НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра програмного забезпечення комп'ютерних систем

ЛАБОРАТОРНА РОБОТА №4

з дисципліни "Основи web-програмування"

тема *"Публікація веб-додатку"*

Виконав: Петренко Нікіта Андрійович
Група: КП-93

1 семестр 2020/2021

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра програмного забезпечення комп'ютерних систем

Мета роботи

Вивчити основні принципи асинхронного програмування в JavaScript. Навчитись асинхронно взаємодіяти з базою даних. Налаштувати взаємодію з віддаленою базою даних та сховищем медіа. Підготувати і опублікувати веб-сервіс в мережі Інтернет.

Завдання

База даних і асинхронне програмування:

1. Змоделювати за допомогою GUI клієнта бази даних об'єкти сутностей із
попередніх лабораторних робіт та додати нову композиційну сутність ({composite}) відповідно до варіанту (див. Додаток ""Структура бази даних"").
Називати таблиці\колекції сутностей у множині англійською мовою.
Всі сутності мають містити унікальні ключі-ідентифікатори.

2. Переписати модуль сховища даних для взаємодії з базою даних. Змінити
інтерфейс сховища для використання асинхронних функцій. Весь доступ до бази
даних має бути асинхронним.

3. Композиційна сутність
i. Створити нову модель {Composite} у модулі models/{composite}.js для композитних сутностей (за варіантом, замість {Composite} використовувати
назву сутності англійською мовою). Реалізувати асинхронні CRUD операції
цих сутностей для взаємодії із базою даних.
ii. Додати відповідні веб-сторінки (або Swagger документацію), що дозволяють
керувати композиційними сутностями.

4. Видалити з директорії /data всі JSON файли з даними, які тепер будуть
розміщені у БД.

5. Створити у проекті файл конфігурації (config.js) і внести у нього шлях
підключення до бази даних. Отримувати шлях зі змінних середовища. Для

цього
встановити модуль env та створити файл env , який обов'язково додати у .gitignore .

Віддалена база даних:

1. Створити Heroku Application для вашого веб-сайту.
2. Створити віддалений екземпляр бази даних (наприклад, на mLab).
3. Налаштувати ваш Heroku Application для доступу до віддаленої БД. Для цього в налаштуванні додатку додайте змінну середовища, що міститиме посилання на віддалену БД.
4. Додати в рішення цієї роботи посилання на створений веб-сайт на Heroku.

Сховище медіа

1. Переписати модуль медіа сховища. Підключити пакет cloudinary та завантажувати всі файли, що прийшли у запиті з форми туди (див. Додатки). API Cloudinary буде віддавати URL на завантажений файл, який і зберігати у полях сутностей.
2. Налаштувати ваш Heroku Application для доступу до віддаленого сховища медіа. Для цього в налаштуванні додатку додайте змінну середовища, що міститиме посилання на сховище.
3. Видалити з проекту директорію, у які зберігались динамічні медіа

Код нових та змінених модулів.

app.js

```
const express = require('express');
const consolidate = require('consolidate');
const mustache_ex = require('mustache-express');
const path = require('path');
require('dotenv').config();
const app = express();
const mongoose = require('mongoose');
const dbPlanet = require('./models/planets')
const dbUser = require('./models/users')
const dbRace = require('./models/race')

const connectOptions = {
    useNewUrlParser: true,
    useUnifiedTopology: true,
};
const morgan = require('morgan');
const body_parser = require('body-parser');
const busboy_body_parser = require('busboy-body-parser');
const mstRouter = require('./routes/route_mst');
const { mustache } = require('consolidate');

const PORT = Number(process.env.PORT) || 3000



app.use(body_parser.urlencoded({ extended: true }))
app.use(body_parser.json());
app.use(busboy_body_parser());

app.use((err, req, res, next) => {
    if (err instanceof SyntaxError && err.status === 400 && 'body'
in err) {
        res.status(400).send({ mess: "Bad request" });
        return;
    }

    next();
});



const viewsDir = path.join(__dirname, 'views');
app.engine("mst", mustache_ex(path.join(viewsDir, "partials")));
app.set('views', viewsDir);
app.set('view engine', 'mst');
// usage
app.get('/', function(req, res) {
    res.render('index', { index_current: 'current', home_link:
'disabled_link' });
});


app.get('/about', function(req, res) {
```

```
    res.render('about', { about_current: 'current', about_link:
'disabled_link' });
});


app.use('', mstRouter);
app.use(express.static("./public"));
app.use(express.static("./data"));
app.use(morgan('dev'));



async function appStart() {
    try {
        console.log('Connection to database...')

        await mongoose.connect(process.env.MONGO_URL, {
            useNewUrlParser: true,
            useCreateIndex: true,
            useUnifiedTopology: true,
        })

        console.log('App has successfully connected to the
database...')

        app.listen(PORT, () => {
            console.log(`App is listening on port ${PORT}...`)
        })
    } catch (err) {
        console.log('Database connection error: ', err.message)
        process.exit(1)
    }
}

appStart()
```

