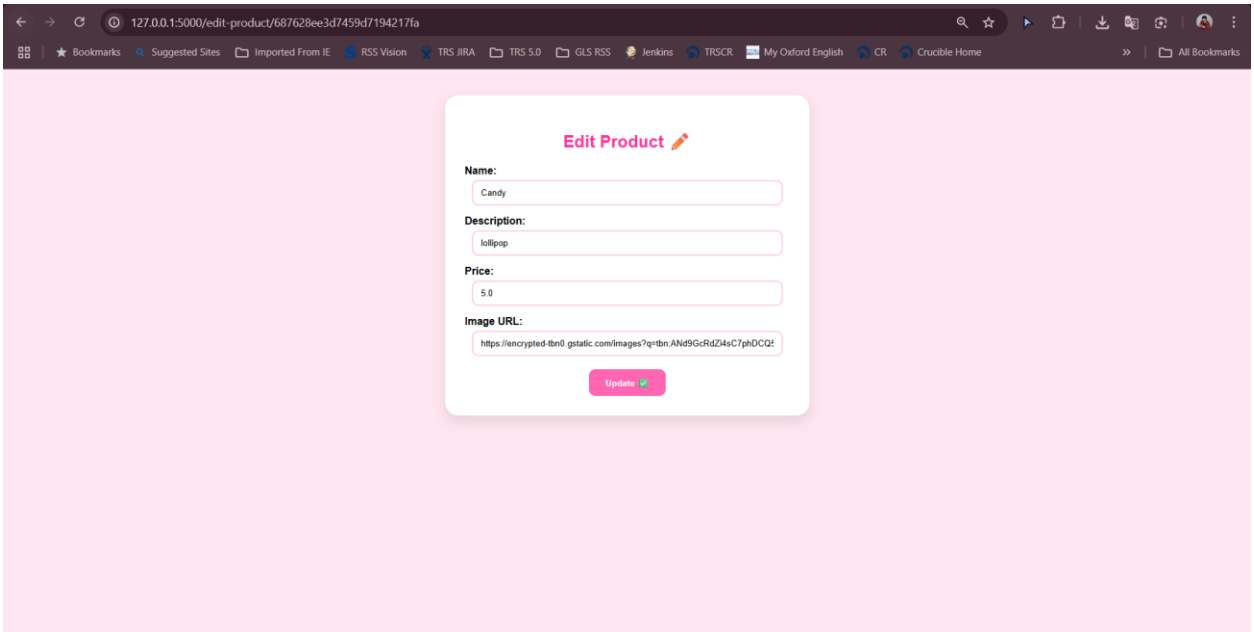


5. Search Products



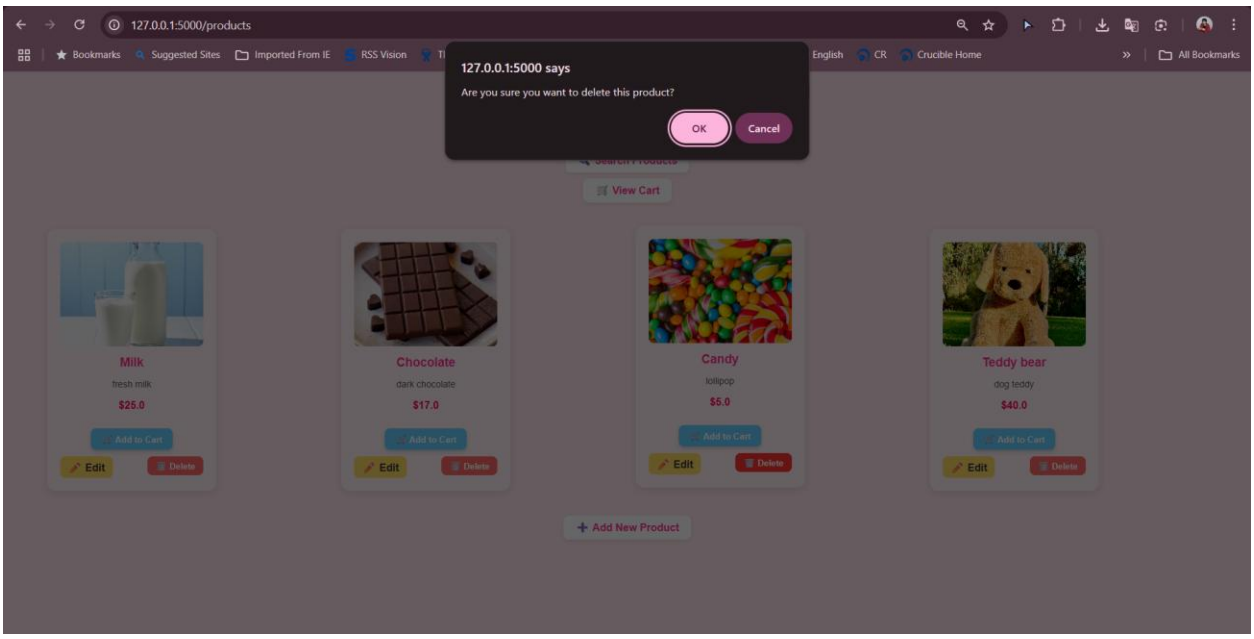
