

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Фронт-энд разработка

Отчет

Домашнее задание №1

“Знакомство с Express.js и Sequelize ORM”

Выполнил:

Кондрашов Е. Ю.

Группа К33402

Проверил:

Добряков Д. И.

Санкт-Петербург

2022 г.

## Задача

1. Продумать свою собственную модель пользователя.
2. Реализовать набор из CRUD-методов для работы с пользователями средствами Express + Sequelize.
3. Написать запрос для получения пользователя по id/email.

## Ход работы

Описание модели пользователя средствами Sequelize ORM:

```
User.init({
  uuid: {
    type: DataTypes.UUID, primaryKey: true, defaultValue:
DataTypes.UUIDV4,
    allowNull: false,
  },
  email: DataTypes.STRING(127),
  firstName: { type: DataTypes.STRING(127), allowNull: true },
  middleName: { type: DataTypes.STRING(127), allowNull: true },
  lastName: { type: DataTypes.STRING(127), allowNull: true },
  createdAt: DataTypes.DATE,
  updatedAt: DataTypes.DATE,
  isActive: { type: DataTypes.BOOLEAN, defaultValue: true },
  isVerified: { type: DataTypes.BOOLEAN, defaultValue: false },
}, {
  sequelize,
  modelName: 'User',
})
```

В ходе создания модели и её миграций был сделан вывод о крайнем неудобстве системы миграций у Sequelize ORM.

## Реализация CRUD-методов с помощью Express.js и Sequelize:

```
app.post('/users', async (req, res) => {  
    // Create user  
  
    const user = await db.User.create(req.body)  
  
    res.send(user.toJSON())  
  
})  
  
app.get('/users', async (req, res) => {  
    // List of users  
  
    const users = await db.User.findAll({ raw: true, nest: true })  
  
    res.send(users)  
  
})  
  
app.get('/users/:uuid', async (req, res) => {  
    // Retrieve user by uuid  
  
    const user = await db.User.findByPk(req.params.uuid)  
  
    if (user) {  
        respBody = user.toJSON()  
  
    } else { respBody = { "msg": "user not found" } }  
  
    res.send(respBody)  
  
})  
  
app.put('/users/:uuid', async (req, res) => {  
    // Update user  
  
    let user = await db.User.findByPk(req.params.uuid)  
  
    if (user) {
```

```

        await db.User.update(req.body, { where: { uuid: req.params.uuid } })

    } else { await db.User.create(req.body) }

    res.send(req.body)

  })

  app.delete('/users/:uuid', async (req, res) => {

    // Delete user by uuid

    const user = await db.User.destroy({ where: { uuid: req.params.uuid } })

    if (user) {

      respBody = { "status": "success" }

    } else { respBody = { "msg": "user not found" } }

    res.send(respBody)

  })

```

Запрос на получение пользователя по email:

```

  app.get('/users/:email', async (req, res) => {

    // Retreive user by email

    const user = await db.User.findOne({ where: { email: req.params.email } })

    let respBody = null

    if (user) {

      respBody = user.toJSON()

    } else { respBody = { "msg": "user not found" } }

    res.send(respBody)

  })

```

**Вывод**

Благодаря Express.js и Sequelize можно довольно быстро создать базовое API для CRUD-операций с моделью. Описание модели в Sequelize ORM схоже с другими популярными ORM, однако отсутствие функционала автоматической генерации миграций при изменении схем моделей заставляет задуматься об использовании этой ORM в серьезных проектах, где нужно иметь историю изменения схемы БД. Ручное написание миграций является нарушением принципа DRY, а найденный мной пакет для автоматической генерации миграций для Sequelize ORM имеет ряд открытых Issues на Github и не обновлялся с 2019 года.