planets_controller.js

```
const path = require('path');
const { nextTick } = require('process');
const planetRepository =
require('../repositories/planetsRepository');
const planetRepo = new planetRepository()
const Planet = require('../models/planets');
const fs = require('fs');

const RaceRepository = require('../repositories/raceRepository')
const raceRepo = new RaceRepository();

require('dotenv').config();

const cloudinary = require('cloudinary').v2;
cloudinary.config({
    cloud_name: process.env.CLOUD_NAME,
    api_key: process.env.CLOUD_API_KEY,
```

```
    api_secret: process.env.CLOUD_API_SECRET
});

function pagination(items, page, per_page) {
    const startInd = (page - 1) * per_page;
    const endInd = page * per_page;
    return (items.slice(startInd, endInd));
}
async function uploadRaw(buffer) {
    return new Promise((resolve, reject) => {
        cloudinary.uploader
            .upload_stream({ resource_type: 'raw' },
                (err, result) => {
                    if (err) {
                        reject(err);
                    } else {
                        resolve(result);
                    }
                })
            .end(buffer);
    });
}

module.exports = {
    async getPlanets(req, res) {
        let page;

        if (req.query.page && isNaN(req.query.page)) {
            res.status(400).send({ mess: 'page and per_page should
be a number' });
            return;
        }

        if (!req.query.page) {
            page = 1;
        } else {
            page = parseInt(req.query.page);
        }

        const per_page = 4;

        let Planets;
        let name = "";
        try {
            if (req.query.name) {
                Planets = await
planetRepo.getPlanetByName(req.query.name);
                name = req.query.name;
            } else {
                Planets = await planetRepo.getPlanets();
            }

            const Pag_planets = pagination(Planets, page, per_page);
            let class_prev;
            let class_next;
```

```
            if (page > 1) {
                class_prev = "";
            } else {
                class_prev = "disabled_link";
            }

            if (page * per_page < Planets.length) {
                class_next = "";
            } else {
                class_next = "disabled_link";
            }

            let max_page = Math.ceil(parseInt(Planets.length) /
(per_page));
            if (max_page == 0) {
                max_page = 1;
            }

            res.render('planets', {
                planets: Pag_planets, search_name: name,
planet_current: 'current', next_page: page + 1, previous_page: page
- 1, page, class_prev, class_next,
                max_page, planet_link: 'disabled_link'
            });
        } catch (err) {
            console.log(err)
            res.status(404).send({ error: 'Cannot get planets' })

        }
    },

    async getPlanetById(req, res) {
        try {
            const Planet = await
planetRepo.getPlanetById(req.params.id);

        res.render('planet', { planet: Planet, races:
Planet.race_id, planet_current: 'current' });
    } catch (err) {
        console.log(err)
        res.status(404).send({ error: 'Cannot get planet by id' })
    }
    },

    newPlanet(req, res) {
        res.render('add_planet');
    },



    async postPlanet(req, res) {
        let time = new Date();
        const name = req.body.name;
        const discoverer = req.body.discoverer;
```

```
        const sat = req.body.sat;
        const mass = req.body.mass;
        let media = "";
        if (req.files['photo']) {
            try {
                const o = await uploadRaw(req.files['photo'].data)
                media_path = o.url;
            } catch (err) {
                res.status(500).send({ error: 'Cannot upload photo'
})
            }
        }

        const planet = { discoverer: discoverer, name: name, sat:
sat, timeDownload: time.toISOString(), mass: mass, media_path:
media_path };
        try {
            const id = await planetRepo.addPlanet(planet)

            res.redirect('/planets/' + id);
        } catch (err) {
            res.status(500).send({ error: 'Cannot add planet' })
        }
    },


    async deletePlanet(req, res) {
        planetRepo.deletePlanet(req.params.id);
        res.redirect('/planets')
    },

    async getupdatePlanet(req, res) {
        try {
            const Planet = await
planetRepo.getPlanetById(req.params.id);

            res.render('update_planet', { planet: Planet, id:
req.params.id });
        } catch (err) {
            res.status(404).send({ error: 'Cannot get planet by id'
})
        }
    },
    async updatePlanet(req, res) {
        const discoverer = req.body.discoverer;
        const name = req.body.name;
        const sat = req.body.sat;
        const mass = req.body.mass;
        const planet = { discoverer: discoverer, name: name, sat:
sat, mass: mass };

        planetRepo.updatePlanet(req.params.id, planet);
        res.redirect('/planets/' + req.params.id);
    },
    async getupdatePlanetPhoto(req, res) {
```

```
        res.render('update_planet_photo', { id: req.params.id });
    },
    async updatePlanetPhoto(req, res) {
        let media_path = "";
        try {
            if (req.files['photo']) {

                const o = await uploadRaw(req.files['photo'].data)
                media_path = o.url;

            }

        } catch (err) {
            res.status(500).send({ error: 'Cannot upload photo' })
        }
        planetRepo.updatePlanetPhoto(req.params.id, media_path);
        res.redirect('/planets/' + req.params.id);
    },
    async addRace(req, res) {
        try {
            Races = await raceRepo.getRaces();
        } catch (err) {
            res.status(404).send({ error: 'Cannot get races' })
        }
        res.render('add_race_to_planet', { planet_id: req.params.id,
races: Races });
    },
    async postAddRace(req, res) {
        const race_id = req.body.races;
        const planet_id = req.params.id;
        await planetRepo.addRaceToList(planet_id, race_id);
        res.redirect('/planets/' + planet_id);
    },
    async deleteRace(req, res) {
        try {
            const Planet = await
planetRepo.getPlanetById(req.params.id);

        res.render('delete_race_from_planet', { planet_id:
req.params.id, races: Planet.race_id });
    } catch (err) {
            res.status(404).send({ error: 'Cannot get planet by id'
})
        }
    },
    async postDelRace(req, res) {
        const race_id = req.body.races;
        const planet_id = req.params.id;
        await planetRepo.deleteRaceFromList(planet_id, race_id);
        res.redirect('/planets/' + planet_id);
    }
}
```