6. Edit / Delete Product

- Edit:

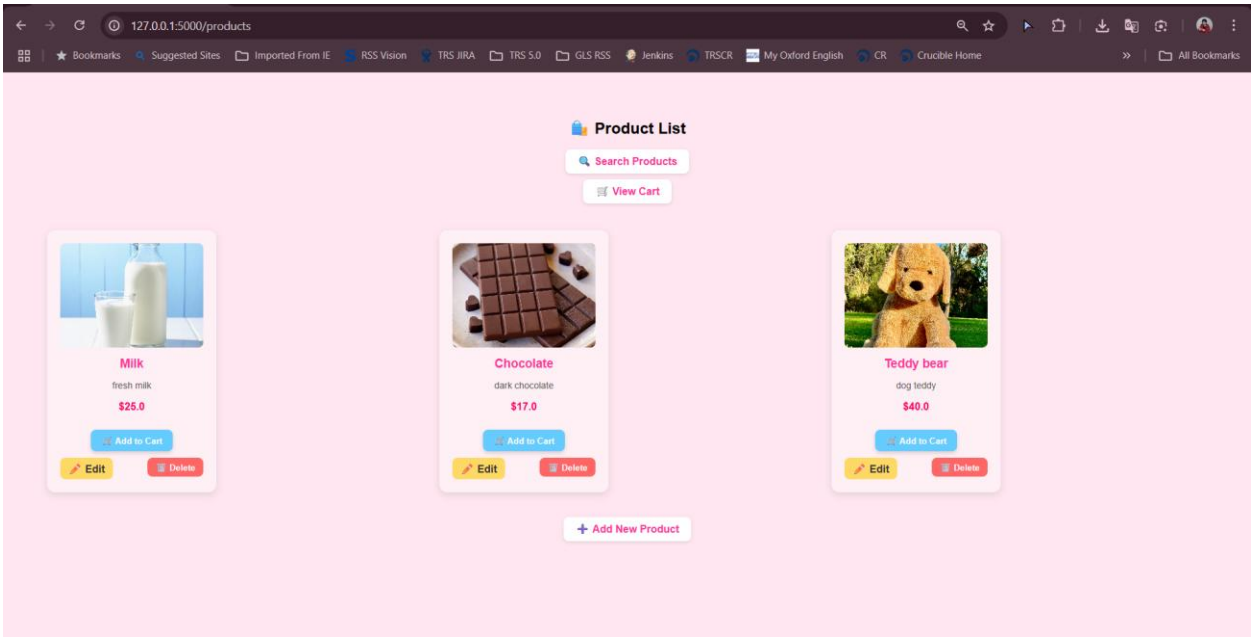


- Delete:

