0.---------------------
Having the following application developed using NodeJS, complete the `POST` method on path `/ships` :

- If no request body is present you should return a json with the following format: `{"message": "body is missing"}`. Response status code should be: `400`; (0.5 pts)
- If the request body properties are missing you should return a json with the following format: `{"message": "malformed request"}`. Response status code should be: `400`; (0.5 pts)
- Displacement should be over 1000, otherwise the server should return a message with the following format: `{"message": "displacement should be over 1000"}`. Response status code should be: `400`; (0.5 pts)
- If the ship is valid, it should be added and a response with the code `201` should be returned. The response body should be `{"message": "created"}`;(0.5 pts)
- If a `GET /ships` request is performed subsequently, the body should contain 11 `ships`, including the one previoulsy added; (0.5 pts)

```
const express = require('express')
const bodyParser = require('body-parser')
const Sequelize = require('sequelize')

const mysql = require('mysql2/promise')

const DB_USERNAME = 'DB_USER'
const DB_PASSWORD = 'DB_PASS'

let conn

mysql.createConnection({
    user: DB_USERNAME,
    password: DB_PASSWORD
})
    .then((connection) => {
        conn = connection
        return connection.query('CREATE DATABASE IF NOT EXISTS tw_exam')
    })
    .then(() => {
        return conn.end()
    })
    .catch((err) => {
        console.warn(err.stack)
    })

const sequelize = new Sequelize('tw_exam', DB_USERNAME, DB_PASSWORD, {
    dialect: 'mysql',
    logging: false
})

let Ship = sequelize.define('student', {
    name: Sequelize.STRING,
    portOfSail: Sequelize.STRING,
    displacement: Sequelize.INTEGER
}, {
    timestamps: false
})


const app = express()
app.use(bodyParser.json())
app.get('/create', async (req, res) => {
    try {
        await sequelize.sync({ force: true })
        for (let i = 0; i < 10; i++) {
            let ship = new Ship({
                name: `name${i}`,
                portOfSail: `port ${i}`,
                displacement: 3000 + 10 * i
            })
            await ship.save()
        }
        res.status(201).json({ message: 'created' })
    }
    catch (err) {
        console.warn(err.stack)
        res.status(500).json({ message: 'server error' })
    }
})

app.get('/ships', async (req, res) => {

    try {
        let ships = await Ship.findAll()
        res.status(200).json(ships)
    }
    catch (err) {
        console.warn(err.stack)
        res.status(500).json({ message: 'server error' })
    }
})

app.post('/ships', async (req, res) => {
    if (Object.keys(req.body).length===0) {
        res.status(400).json({ message: "body is missing" })
        return
    }
    if (!req.body.name || !req.body.portOfSail || !req.body.displacement) {
        res.status(400).json({ message: "malformed request" })
        return
    }

    if (req.body.displacement < 1000) {
        res.status(400).json({ message: "displacement should be over 1000" })
        return
    }

    let sh=new Ship(req.body)

    sh.save()

    res.status(201).json({message: "created"})



})

module.exports = app
```

1.------------------
# Having the following application developed using NodeJS, complete the `PUT` and `DELETE` methods on path `/ships/id` :