race_controller.js

```
const path = require('path');
const { nextTick } = require('process');
const raceRepository = require('../repositories/raceRepository');
const raceRepo = new raceRepository();
const Race = require ('../models/race');
const fs = require('fs');
require('dotenv').config();

const cloudinary = require('cloudinary').v2;
cloudinary.config({
    cloud_name: process.env.CLOUD_NAME,
    api_key: process.env.CLOUD_API_KEY,
    api_secret: process.env.CLOUD_API_SECRET
});

function pagination(items, page, per_page) {
    const startInd = (page - 1) * per_page;
    const endInd = page * per_page;
    return (items.slice(startInd, endInd));
}


async function uploadRaw(buffer) {
    return new Promise((resolve, reject) => {
        cloudinary.uploader
            .upload_stream({ resource_type: 'raw' },
                (err, result) => {
                    if (err) {
                        reject(err);
                    } else {
                        resolve(result);
                    }
                })
            .end(buffer);
    });
}

module.exports = {
    async getRraces(req, res) {
        let page;

        if (req.query.page && isNaN(req.query.page)) {
            res.status(400).send({ mess: 'page and per_page should
be a number' });
            return;
        }

        if (!req.query.page) {
            page = 1;
        } else {
            page = parseInt(req.query.page);
        }

        const per_page = 4;
```

```
        let Races;

        let name = "";
        try {
            if (req.query.name) {
                Races = await
raceRepo.getRaceByName(req.query.name);
                name = req.query.name;
            } else {
                Races = await raceRepo.getRaces();
            }
        const Pag_races = pagination(Races, page, per_page);

        let class_prev;
        let class_next;

        if (page > 1) {
            class_prev = "";
        } else {
            class_prev = "disabled_link";
        }

        if (page * per_page < Races.length) {
            class_next = "";
        } else {
            class_next = "disabled_link";
        }

        let max_page = Math.ceil(parseInt(Races.length) /
(per_page));
        if (max_page == 0) {
            max_page = 1;
        }
        res.render('races', {races: Pag_races, search_name: name,
race_current: 'current', next_page: page + 1, previous_page: page -
1, page, class_prev, class_next, max_page, race_link:
'disabled_link' });
    } catch (err) {
        res.status(404).send({ error: 'Cannot get races' })
    }
    },

    async getRaceById(req, res) {
        try {
            const Race = await raceRepo.getRaceById(req.params.id);

        res.render('race', { race: Race, race_current: 'current' });
    } catch (err) {
        res.status(404).send({ error: 'Cannot get race by id' })
    }
    },

    newRace(req, res) {
        res.render('add_race');
```

```
    },

    async postRace(req, res) {
        const name = req.body.name;
        const strength = req.body.strength;
        const intellect = req.body.intellect
        const dexterity = req.body.dexterity;
        let media_path = "";
        try {
        if (req.files['photo']) {


                const o = await uploadRaw(req.files['photo'].data)
                media_path = o.url;
            }
        } catch (err) {
                res.status(500).send({ error: 'Cannot upload
11photo' })
            }
        const race = {
            name: name,
            strength: strength,
            intellect: intellect,
            dexterity: dexterity,
            media_path: media_path
        };
        try {
            const id = await raceRepo.addRace(race);

        res.redirect('/races/' + id);}
        catch (err) {
            res.status(404).send({ error: 'Cannot get races by id'
})
        }
    },


    async deleteRace(req, res) {
        raceRepo.deleteRace(req.params.id);
        res.redirect('/races');
    },
    async getupdateRace(req, res) {
        try {
            const Race = await raceRepo.getRaceById(req.params.id);
        } catch (err) {
            res.status(404).send({ error: 'Cannot get races by id'
})
        }
        res.render('update_race', { race: Race, id: req.params.id});
    },
    async updateRace(req, res) {
        const name = req.body.name;
        const strength = req.body.strength;
        const intellect = req.body.intellect;
        const dexterity = req.body.dexterity;
```

```
        const Race = {
            name: name,
            strength: strength,
            intellect: intellect,
            dexterity: dexterity
        };
        raceRepo.updateRace(req.params.id, Race);
        res.redirect('/races/' + req.params.id);
    },
    async getupdateRacePhoto(req, res) {
        res.render('update_race_photo', { id: req.params.id });
    },
    async updateRacePhoto(req, res) {
        let media_path = "";
        try {
        if (req.files['photo']) {
                const o = await uploadRaw(req.files['photo'].data)
                media_path = o.url;

        }
    } catch (err) {
        res.status(500).send({ error: 'Cannot upload photo' })
    }
    try{
        const updateinfo = await
raceRepo.updateRacePhoto(req.params.id, media_path);
        res.redirect('/races/' + req.params.id);
    } catch(err){
        res.status(500).send({ error: 'Cannot upload phot43o' })
        }
}
}
```

