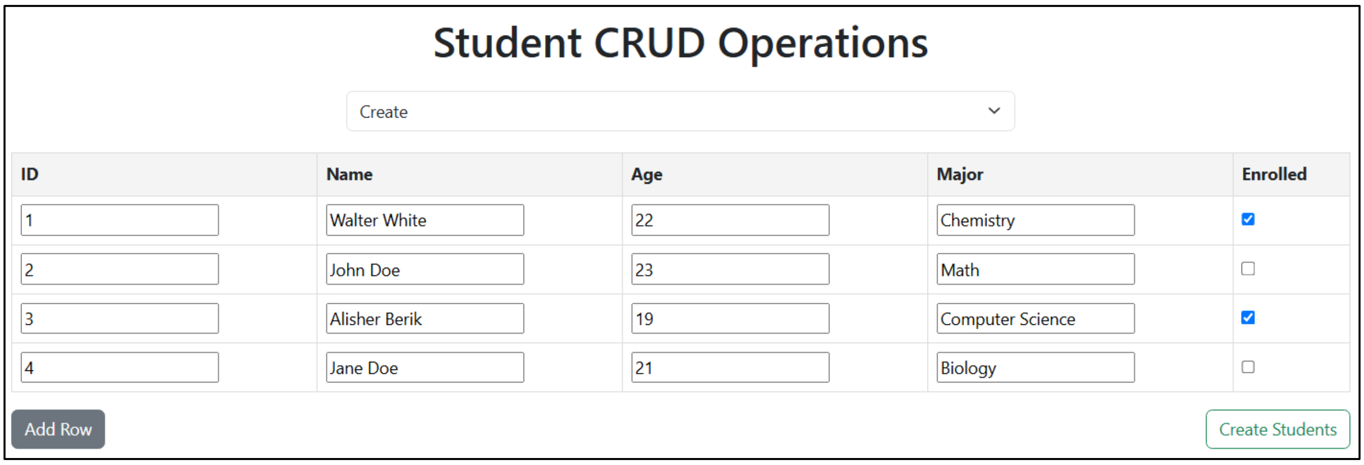
Class Lab work 1 – Week 6 – Berik Alisher – IT-2308 – ID: 230577

Task A (Create)

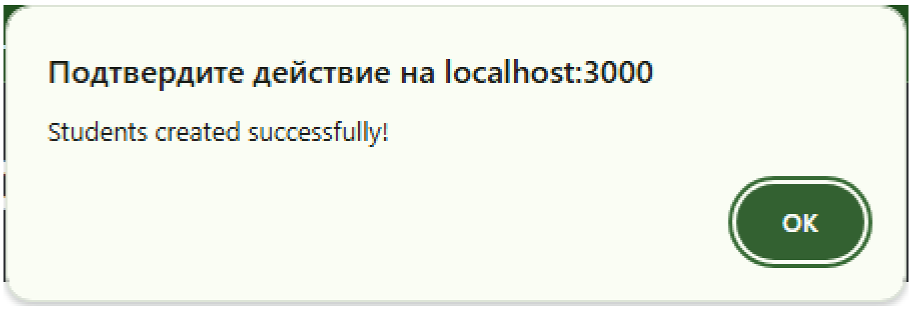
I created 4 students in one array



Student CRUD Operations

Create

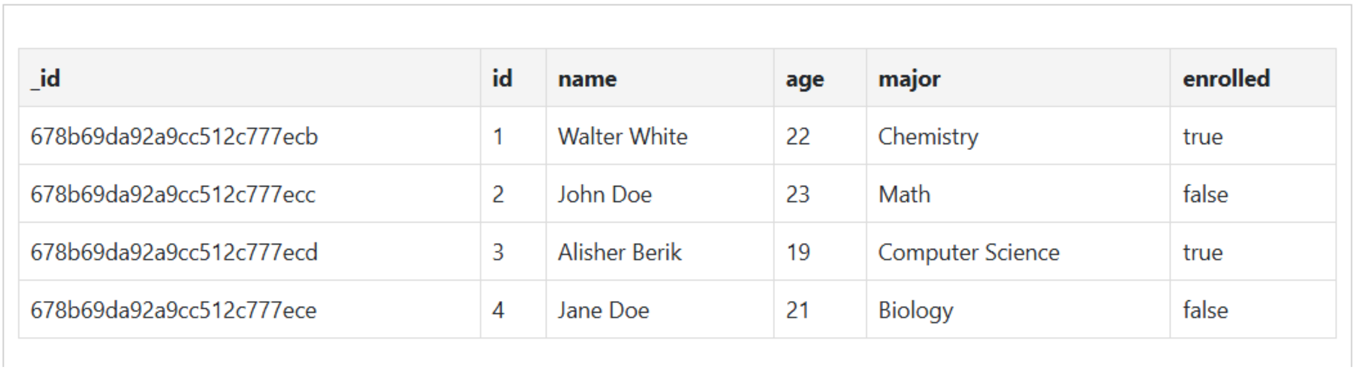
ID	Name	Age	Major	Enrolled
1	Walter White	22	Chemistry	✓
2	John Doe	23	Math	☐
3	Alisher Berik	19	Computer Science	✓
4	Jane Doe	21	Biology	☐



Подтвердите действие на localhost:3000

Students created successfully!

Results:



_id	id	name	age	major	enrolled
678b69da92a9cc512c777ecb	1	Walter White	22	Chemistry	true
678b69da92a9cc512c777ecc	2	John Doe	23	Math	false
678b69da92a9cc512c777ecd	3	Alisher Berik	19	Computer Science	true
678b69da92a9cc512c777ece	4	Jane Doe	21	Biology	false