- If attempting to update a non existent ship the response should be `{"message": "not found"}`. Response status code should be: `404`; (0.5 pts)
- If correctly updating an existent ship the response should be: `{"message": "accepted"}`. Response status code should be: `202`; (0.5 pts)
- A subsequent request for the edited ship should show the modifications. Response status code should be: `200`; (0.5 pts)
- If correctly deleting an existent ship the response should be: `{"message": "accepted"}`. Response status code should be: `202`; (0.5 pts)
- A subsequent request for the deleted ship should return `{"message": "not found"}`. Response status code should be: `404`; (0.5 pts)

```
const express = require('express')
const bodyParser = require('body-parser')
const Sequelize = require('sequelize')

const mysql = require('mysql2/promise')

const DB_USERNAME = 'root'
const DB_PASSWORD = 'pass'

let conn

mysql.createConnection({
    user : DB_USERNAME,
    password : DB_PASSWORD
})
.then((connection) => {
    conn = connection
    return connection.query('CREATE DATABASE IF NOT EXISTS tw_exam')
})
.then(() => {
    return conn.end()
})
.catch((err) => {
    console.warn(err.stack)
})

const sequelize = new Sequelize('tw_exam', DB_USERNAME, DB_PASSWORD,{
    dialect : 'mysql',
    logging: false
})

let Ship = sequelize.define('student', {
    name : Sequelize.STRING,
    portOfSail : Sequelize.STRING,
    displacement : Sequelize.INTEGER
},{
    timestamps : false
})


const app = express()
app.use(bodyParser.json())

app.get('/create', async (req, res) => {
    try{
        await sequelize.sync({force : true})
        for (let i = 0; i < 10; i++){
            let ship = new Ship({
                name : `name${i}`,
                portOfSail : `port ${i}`,
                displacement : 3000 + 10 * i
            })
            await ship.save()
        }
        res.status(201).json({message : 'created'})
    }
    catch(err){
        console.warn(err.stack)
        res.status(500).json({message : 'server error'})
    }
})

app.get('/ships/:id', async (req, res) => {
    try{
        let ship = await Ship.findByPk(req.params.id)
            if (!ship){
                  res.status(404).json({message : 'not found'})
            }
            else{
            res.status(200).json(ship)
        }
    }
    catch(err){
        console.warn(err.stack)
        res.status(500).json({message : 'server error'})
    }
})


app.put('/ships/:id', async (req, res) => {
    Ship.findByPk(req.params.id)
    .then((ship) => {
        if (ship) {
            ship.update(req.body).then((result)=> {
                res.status(202).json({message: "accepted"})
            })
        } else {
            res.status(404).json({message : 'not found'})
        }
    }).catch((err) => {
        console.log(err)
        console.status(500).send('database error')
    })

})

app.delete('/ships/:id', async (req, res) => {


        Ship.findByPk(req.params.id)
        .then((ship) => {
            if (ship) {
                ship.destroy().then((result) => {
                    res.status(202).json({message: "accepted"})
                })
            } else {
                res.status(404).json({message : 'not found'})
            }
        }).catch((err) => {
            console.log(err)

        })


})


module.exports = app


2.------------------
# Having the following application developed using NodeJS, complete the `GET`
method on path `/ships` :
- supported query params are `page` and `pageSize`

- If no page or page size is specified, all ships are returned; (0.5 pts)
- If a page is specified, but no page size, the page size is assumed to be 5 and
the nth page of 5 is returned (0.5 pts)
- If both page and page size are specified, the page-th page of the specified size
is returned (0.5 pts)
- If a malformed page or page size is specified, all ships are returned; (0.5 pts)
- If the specified page is beyond the last available record, an empty array of
pages is returned. (0.5 pts)

const express = require('express')
const bodyParser = require('body-parser')
const Sequelize = require('sequelize')

const mysql = require('mysql2/promise')

const DB_USERNAME = 'root'
const DB_PASSWORD = 'pass'

let conn

mysql.createConnection({
    user : DB_USERNAME,
    password : DB_PASSWORD
})
.then((connection) => {
    conn = connection
    return connection.query('CREATE DATABASE IF NOT EXISTS tw_exam')
})
.then(() => {
    return conn.end()
})
.catch((err) => {
    console.warn(err.stack)
})

const sequelize = new Sequelize('tw_exam', DB_USERNAME, DB_PASSWORD,{
    dialect : 'mysql',
    logging: false
```

```
})

let Ship = sequelize.define('student', {
    name : Sequelize.STRING,
    portOfSail : Sequelize.STRING,
    displacement : Sequelize.INTEGER
},{
    timestamps : false
})


const app = express()
app.use(bodyParser.json())

app.get('/create', async (req, res) => {
    try{
        await sequelize.sync({force : true})
        for (let i = 0; i < 10; i++){