user_controller.js

```
const path = require('path');
const User = require('../models/users');
const userRepository = require ('../repositories/userRepository');
const userRepo = new userRepository();

const planetRepository = require(path.resolve(__dirname,
'../repositories/planetsRepository'))
const planetRepo = new planetRepository()

function pagination(items, page, per_page) {
    const startInd = (page - 1) * per_page;
    const endInd = page * per_page;
    return (items.slice(startInd, endInd));

}



function ToUserFriendlyDate(Users) {
    Users.forEach(element => {
```

```
        date = new Date(element.registeredAt);
        element.registeredAt = date.toUTCString().replace('GMT',
'');
    });
    return Users;

}

module.exports = {
   async getUsers(req, res) {
        let page, per_page;
        if (req.query.per_page && isNaN(req.query.per_page)) {
            res.status(400).send({ mess: 'per_page and page should
be a number' });
            return;
        }

        if (req.query.page && isNaN(req.query.page)) {
            res.status(400).send({ mess: 'page and per_page should
be a number' });
            return;
        }
        if (!req.query.per_page) {
            per_page = 4;
        } else if (parseInt(req.query.per_page) > 10 ||
parseInt(req.query.per_page) < 1) {
            res.status(400).send({ mess: 'per_page should`n be more
than 10 and less then 1' });
            return;
        } else {
            per_page = parseInt(req.query.per_page);
        }

        if (!req.query.page) {
            page = 1;
        } else if (parseInt(req.query.page) < 0) {
            res.status(400).send({ mess: 'page should`n be less then
0' });
            return;
        } else {
            page = parseInt(req.query.page);
        }
        try{
            const Users = await userRepo.getUsers();
            const Pag_Users = pagination(Users, page, per_page);
        const User_friendly = ToUserFriendlyDate(Pag_Users);
        res.status(200).render('users', { users: User_friendly,
user_current: 'current', user_link: 'disabled_link' });
        }catch(err) {res.status(404).send({ error: 'Cannot get
users'})}


    },
    async getUserById(req, res) {
        try {
```

```
            const User = await userRepo.getUserById(req.params.id);
            date = new Date(User.registeredAt);
            res.status(200).render('user', { user: User, planets:
User.planet_id, registeredAt: date.toUTCString().replace('GMT', ''),
user_current: 'current' });
        } catch (err) {
            res.status(404).send({ error: 'Cannot get planets by id'
})
        }
    },
    async addPlanet(req, res) {
        try {
            Planet = await planetRepo.getPlanets();
        } catch (err) {
            res.status(404).send({ error: 'Cannot get planets' })
        }
        res.render('add_planet_to_user', { user_id: req.params.id,
planets: Planet });
    },
    async postPlanet(req, res) {
        const planet_id = req.body.planets;
        const user_id = req.params.id;
        try {
            await userRepo.addPlanetToList(user_id, planet_id);
        } catch (err) {
            res.status(404).send({ error: 'Cannot add planet to
list' })
        }
        res.redirect('/users/' + user_id);
    },
    async deletePlanet(req, res) {
        try {
            const User = await userRepo.getUserById(req.params.id);
        } catch (err) {
            res.status(404).send({ error: 'Cannot get user by id' })
        }
        res.render('delete_planet_from_user', { user_id:
req.params.id, planets: User.planet_id });
    },
    async postDelPlanet(req, res) {
        const planet_id = req.body.planets;
        const user_id = req.params.id;
        try {
            await userRepo.deletePlanetFromList(user_id, planet_id);
        } catch (err) {
            res.status(404).send({ error: 'Cannot delete planet from
list' })
        }
        res.redirect('/users/' + user_id);
    }
}
```