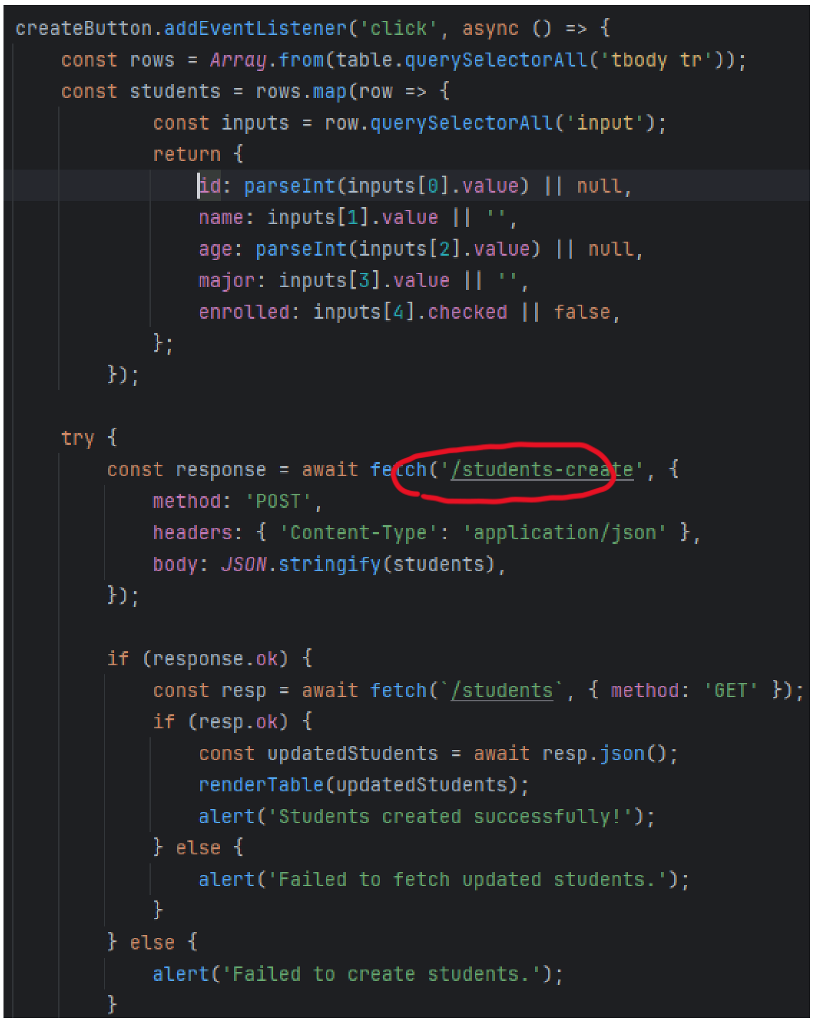
Code:

There I retrieve values from inputs and put them into the **common array** as individual objects.

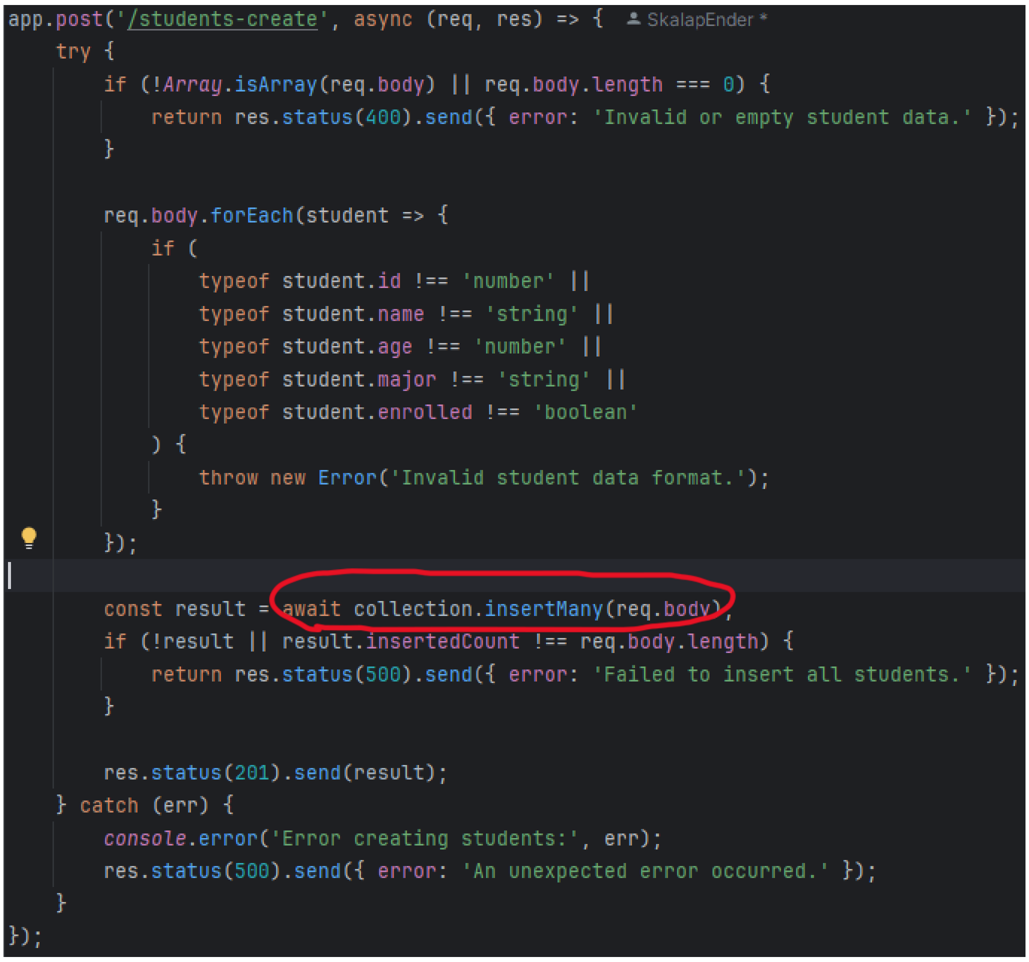
Then, I convert the array to JSON array because Mongo accepts only JSON objects.

Finally, I send JSON to server that checks data and performs inserting to database.

Query: db.students.insertMany([{student 1 data}, {student 2 data}, …]



```javascript
createButton.addEventListener('click', async () => {
    const rows = Array.from(table.querySelectorAll('tbody tr'));
    const students = rows.map(row => {
        const inputs = row.querySelectorAll('input');
        return {
            id: parseInt(inputs[0].value) || null,
            name: inputs[1].value || '',
            age: parseInt(inputs[2].value) || null,
            major: inputs[3].value || '',
            enrolled: inputs[4].checked || false,
        };
    });

    try {
        const response = await fetch('/students-create', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(students),
        });

        if (response.ok) {
            const resp = await fetch(`/students`, { method: 'GET' });
            if (resp.ok) {
                const updatedStudents = await resp.json();
                renderTable(updatedStudents);
                alert('Students created successfully!');
            } else {
                alert('Failed to fetch updated students.');
            }
        } else {
            alert('Failed to create students.');
        }
```