            let ship = new Ship({
                name : `name${i}`,
                portOfSail : `port ${i}`,
                displacement : 3000 + 10 * i
            })
            await ship.save()
        }
        res.status(201).json({message : 'created'})
    }
    catch(err){
        console.warn(err.stack)
        res.status(500).json({message : 'server error'})
    }
})

app.get('/ships', async (req, res) => {

    //1.
    if ((!req.query.pageNo && !req.query.pageSize)) {
        const ships = await Ship.findAll()
        return res.status(200).send(ships)
    }
    //2.
    if (req.query.pageNo && !req.query.pageSize) {
        const ships = await Ship.findAll({
            limit: 5,
            offset: req.query.pageNo * 5
        })
        return res.status(200).send(ships)
    }
    //3.
    if (req.query.pageNo && req.query.pageSize) {

        //4.
        if ((isNaN(req.query.pageNo) || isNaN(req.query.pageSize))) {
            const ships = await Ship.findAll()
            return res.status(200).send(ships)
        }
        //5.
        const ships = await Ship.findAll({
            offset: req.query.pageNo * req.query.pageSize,
            limit: parseInt(req.query.pageSize, 10)
        })
        return res.status(200).send(ships)
    }
})

app.post('/ships', async (req, res) => {
      try{
            let ship = new Ship(req.body)
            await ship.save()
            res.status(201).json({message : 'created'})
      }
      catch(err){
            res.status(500).json({message : 'server error'})
      }
})

module.exports = app

3.----------------
# Having the `app.js` file complete `POST` and `DELETE` methods on paths `/device`
and `/device/:id`:
- `POST /device` returns status code 400 and response `{message: 'bad request'}` if
`body` is null (0.5 pts);
- `POST /device` returns status code 400 and response `{message: 'bad request'}` if
`price < 0` (0.5 pts);
- `POST /device` returns status code 400 and response `{message: 'bad request'}` if
`name contains less than 4 characters` (0.5 pts);
- `POST /device` returns status code 201 and response `{message: "device created"}`
if `body is valid` (0.5 pts);
- `DELETE /device/:id` returns status code 202 and response `{message: "device
deleted"}` if `id` for a device is present in the database (0.5 pts);
const express = require('express')
const bodyParser = require('body-parser')
const Sequelize = require('sequelize')

const mysql = require('mysql2/promise')

const DB_USERNAME = 'root'
const DB_PASSWORD = 'pass'

let conn

mysql.createConnection({
    user : DB_USERNAME,
    password : DB_PASSWORD
})
.then((connection) => {
    conn = connection
    return connection.query('CREATE DATABASE IF NOT EXISTS tw_exam')
})
.then(() => {
    return conn.end()
})
.catch((err) => {
    console.warn(err.stack)
})

const sequelize = new Sequelize('tw_exam', DB_USERNAME, DB_PASSWORD,{
    dialect : 'mysql',
    logging: false
})

let Ship = sequelize.define('student', {
    name : Sequelize.STRING,
    portOfSail : Sequelize.STRING,
    displacement : Sequelize.INTEGER
},{
    timestamps : false
})


const app = express()
app.use(bodyParser.json())

app.get('/create', async (req, res) => {
    try{
        await sequelize.sync({force : true})
        for (let i = 0; i < 10; i++){
            let ship = new Ship({
                name : `name${i}`,
                portOfSail : `port ${i}`,
                displacement : 3000 + 10 * i
            })
            await ship.save()
        }
        res.status(201).json({message : 'created'})
    }
    catch(err){
        console.warn(err.stack)
        res.status(500).json({message : 'server error'})
    }
})

app.get('/ships/:id', async (req, res) => {
    try{
        let ship = await Ship.findByPk(req.params.id)
            if (!ship){
                  res.status(404).json({message : 'not found'})
            }
            else{
            res.status(200).json(ship)
        }
    }
    catch(err){
        console.warn(err.stack)
        res.status(500).json({message : 'server error'})
    }
})


app.put('/ships/:id', async (req, res) => {
    Ship.findByPk(req.params.id)
    .then((ship) => {
        if (ship) {
            ship.update(req.body).then((result)=> {
                res.status(202).json({message: "accepted"})
            })
        } else {
            res.status(404).json({message : 'not found'})
        }
    }).catch((err) => {
        console.log(err)
        console.status(500).send('database error')
    })

})

app.delete('/ships/:id', async (req, res) => {


        Ship.findByPk(req.params.id)
        .then((ship) => {
            if (ship) {
                ship.destroy().then((result) => {
                    res.status(202).json({message: "accepted"})
                })
            } else {
                res.status(404).json({message : 'not found'})
            }
        }).catch((err) => {
            console.log(err)

        })


})


module.exports = app
```