planets.js

```
const mongoose = require('mongoose')
/**
```

```
 * @typedef Planet
 * @property {integer} id
 * @property {string} discoverer.required
 * @property {string} name.required
 * @property {integer} sat.required
 * @property {integer} mass.required
 * @property {string} timeDownload
*/

const PlanetSchema = new mongoose.Schema({
    discoverer: {type: String},
    name: {type: String},
    sat: {type: Number},
    mass: {type: Number},
    timeDownload: { type: Date, default: Date.now },
    media_path: {type: String},
    race_id: [{type: mongoose.mongo.ObjectId, ref: 'races'}]
});

module.exports = mongoose.model('planets', PlanetSchema)
```

race.js

```
const mongoose = require('mongoose')

const RaceSchema = new mongoose.Schema({
    name: {type: String},
    strength: {type: Number},
    intellect: {type: Number},
    dexterity: {type: Number},
    media_path: { type: String }
});
module.exports = mongoose.model('races', RaceSchema);
```

user.js

```
const mongoose = require('mongoose')
/**
 * @typedef User
 * @property {integer} id.required
 * @property {string} login.
 * @property {string} fullname.
 * @property {integer} role
 * @property {string} registeredAt
 * @property {string} avaUrl
 * @property {integer} isEnabled
 */

const UserSchema = new mongoose.Schema({
    login: { type: String },
    fullname: { type: String },
    role: { type: Boolean },
```

```
    registeredAt: { type: Date, default: Date.now },
    avaUrl: { type: String },
    isEnabled: { type: Boolean },
    Bio: { type: String },
    planet_id: [{ type: mongoose.mongo.ObjectId, ref: "planets" }]
});

module.exports = mongoose.model('Users', UserSchema);
```