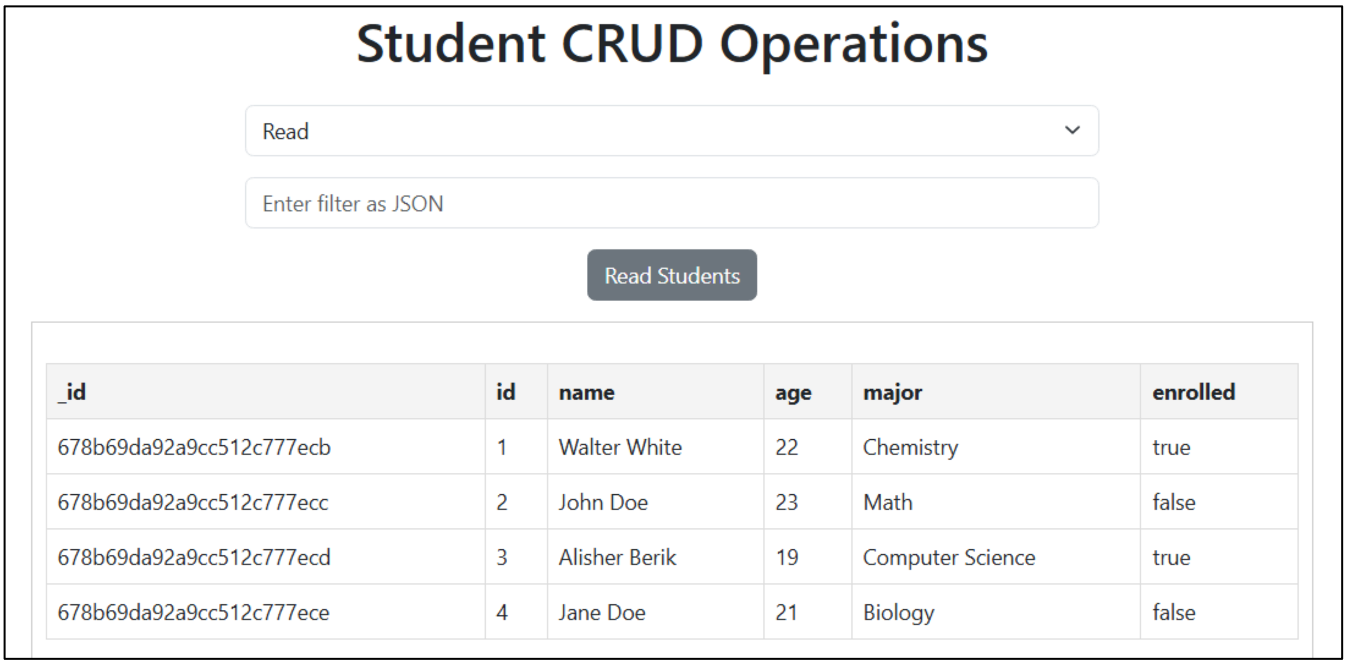
```javascript
app.post('/students-create', async (req, res) => {  SkalapEnder *
    try {
        if (!Array.isArray(req.body) || req.body.length === 0) {
            return res.status(400).send({ error: 'Invalid or empty student data.' });
        }

        req.body.forEach(student => {
            if (
                typeof student.id !== 'number' ||
                typeof student.name !== 'string' ||
                typeof student.age !== 'number' ||
                typeof student.major !== 'string' ||
                typeof student.enrolled !== 'boolean'
            ) {
                throw new Error('Invalid student data format.');
            }
        });

        const result = await collection.insertMany(req.body);
        if (!result || result.insertedCount !== req.body.length) {
            return res.status(500).send({ error: 'Failed to insert all students.' });
        }

        res.status(201).send(result);
    } catch (err) {
        console.error('Error creating students:', err);
        res.status(500).send({ error: 'An unexpected error occurred.' });
    }
});
```

Task B (Read)

Left filter empty if you want to get all students.

Otherwise, write filter with bracket and fields in quotes. Example: {“age”: {“$gt”: 20}}

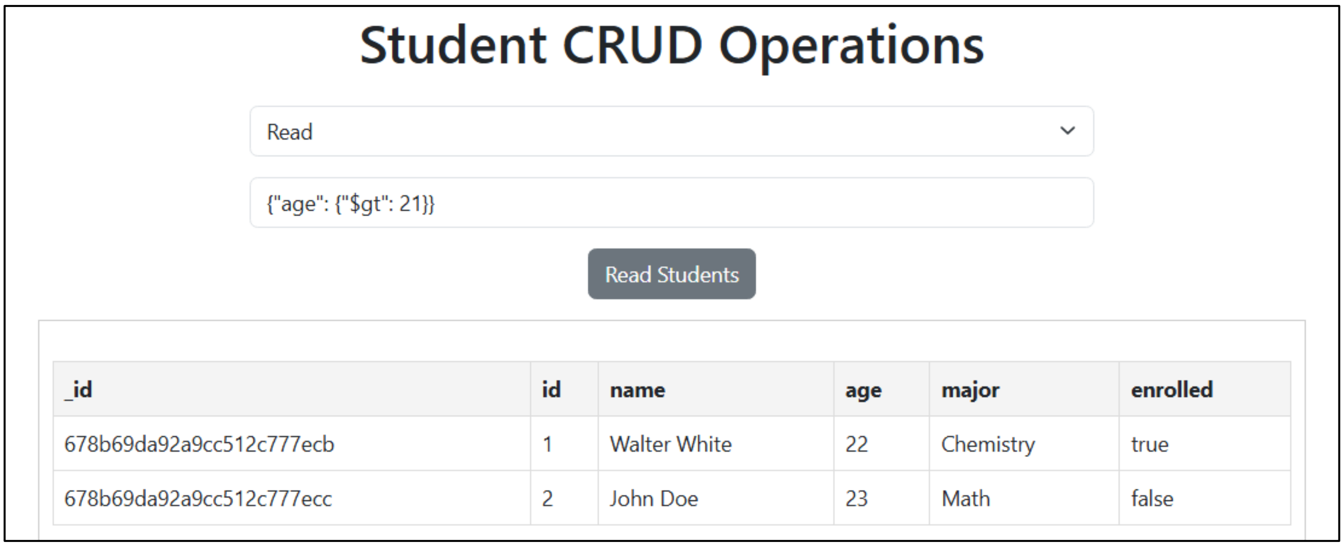
Without filter



Student CRUD Operations

_id	id	name	age	major	enrolled
678b69da92a9cc512c777ecb	1	Walter White	22	Chemistry	true
678b69da92a9cc512c777ecc	2	John Doe	23	Math	false
678b69da92a9cc512c777ecd	3	Alisher Berik	19	Computer Science	true
678b69da92a9cc512c777ece	4	Jane Doe	21	Biology	false