4.-------------------
# Having the following `app` complete the method `GET` at the enpoint `/homeworks` and the method `GET` at the endpoint `/homeworks/id` so that:

- when GET /homeworks is called without any params it will return all entries in the database (0.5p)
- when GET /homeworks is called with the param pass=true it will return all homeworks that are >= 5 (0.5p)
- when GET /homeworks/id is called with a id not present in the database will return status code `404`(0.5p)
- when GET /homeworks/id is called with a valid id will return a json object (0.5p)
- given that the client only accepts `text/plain` GET /homeworks/id will return only the content

```
const express = require('express')
const bodyParser = require('body-parser')
const Sequelize = require('sequelize')

const mysql = require('mysql2/promise')

const DB_USERNAME = 'root'
const DB_PASSWORD = 'pass'

let conn

mysql.createConnection({
    user : DB_USERNAME,
    password : DB_PASSWORD
})
.then((connection) => {
    conn = connection
    return connection.query('CREATE DATABASE IF NOT EXISTS tw_exam')
})
.then(() => {
    return conn.end()
})
.catch((err) => {
    console.warn(err.stack)
})

const sequelize = new Sequelize('tw_exam', DB_USERNAME, DB_PASSWORD,{
    dialect : 'mysql',
    logging: false
})

let Homework = sequelize.define('homework', {
    student : Sequelize.STRING,
    content : Sequelize.STRING,
    grade : Sequelize.INTEGER
},{
    timestamps : false
})


const app = express()
app.use(bodyParser.json())

app.get('/create', async (req, res) => {
    try{
        await sequelize.sync({force : true})
        const grades  = [2, 5, 7, 7, 3, 10, 9, 4, 10, 8]
        for (let i = 0; i < 10; i++){
            let homework = new Homework({
                student : `name${i}`,
                content : `some text here ${i}`,
                grade : grades[i]
            })
            await homework.save()
        }
        res.status(201).json({message : 'created'})
    }
    catch(err){
        console.warn(err.stack)
        res.status(500).json({message : 'server error'})
    }
})

app.get('/homeworks', async (req, res) => {
    try{
        if (Object.keys(req.query).length === 0) {
            const homeworks = await Homework.findAll();
            if (!homeworks) {
                return res.status(404).send({ message: "not found" })
            }
            return res.status(200).send(homeworks)
        }

        if (req.query.pass == 'true') {
            const homeworks = await Homework.findAll({
                where: {
                    grade: {
                        [Sequelize.Op.gte]: 5
                    }
                }
            })
            return res.status(200).send(homeworks)
        }


    }
    catch(err){
        console.warn(err.stack)
        res.status(500).json({message : 'server error'})
    }
})

app.get('/homeworks/:id', async (req, res) => {
    try{
        let hw = await Homework.findByPk(req.params.id)
            if (!hw){
                  return res.status(404).send()
            }
            if (req.header('accept') === 'text/plain') {
            return res.status(200).send(hw.content)
        }
        return res.status(200).send(hw)
    }
    catch(err){
        console.warn(err.stack)
        res.status(500).json({message : 'server error'})
    }
})



module.exports = app


----------------------------------------------------------------------
2021------------------------------------------------------------
#Subject 3 (2.5 pts)
#TOPIC: REST
------------------------------------------------------------------
# Having the following application developed using NodeJS, complete the `GET`
method on path `/ships` :
- supported query params are `page` and `pageSize`

- If no page or page size is specified, all ships are returned; (0.5 pts)
- If a page is specified, but no page size, the page size is assumed to be 5 and
the nth page of 5 is returned (0.5 pts)
- If both page and page size are specified, the page-th page of the specified size
is returned (0.5 pts)
- If a malformed page or page size is specified, all ships are returned; (0.5 pts)
```