raceRepository.js

```
const RaceModel = require('../models/race');
const PlanetModel = require('../models/planets');
// const fs = require('fs');
class RaceRepository {

    constructor() {

    }
    async addRace(race) {
        const result = await new RaceModel(race).save();
        return result.toJSON()._id;
    }

    async getRaces() {
        const raceDocs = await RaceModel.find({});
        const raceArr = raceDocs.map(i => i.toJSON());
        return raceArr;
    }

    async getRaceByName(name) {
        let items = await this.getRaces();
        items = items.filter(item => item.name.includes(name));
        return items;
    }

    async getRaceById(id) {
        const race = await RaceModel.findOne({ _id: id });
        return race;
    }

    async updateRace(race_id, race) {
        const updateInfo = await RaceModel.updateOne({ _id: race_id
}, {
            $set: {
                name: race.name,
                strength: race.strength,
```

```
                intellect: race.intellect,
                dexterity: race.dexterity
            }
        });
    }
    async updateRacePhoto(planet_id, media_path) {
        const updateInfo = await RaceModel.updateOne({ _id:
planet_id }, { $set: { media_path: media_path } });
        return updateInfo;
    }

    async deleteRace(race_id) {
        const updateInfo = await PlanetModel.updateMany({ race_id:
race_id }, { '$pull': { 'race_id': race_id } });
        const result = await RaceModel.deleteOne({ _id: race_id });
        return result;
    }
};

module.exports = RaceRepository;
```

planetsRepository.js

```
const PlanetModel = require('../models/planets');
const UserModel = require('../models/users');

// const fs = require('fs');
class PlanetRepository {

    constructor() {

    }
    async addPlanet(planet) {

        const result = await new PlanetModel(planet).save();
        return result.toJSON()._id;

    }

    async getPlanets() {

        const planetDocs = await PlanetModel.find({});
        const planetArr = planetDocs.map(i => i.toJSON());
        console.log(planetArr)
        return planetArr;

    }

    async getPlanetByName(name) {

        let items = await this.getPlanets();
        items = items.filter(item => item.name.includes(name));

        return items;

    }
```

```
    async getPlanetById(id) {

        const planet = await PlanetModel.findOne({ _id: id
}).populate("race_id", ["_id", "name"]);
        return planet;

    }

    async updatePlanet(planet_id, planet) {

        const updateInfo = await PlanetModel.updateOne({ _id:
planet_id }, { $set: { discoverer: planet.discoverer, name:
planet.name, sat: planet.sat, mass: planet.mass} });


    }
    async updatePlanetPhoto(planet_id, media_path) {

        const updateInfo = await PlanetModel.updateOne({ _id:
planet_id }, { $set: { media_path: media_path } });

    }
    async deletePlanet(planet_id) {

        const updateInfo = await UserModel.updateMany({ planet_id:
planet_id }, { '$pull': { 'planet_id': planet_id } });
        const result = await PlanetModel.deleteOne({ _id: planet_id
});
        return result;

    }
    async addRaceToList(planet_id, race_id) {

        const updateInfo = await PlanetModel.updateOne({ _id:
planet_id }, { '$addToSet': { 'race_id': race_id } });

    }
    async deleteRaceFromList(planet_id, race_id) {
        const updateInfo = await PlanetModel.updateOne({ _id:
planet_id }, { '$pull': { 'race_id': race_id } });
    }
};


module.exports = PlanetRepository;
```

userRepository.js

```
const UserModel = require('../models/users');
class UserRepository {

    constructor() {

    }
```

```
    async getUsers() {

        const userDocs = await UserModel.find();
        const usersArr = userDocs.map(i => i.toJSON());
        return usersArr;

    }

    async getUserById(id) {

        const user = await
UserModel.findById(id).populate("planet_id", ["_id", "name",
"discoverer"]);
        return user;

    }
    async addPlanetToList(user_id, planet_id) {

        const updateInfo = await UserModel.updateOne({ _id: user_id
}, { '$addToSet': { 'planet_id': planet_id } });

    }
    async deletePlanetFromList(user_id, planet_id) {

        const updateInfo = await UserModel.updateOne({ _id: user_id
}, { '$pull': { 'planet_id': planet_id } });

    }
};

module.exports = UserRepository;
```

user_route_mst.js

```
const router = require('express').Router();

const userController =
require('../controllers/users_controller_mst');

router
    .get("/:id", userController.getUserById)
    .get("/", userController.getUsers)
    .get("/addplanet/:id", userController.addPlanet)
    .post("/addplanet/:id", userController.postPlanet)
    .get("/deleteplanet/:id", userController.deletePlanet)
    .post("/deleteplanet/:id", userController.postDelPlanet);


module.exports = router;
```

race_route_mst.js

```
const router = require('express').Router();
const raceController =
```

```
require('../controllers/races_controller_mst');

router
    .get("/", raceController.getRaces)
    .get("/new", raceController.newRace)
    .get("/:id", raceController.getRaceById)
    .post("/", raceController.postRace)
    .post("/:id", raceController.deleteRace)
    .get("/getupdate/:id", raceController.getupdateRace)
    .post("/getupdate/:id", raceController.updateRace)
    .get("/getupdatephoto/:id", raceController.getupdateRacePhoto)
    .post("/getupdatephoto/:id", raceController.updateRacePhoto);

module.exports = router;
```