With filter



Student CRUD Operations

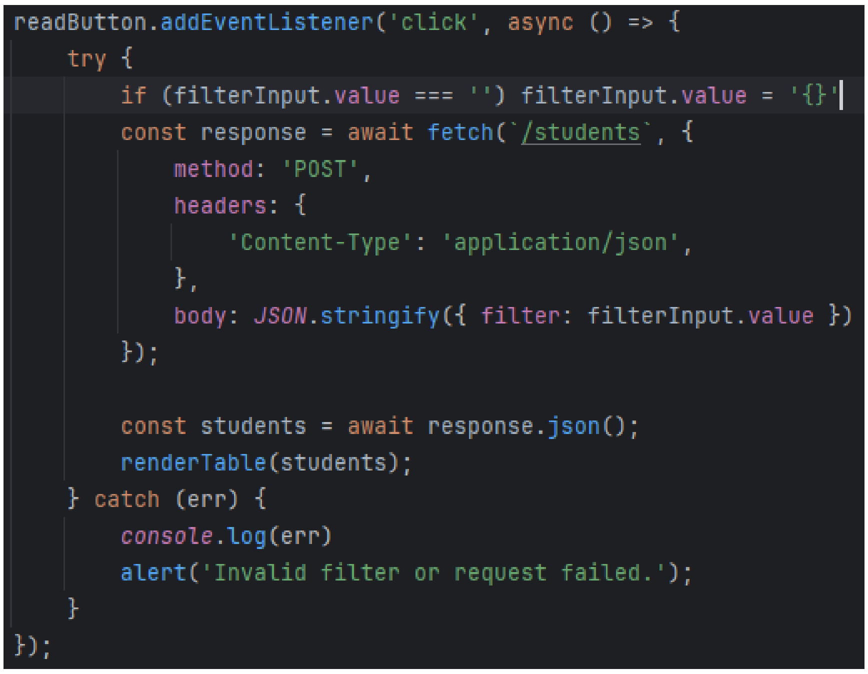
_id	id	name	age	major	enrolled
678b69da92a9cc512c777ecb	1	Walter White	22	Chemistry	true
678b69da92a9cc512c777ecc	2	John Doe	23	Math	false

The code:

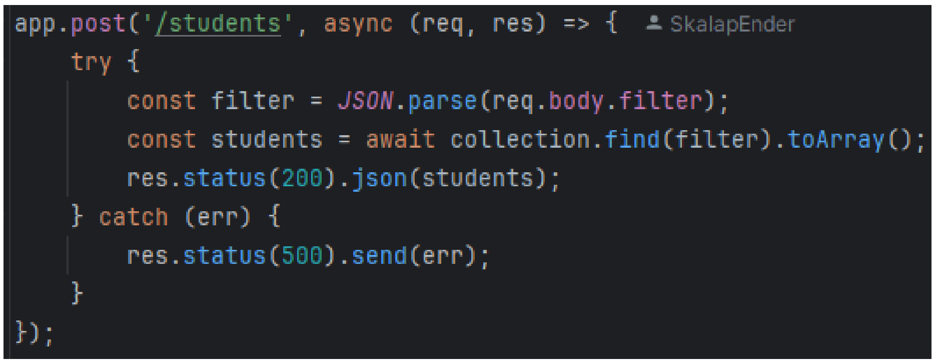
I created condition if user didn’t write anything then we put empty curve brackets. It means no filters.

Query to find all: db.students.find({})

Query with filter: db.students.find({ *filter* })



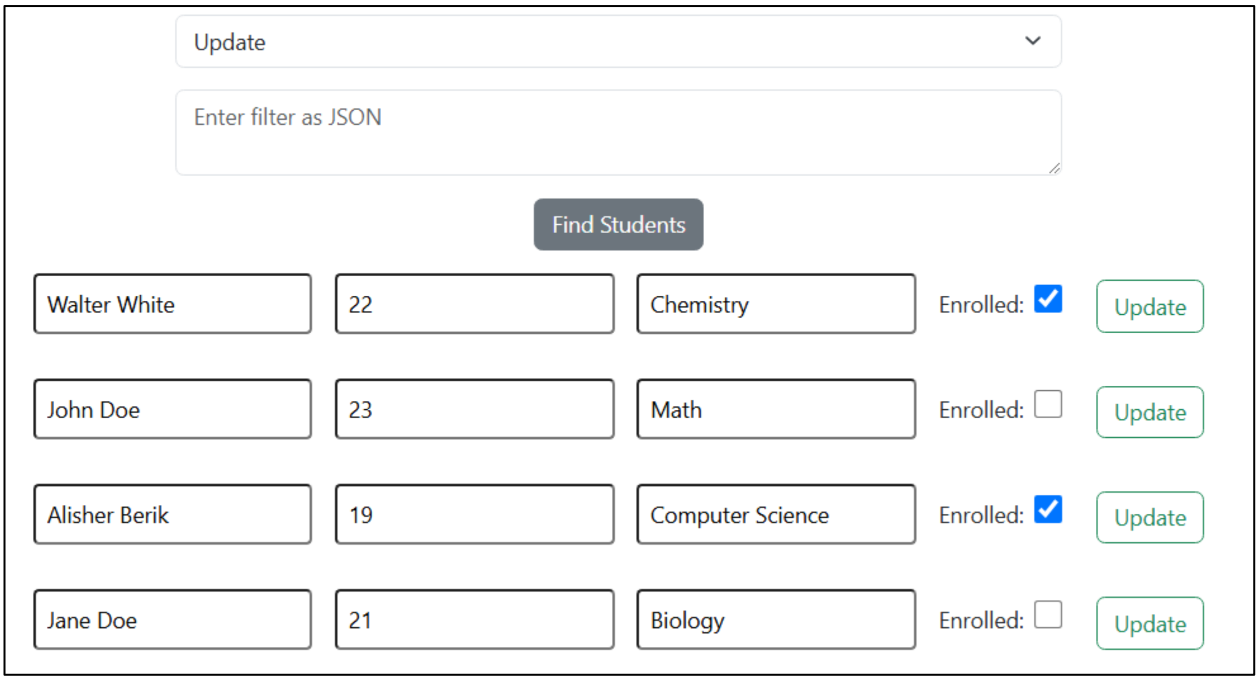
```
readButton.addEventListener('click', async () => {
    try {
        if (filterInput.value === '') filterInput.value = '{}'
        const response = await fetch(`/students`, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({ filter: filterInput.value })
        });

        const students = await response.json();
        renderTable(students);
    } catch (err) {
        console.log(err)
        alert('Invalid filter or request failed.');
    }
});
```



```
app.post('/students', async (req, res) => {  SkalapEnder
    try {
        const filter = JSON.parse(req.body.filter);
        const students = await collection.find(filter).toArray();
        res.status(200).json(students);
    } catch (err) {
        res.status(500).send(err);
    }
});
```

Task C (Update)

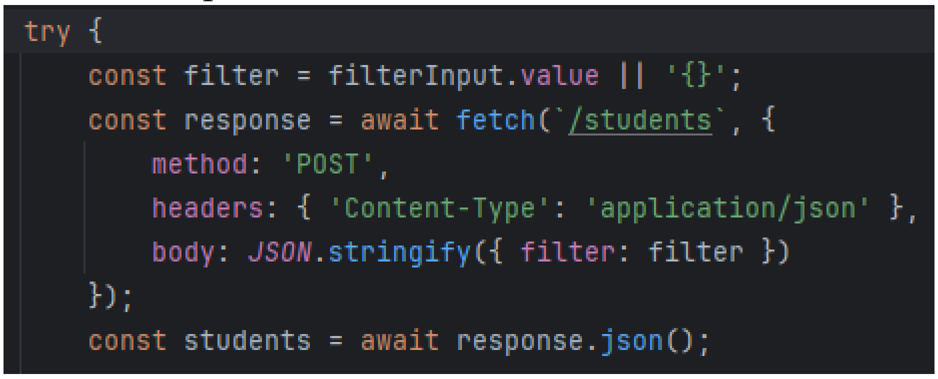
There we can dynamically change student’s data and press Update button. After pressing, we send PUT method to server in order to modify data.