- If the specified page is beyond the last available record, an empty array of pages is returned. (0.5 pts)

```
const express = require('express')
const bodyParser = require('body-parser')
const Sequelize = require('sequelize')

const mysql = require('mysql2/promise')

const DB_USERNAME = 'root'
const DB_PASSWORD = 'welcome12#'

let conn

mysql.createConnection({
    user : DB_USERNAME,
    password : DB_PASSWORD
})
.then((connection) => {
    conn = connection
    return connection.query('CREATE DATABASE IF NOT EXISTS tw_exam')
})
.then(() => {
    return conn.end()
})
.catch((err) => {
    console.warn(err.stack)
})

const sequelize = new Sequelize('tw_exam', DB_USERNAME, DB_PASSWORD,{
    dialect : 'mysql',
    logging: false
})

let Ship = sequelize.define('ship', {
    name : Sequelize.STRING,
    portOfSail : Sequelize.STRING,
    displacement : Sequelize.INTEGER
},{
    timestamps : false
})


const app = express()

app.get('/create', async (req, res) => {
    try{
        await sequelize.sync({force : true})
        for (let i = 0; i < 10; i++){
            let ship = new Ship({
                name : `name${i}`,
                portOfSail : `port ${i}`,
                displacement : 3000 + 10 * i
            })
            await ship.save()
        }
        res.status(201).json({message : 'created'})
    }
    catch(err){
        console.warn(err.stack)
        res.status(500).json({message : 'server error'})
    }
})

app.get('/ships', async (req, res) => {

})

app.post('/ships', async (req, res) => {
      try{
            let ship = new Ship(req.body)
            await ship.save()
            res.status(201).json({message : 'created'})
      }
      catch(err){
            res.status(500).json({message : 'server error'})
      }
})

module.exports = app
```

# Having the following application developed using NodeJS, complete the `POST` method on path `/students` :

- If no request body is present you should return a json with the following format: `{message: "Body is missing"}`. Response status code should be: `500`;
- If the request body properties are missing you should return a json with the following format: `{message: "Invlid body format"}`. Response status code should be: `500`;
- Student age should be positive, otherwise return a json with the following format: `{message: "Age should be a positive number"}`. Response status code should be: `500`;
- If the student already exists in the array. Return a json with the following format: `{message: "Student already exists"}`.Response status code should be: `500`. Checking is done by `name`;
- If the request body is good, student should be added in the array and return a json with the following format: `{message: "Created"}`.Response status code should be: `201`;

# Having the following application developed using NodeJS, complete the `POST` method on path `/products` :
 ---> 4/5
<--

- If no request body is present you should return a json with the following format: `{message: "Body is missing"}`. Response status code should be: `500`;
- If the request body properties are missing you should return a json with the following format: `{message: "Invlid body format"}`. Response status code should be: `500`;
- Product price should be positive, otherwise return a json with the following

```
format: `{message: "Price should be a positive number"}`. Response status code
should be: `500`;
- If the product already exists in the array. Return a json with the following
format: `{message: "Product already exists"}`.Response status code should be:
`500`;
- If the request body is good, product should be added in the array and return a
json with the following format: `{message: "Created"}`.Response status code should
be: `201`;

const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');

const app = express();

app.use(bodyParser.json());
app.use(cors());

app.locals.students = [
    {
        name: "Gigel",
        surname: "Popel",
        age: 23
    },
    {
        name: "Gigescu",
        surname: "Ionel",
        age: 25
    }
];

app.get('/students', (req, res) => {
    res.status(200).json(app.locals.products);
});

app.post('/students', (req, res) => {
  const body = req.body;
  if(Object.keys(body).length === 0) {
    res.status(500).json({message: "Body is missing"});
  }
  else if(!body.hasOwnProperty('name') || !body.hasOwnProperty('surname') || !
body.hasOwnProperty('age')) {
    res.status(500).json({message: "Invalid body format"});
  }
  //Student age should be positive, otherwise return a json with the following
format: `{message: "Age should be a positive number"}`. Response status code should
be: `500`;
  else if(body.age < 0) {
    res.status(500).json({message: "Age should be a positive number"});
  }
  // If the student already exists in the array. Return a json with the following
format: `{message: "Student already exists"}`.Response status code should be:
`500`. Checking is done by `name`;
  else if(app.locals.students.some(student => student.name === body.name)) {
    res.status(500).json({message: "Student already exists"});
  }
  //If the body is valid the student should be added to the array
  else {
    app.locals.students.push(body);
    res.status(201).json({message: "Created"});

  }

  });
module.exports = app;
```

```
-----------------------------------------------------------------------------------
------------------------------------------------------------
```