planets_route_mst.js

```
const router = require('express').Router();
const planetController =
require('../controllers/planets_controller_mst');

router
    .get("/", planetController.getPlanets)
    .get("/new", planetController.newPlanet)
    .get("/:id", planetController.getPlanetById)
    .post("/", planetController.postPlanet)
    .post("/delete/:id", planetController.deletePlanet)
    .get("/getupdate/:id", planetController.getupdatePlanet)
    .post("/getupdate/:id", planetController.updatePlanet)
    .get("/getupdatephoto/:id",
planetController.getupdatePlanetPhoto)
    .post("/getupdatephoto/:id", planetController.updatePlanetPhoto)
    .get("/addrace/:id", planetController.addRace)
    .post("/addrace/:id", planetController.postAddRace)
    .get("/deleterace/:id", planetController.deleteRace)
    .post("/deleterace/:id", planetController.postDelRace);

module.exports = router;
```

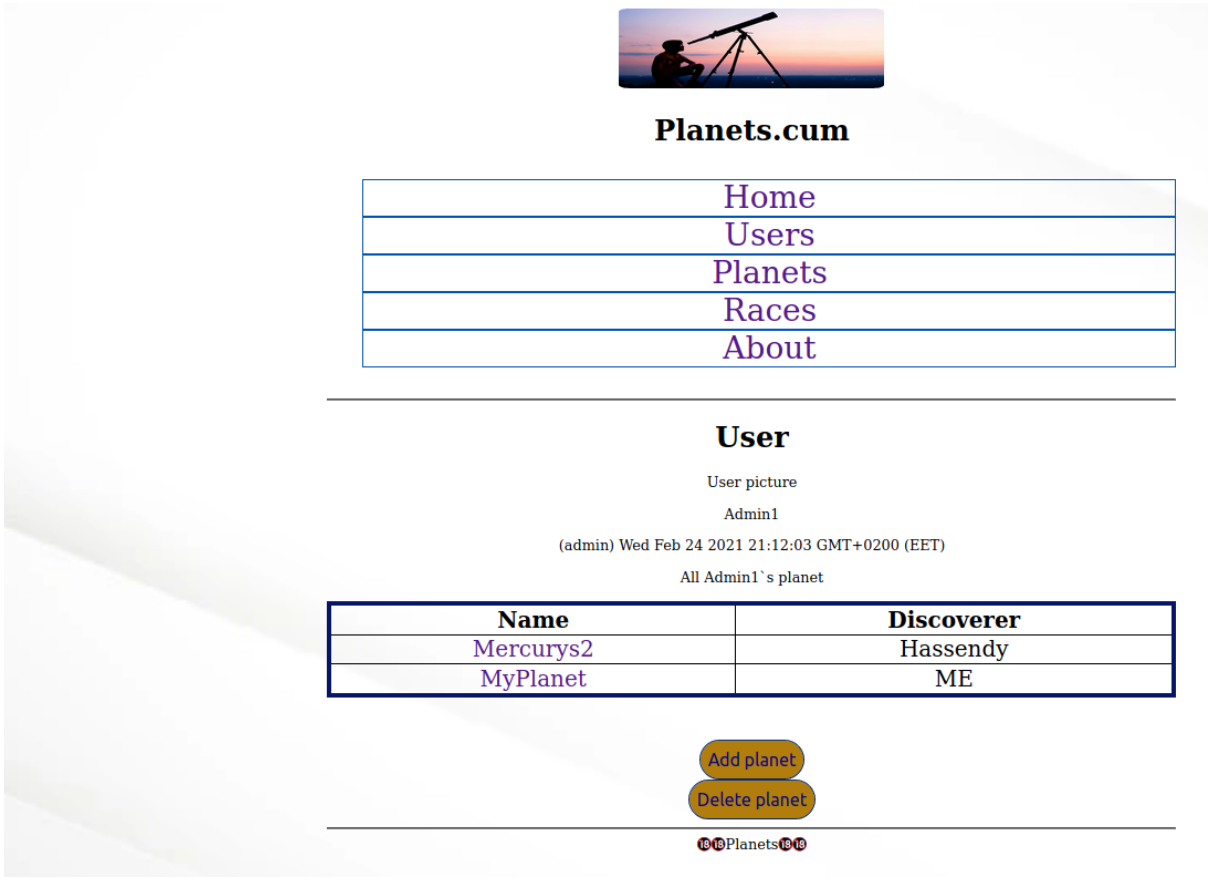
route_mst.js

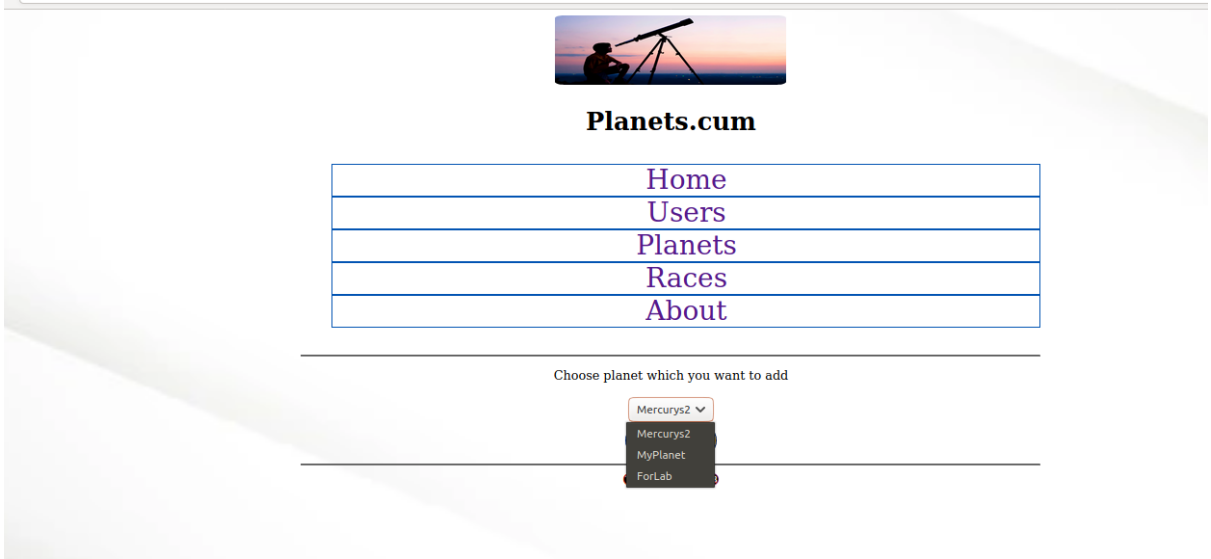
```
const userRouter = require('./users_route_mst');
const planetRouter = require('./planets_route_mst');
const raceRouter = require('./race_route_mst')
const mstRouter = require('express').Router();

mstRouter
    .use('/users', userRouter)