Walter White	22	Chemistry	Enrolled: ✓	Update
John Doe	23	Math	Enrolled: ☐	Update
Alisher Berik	19	Computer Science	Enrolled: ✓	Update
Jane Doe	21	Biology	Enrolled: ☐	Update

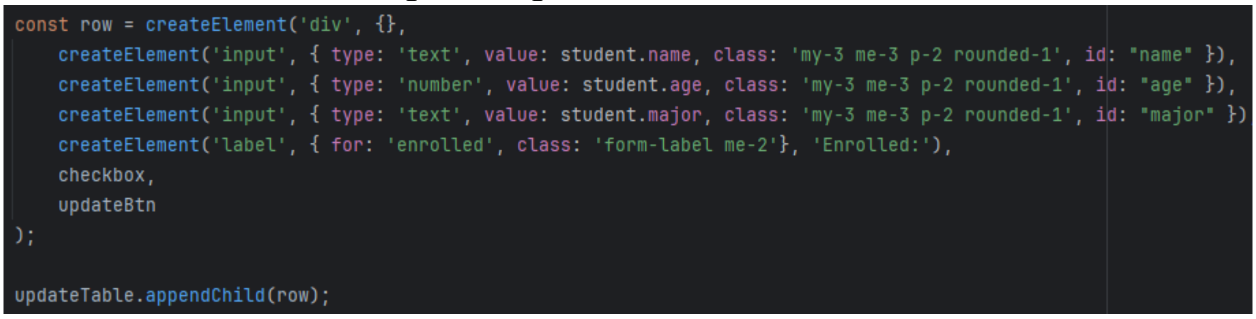
Code:

Request list of students based on filter



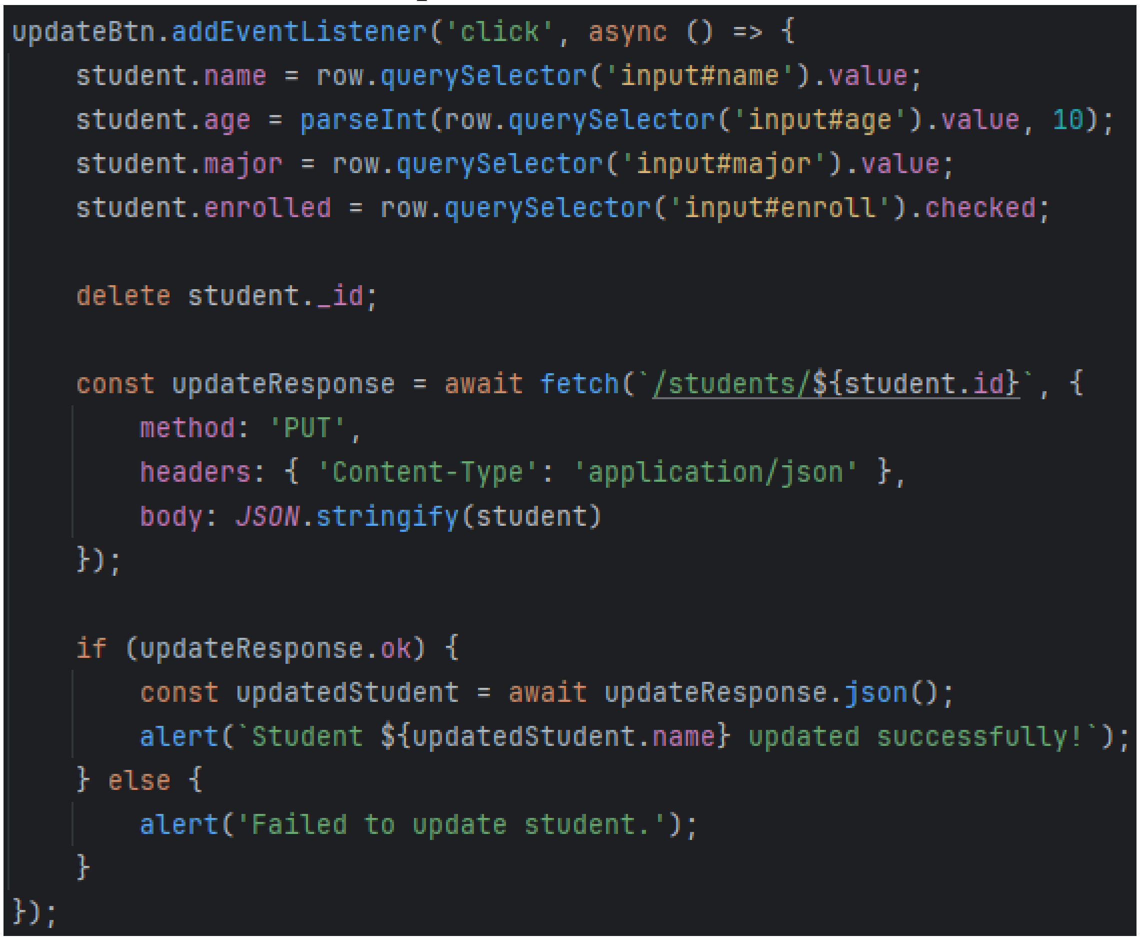
```
try {
    const filter = filterInput.value || '{}';
    const response = await fetch(`/students`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ filter: filter })
    });
    const students = await response.json();
```

Create inputs to put student's data into them



```
const row = createElement('div', {},
    createElement('input', { type: 'text', value: student.name, class: 'my-3 me-3 p-2 rounded-1', id: "name" }),
    createElement('input', { type: 'number', value: student.age, class: 'my-3 me-3 p-2 rounded-1', id: "age" }),
    createElement('input', { type: 'text', value: student.major, class: 'my-3 me-3 p-2 rounded-1', id: "major" })
    createElement('label', { for: 'enrolled', class: 'form-label me-2'}, 'Enrolled:'),
    checkbox,
    updateBtn
);

updateTable.appendChild(row);
```