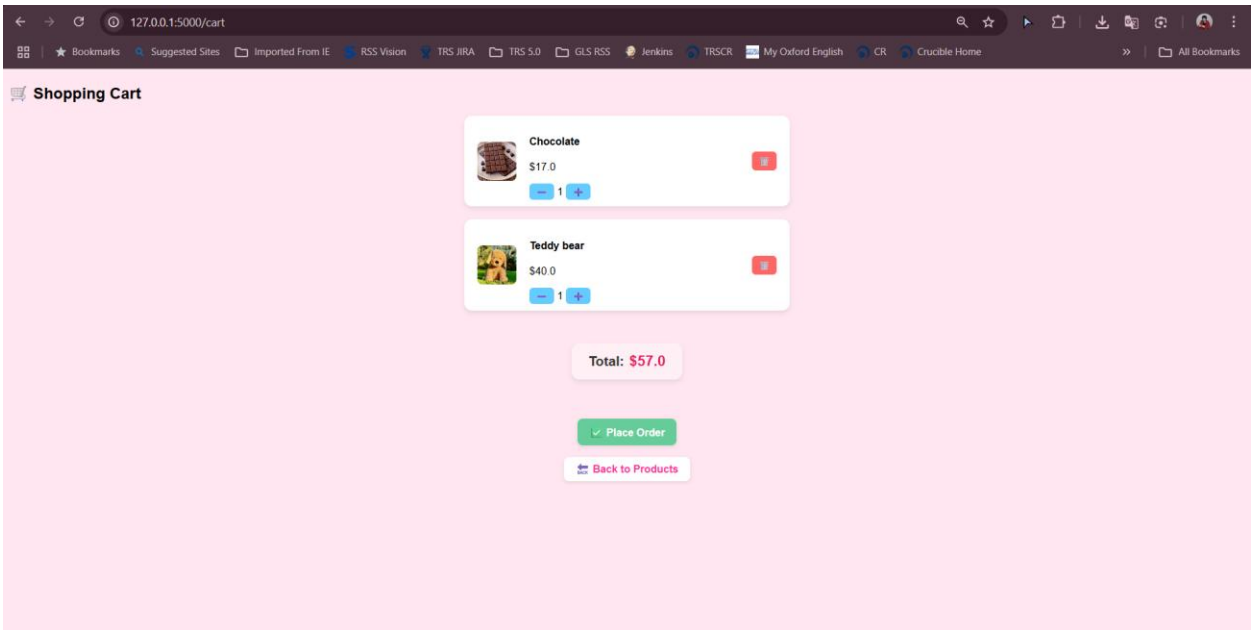
When I click button “Delete” in Candy, it shows notification below:



After Delete:

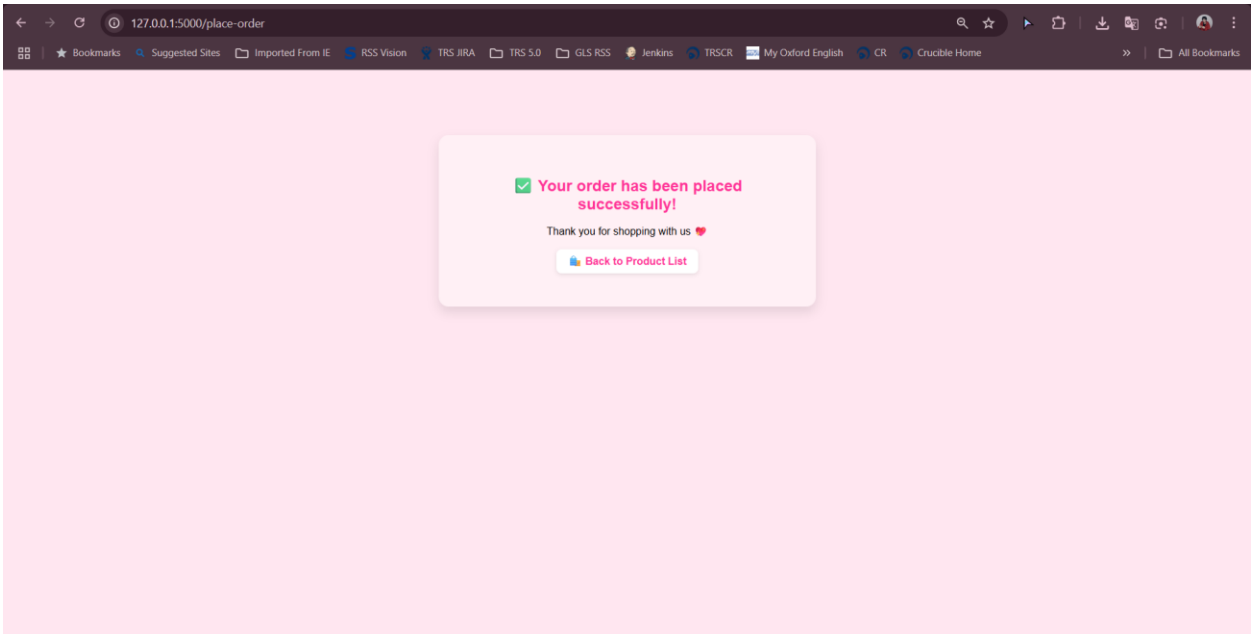


7. Shopping Cart



8. Checkout

When I Click “Place Order”, it show message below:


127.0.0.1:5000/place-order
Bookmarks
Suggested Sites
Imported From IE
RSS Vision
TRS JIRA
TRS 5.0
GLS RSS
Jenkins
TRSCR
My Oxford English
CR
Crucible Home
All Bookmarks
Your order has been placed successfully!
Thank you for shopping with us
Back to Product List