# Having the following application developed using NodeJS, complete the `POST` method on path `/students` :

- If no request body is present you should return a json with the following format: `{message: "Body is missing"}`. Response status code should be: `500`;
- If the request body properties are missing you should return a json with the following format: `{message: "Invlid body format"}`. Response status code should be: `500`;
- Student age should be positive, otherwise return a json with the following format: `{message: "Age should be a positive number"}`. Response status code should be: `500`;
- If the student already exists in the array. Return a json with the following format: `{message: "Student already exists"}`.Response status code should be: `500`. Checking is done by `name`;
- If the request body is good, student should be added in the array and return a json with the following format: `{message: "Created"}`.Response status code should be: `201`;

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');

const app = express();

app.use(bodyParser.json());
app.use(cors());

app.locals.students = [
    {
        name: "Gigel",
        surname: "Popel",
        age: 23
    },
    {
        name: "Gigescu",
        surname: "Ionel",
        age: 25
    }
];

app.get('/students', (req, res) => {
    res.status(200).json(app.locals.products);
});

app.post('/students', (req, res) => {
    if (!req.body) {
      return res.status(500).json({ message: 'Body is missing' });
    }

    const { name, age } = req.body;
    if (!name || !age) {
      return res.status(500).json({ message: 'Invalid body format' });
    }

    if (age < 0) {
      return res.status(500).json({ message: 'Age should be a positive number' });
    }

    // Check if student already exists in the array
    const students = [/* array of students */];
    const studentExists = students.some(student => student.name === name);
    if (studentExists) {
      return res.status(500).json({ message: 'Student already exists' });
    }

    // Add the student to the array
    students.push({ name, age });

    return res.status(201).json({ message: 'Created' });
  });
module.exports = app;
```


# Having the following application developed using NodeJS, complete the `GET` method on path `/ships` :

- supported query params are `page` and `pageSize`

- If no page or page size is specified, all ships are returned; (0.5 pts)
- If a page is specified, but no page size, the page size is assumed to be 5 and the nth page of 5 is returned (0.5 pts)
- If both page and page size are specified, the page-th page of the specified size is returned (0.5 pts)
- If a malformed page or page size is specified, all ships are returned; (0.5 pts)
- If the specified page is beyond the last available record, an empty array of pages is returned. (0.5 pts)

```
app.get('/ships', async (req, res) => {
     //1.
     if ((!req.query.pageNo && !req.query.pageSize)) {
        const ships = await Ship.findAll()
        return res.status(200).send(ships)
    }
    //2.
    if (req.query.pageNo && !req.query.pageSize) {
        const ships = await Ship.findAll({
            limit: 5,
            offset: req.query.pageNo * 5
        })
        return res.status(200).send(ships)
    }
    //3.
    if (req.query.pageNo && req.query.pageSize) {
        //4.
        if ((isNaN(req.query.pageNo) || isNaN(req.query.pageSize))) {
            const ships = await Ship.findAll()
            return res.status(200).send(ships)
        }
        //5.
        const ships = await Ship.findAll({
            offset: req.query.pageNo * req.query.pageSize,
            limit: parseInt(req.query.pageSize, 10)
        })
        return res.status(200).send(ships)
    }

})

------------------------------------------------------------------------------------
--------------------------------------------------
#Subiect 3 (2.5 pts)
#TOPIC: REST

# Having the `app.js` file complete `POST` and `DELETE` methods on paths `/device`
and `/device/:id`:
- `POST /device` returns status code 400 and response `{message: 'bad request'}` if
`body` is null (0.5 pts);
- `POST /device` returns status code 400 and response `{message: 'bad request'}` if
`price < 0` (0.5 pts);
- `POST /device` returns status code 400 and response `{message: 'bad request'}` if
`name contains less than 4 characters` (0.5 pts);
- `POST /device` returns status code 201 and response `{message: "device created"}`
if `body is valid` (0.5 pts);
- `DELETE /device/:id` returns status code 202 and response `{message: "device
deleted"}` if `id` for a device is present in the database (0.5 pts);


app.post('/device', async (req, res) => {
    try {
      // Validate the request body
      if (!req.body) {
        return res.status(400).json({ message: 'bad request' });
      }
      if (req.body.price < 0) {
        return res.status(400).json({ message: 'bad request' });
      }
      if (req.body.name.length < 4) {
        return res.status(400).json({ message: 'bad request' });
      }

      // Create the device
      const device = await Device.create(req.body);
      res.status(201).json({ message: 'device created' });
    } catch (err) {
      res.status(500).json({ message: 'server error' });
    }
  });
```