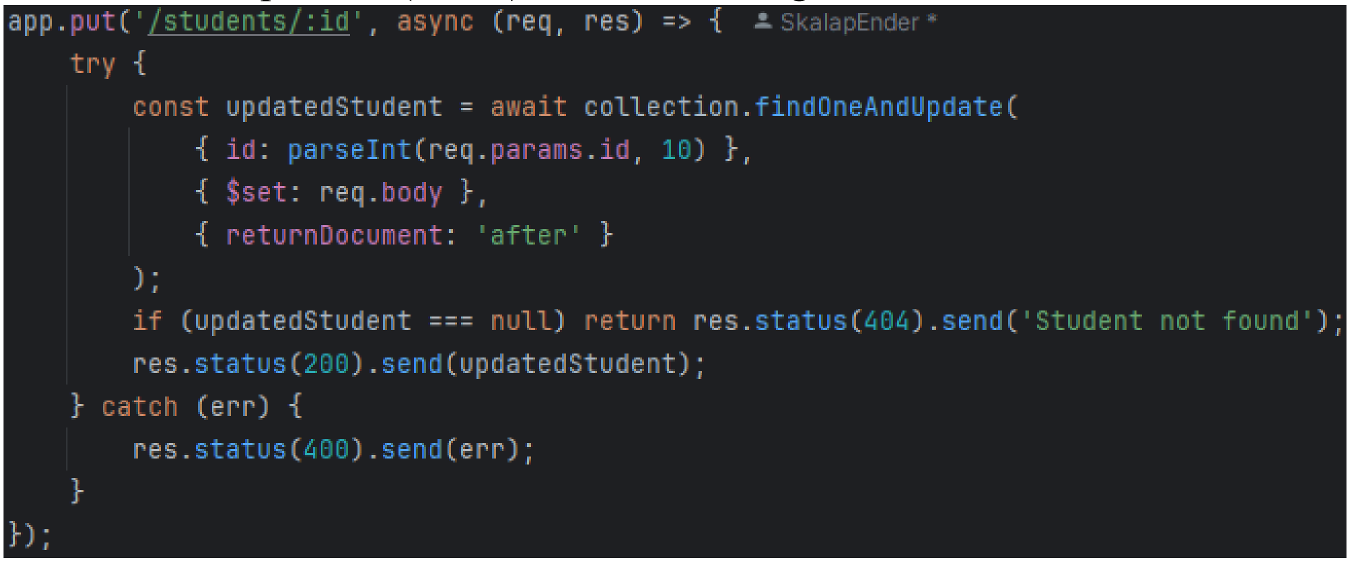
Update student's data



```
updateBtn.addEventListener('click', async () => {
    student.name = row.querySelector('input#name').value;
    student.age = parseInt(row.querySelector('input#age').value, 10);
    student.major = row.querySelector('input#major').value;
    student.enrolled = row.querySelector('input#enroll').checked;

    delete student._id;

    const updateResponse = await fetch(`/students/${student.id}`, {
        method: 'PUT',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(student)
    });

    if (updateResponse.ok) {
        const updatedStudent = await updateResponse.json();
        alert(`Student ${updatedStudent.name} updated successfully!`);
    } else {
        alert('Failed to update student.');
    }
});
```

In server, we send findOneAndUpdate query to find student by his ID. Also, this query return document. I check if returned document is exist in order to check if student's data was modified in database.

P.S.: parseInt(id, 10) means that we get number on base 10

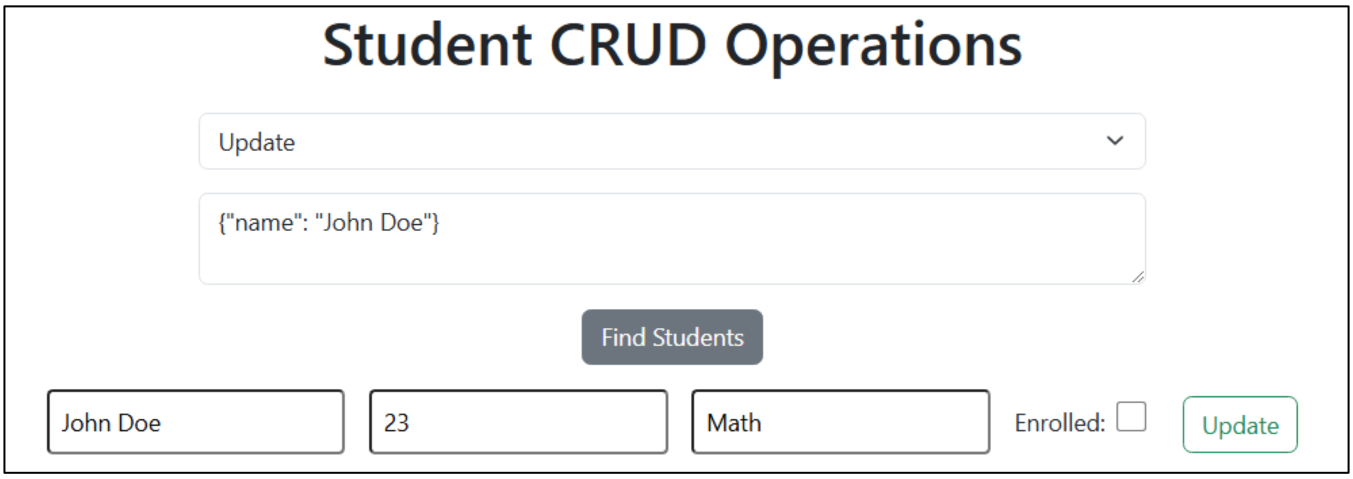


```
app.put('/students/:id', async (req, res) => {  SkalapEnder *
    try {
        const updatedStudent = await collection.findOneAndUpdate(
            { id: parseInt(req.params.id, 10) },
            { $set: req.body },
            { returnDocument: 'after' }
        );
        if (updatedStudent === null) return res.status(404).send('Student not found');
        res.status(200).send(updatedStudent);
    } catch (err) {
        res.status(400).send(err);
    }
});
```

Query: db.students.findOneAndUpdate({id to find}, {$set: new data}, {returnDocument: ‘after’})

Update student task (John Doe)