    .use('/planets', planetRouter)
    .use('/races', raceRouter);


module.exports = mstRouter;
```

Зображення нових та змінених сторінок





Planets.cum

Home
Users
Planets
Races
About

Choose planet which you want to delete

Mercurys2

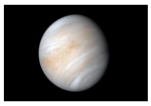
Delete

Planets

Planets.cum

Home
Users
Planets
Races
About

Planet



Planet: *Mercurys2*

Discoverer: *Hassendy*

Sat's: *7*

Mass: *99999999994*

Races who live on planet

Name
Suslenko

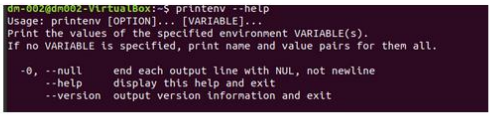
Add race
Delete race
Update planet
Delete planet
Update planet photo?

Planets

Planets.cum

Home
Users
Planets
Races
About

Race



Race *Suslenko*

Strength *15*

Intellect *4*

Dexterity *3*

Update race
Delete race
Update race photo

PlanetsDM

Planets.cum

Home
Users
Planets
Races
About

Name:
Suslenko1
Strength:
7
Intellect:
2
Dexterity:
6
Update race

PlanetsDM

Planets.cum

Home
Users
Planets
Races
About

Browse... No file selected.

Update photo

Planets

Висновки

У процесі виконання роботи вдалось Вивчити основні принципи асинхронного програмування в JavaScript. Навчитись асинхронно взаємодіяти з базою даних. Налаштувати взаємодію з віддаленою базою даних та сховищем медіа. Підготувати і опублікувати веб-сервіс в мережі Інтернет.