Original data



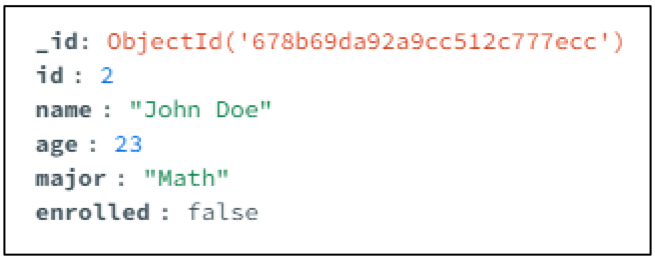
Student CRUD Operations

Update

{"name": "John Doe"}

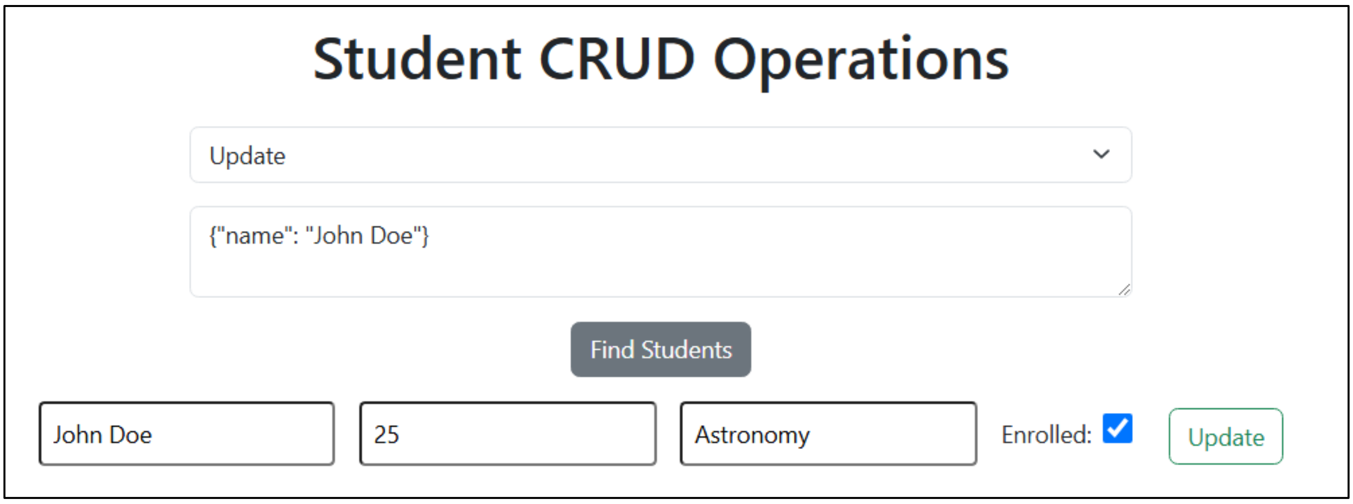
Find Students

John Doe	23	Math	Enrolled: ☐	Update



```
_id: ObjectId('678b69da92a9cc512c777ecc')
id : 2
name : "John Doe"
age : 23
major : "Math"
enrolled : false
```

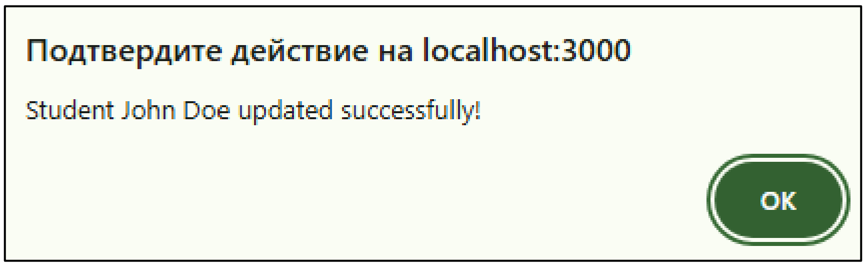
Modified data



Student CRUD Operations

{"name": "John Doe"}

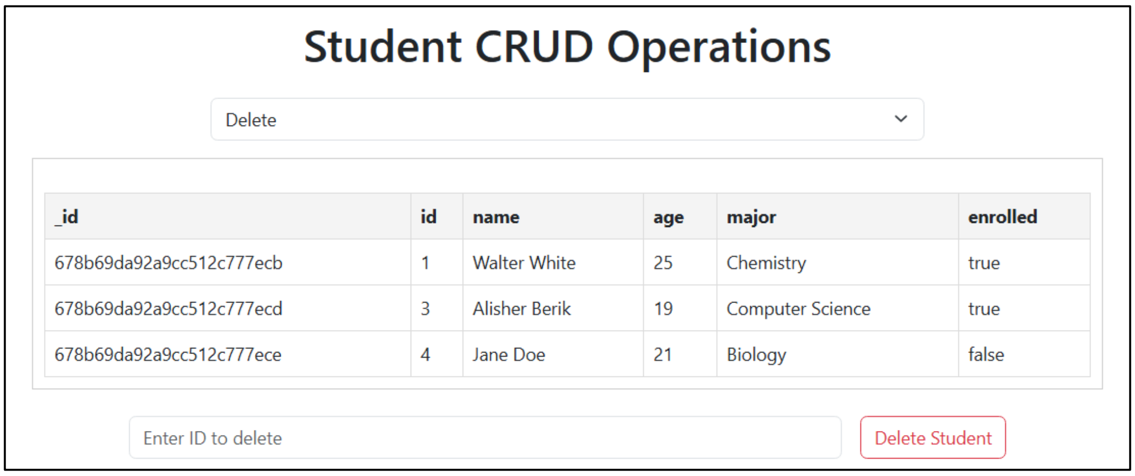
John Doe	25	Astronomy	Enrolled: ✓



Подтвердите действие на localhost:3000

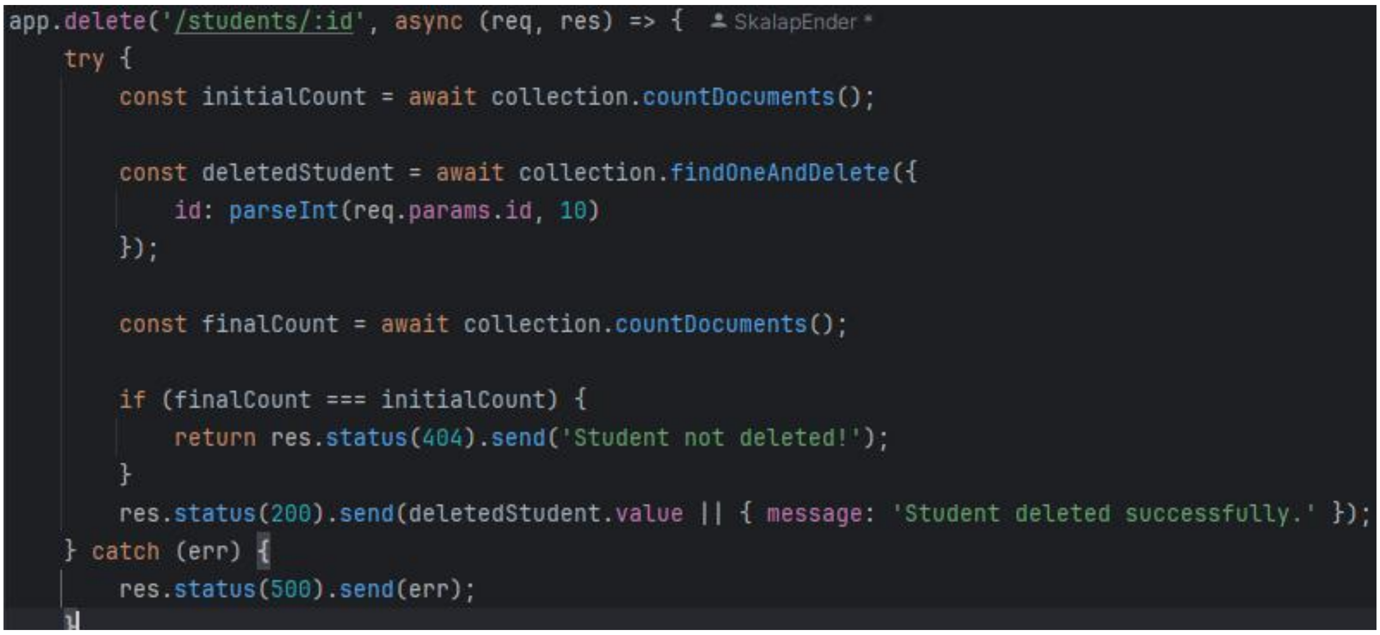
Student John Doe updated successfully!

Task D (Delete)



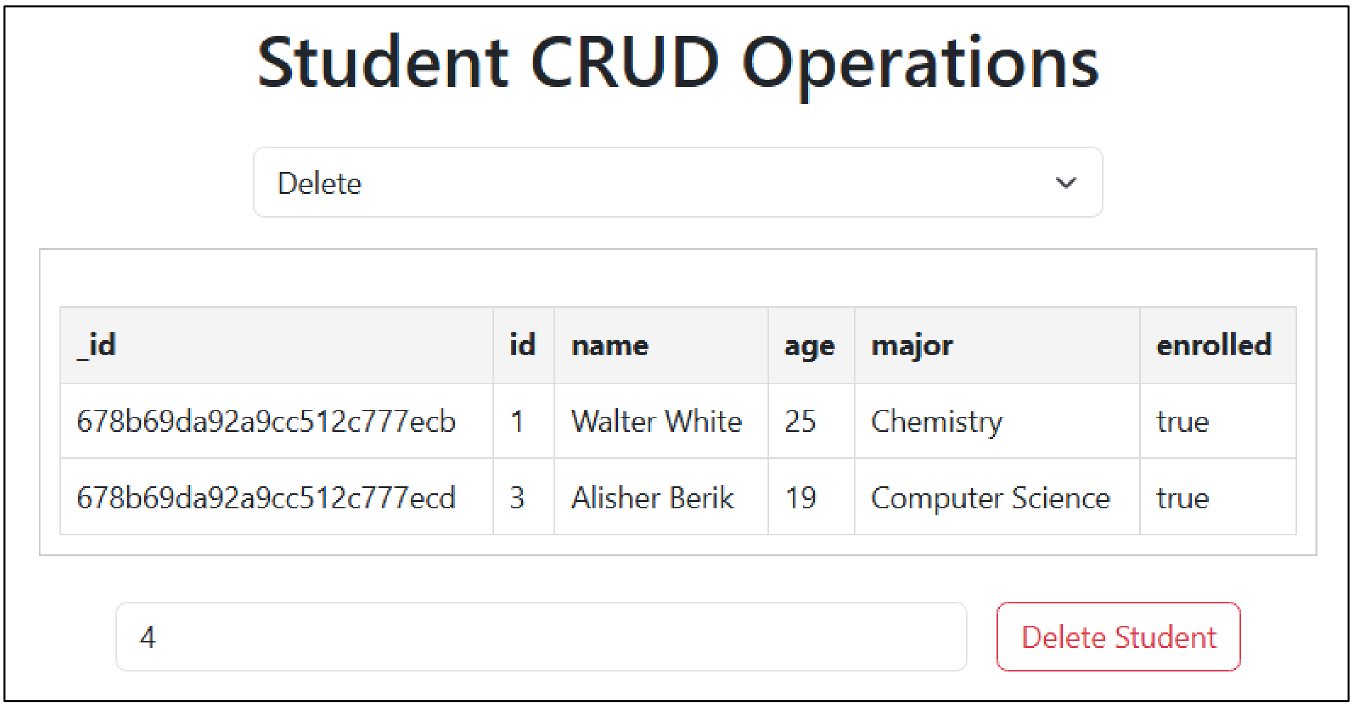
Student CRUD Operations

_id	id	name	age	major	enrolled
678b69da92a9cc512c777ecb	1	Walter White	25	Chemistry	true
678b69da92a9cc512c777ecd	3	Alisher Berik	19	Computer Science	true
678b69da92a9cc512c777ece	4	Jane Doe	21	Biology	false



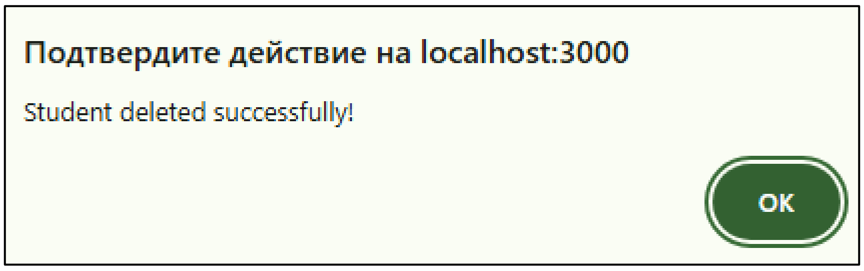
```
app.delete('/students/:id', async (req, res) => {  SkalapEnder *
    try {
        const initialCount = await collection.countDocuments();

        const deletedStudent = await collection.findOneAndDelete({
            id: parseInt(req.params.id, 10)
        });

        const finalCount = await collection.countDocuments();

        if (finalCount === initialCount) {
            return res.status(404).send('Student not deleted!');
        }
        res.status(200).send(deletedStudent.value || { message: 'Student deleted successfully.' });
    } catch (err) {
        res.status(500).send(err);
    }
```

Delete student (ID: 4)



_id	id	name	age	major	enrolled
678b69da92a9cc512c777ecb	1	Walter White	25	Chemistry	true
678b69da92a9cc512c777ecd	3	Alisher Berik	19	Computer Science	true

4



Подтвердите действие на localhost:3000

Student deleted successfully!

Query: db.students.findOneAndDelete({id: id});