به نام او

علی امینی

401170529

تمرین دست گرمی 2

این تمرین از 4 بخش اصلی تشکیل شده است:

1) Server
2) Models
3) Routes
4) Controllers

Server

این بخش نکته خاصی ندارد صرفا در آن routes را می دهیم و آن را به دیتابیس خود که در اینجا mongo هست وصل می کنیم.

```
1   const express = require('express');
2   const mongoose = require('mongoose');
3   const formRoutes = require('./routes/formRoutes');
4   const fieldRoutes = require('./routes/fieldRoutes');
5
6   const app = express();
7
8
9   app.use(express.json());
10
11
12  app.use('/forms', formRoutes);
13  app.use('/fields', fieldRoutes);
14
15
16  mongoose.connect('mongodb://127.0.0.1:27017/form-api')
17    .then(() => {
18      console.log('Connected to MongoDB');
19      app.listen(3000, () => console.log('Server running on http://localhost:3000'));
20    })
21    .catch((err) => console.error('Error connecting to MongoDB', err));
22
```

Models

مدل ها به دو کد field و from تقسیم می شوند.

برای form ها 4 فیلد published , fields , description , name را تعریف می کنیم.

کد آن به شکل زیر هست:

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

const formSchema = new mongoose.Schema({
  name: { type: String, required: true },
  description: { type: String },
  fields: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Field' }],
  published: { ty                    },
});

module.exports = mongoose.model('Form', formSchema);
```

(alias) module "mongoose"
import mongoose

برای field ها label , type , default value , form را تعریف می کنیم به شکل زیر:

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

const fieldSchema = new mongoose.Schema({
  label: { type: String, required: true },
  type: { type: String, enum: ['text', 'number', 'boolean'], required: true },
  defaultValue: {
    type: mongoose.Schema.Types.Mixed,
    validate: {
      validator: function(value) {
        if (this.type === 'text') return typeof value === 'string';
        if (this.type === 'number') return typeof value === 'number';
        if (this.type === 'boolean') return typeof value === 'boolean';
        return false;
      },
      message: 'Default value does not match the field type'
    }
  },
  form: { type: mongoose.Schema.Types.ObjectId, ref: 'Form' }
}, { timestamps: true });


fieldSchema.index({ form: 1 });
```

همچنین برای اینکه اگر فیلدی را پاک کردیم از ارایه فیلدهای فرم ها هم پاک شود آن ها را باهم سینک می کنیم. برای اینکار از کد زیر کمک می گیریم:

```
fieldSchema.pre('remove', async function (next) {
  const field = this;
  try {
    await mongoose.model('Form').findByIdAndUpdate(field.form, {
      $pull: { fields: field._id }
    });
    next();
  } catch (err) {
    next(err);
  }
});

module.exports = mongoose.model('Field', fieldSchema);

```

Routes

این بخش هم به دو بخش form و field تقسیم می شود که عکس آن ها را در شکل های زیر می بینید.

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


router.get('/published', formController.getPublishedForms);
router.post('/:id/publish', formController.togglePublishedStatus);
router.post('/', formController.createForm);
router.get('/', formController.getForms);
router.get('/:id', formController.getForm);
router.put('/:id', formController.updateForm);
router.delete('/:id', formController.deleteForm);


module.exports = router;

```

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


router.post('/', fieldController.createField);
router.get('/:formId', fieldController.getFieldsByFormId);
router.put('/:id', fieldController.updateField);
router.delete('/:id', fieldController.deleteField);

module.exports = router;

```

کاری که می کنیم این هست که در اول کد مدل ریکوست را می دهیم و در بخش دوم آن تابع مربوطه را صدا می زنیم.

Controllers

کد این بخش به شکل تابع تابع هست.

با کنترلر فرم ها شروع می کنیم.

```
exports.createForm = async (req, res) => {
  try {
    const form = new Form(req.body);
    await form.save();
    res.status(201).json(form);
  } catch (err) {
    res.status(400).json({ message: 'Error creating form', error: err });
  }
};

```

این تابع برای ساختن فرم جدید هست. همچنین در صورت بروز مشکل ارور برای آن در نظر گرفته شده است.

```
exports.getForms = async (req, res) => {
  try {
    const forms = await Form.find().populate('fields');
    res.status(200).json(forms);
  } catch (err) {
    res.status(400).json({ message: 'Error retrieving forms', error: err });
  }
};

```

این تابع برای گرفتن همه ی فرم ها هست. در ابتدای تابع forms را با استفاده از فایند درست کرده و آن را برمی گردانیم. در صورت نداشتن هیچ فرمی [] برگشت داده خواهد شد.

```
exports.getForm = async (req, res) => {
  try {
    const { id } = req.params;

    if (!mongoose.Types.ObjectId.isValid(id)) {
      console.log('Invalid ID format:', id);
      return res.status(400).json({ message: 'Invalid ID format' });
    }

    const form = await Form.findById(id);
    if (!form) {
      console.log('Form not found:', id);
      return res.status(404).json({ message: 'Form not found' });
    }

    res.status(200).json(form);
  } catch (err) {
    console.error('Error in getForm controller:', err);
    res.status(500).json({ message: 'Error retrieving form', error: err.message });
  }
};
```

این تابع برای گرفتن فرم دلخواه با استفاده از id آن هست.

اول id را از پارامتر های ریکوست استخراج کرده سپس اگر ایدی ما به فرمت ولید نبود ارور می دهیم.

برای این ایدی در دیتابیس خود فایند می زنیم که اگر پیدا نشود ارور می دهیم و اگر پیدا شود آن را بر میگردانیم.

```
exports.updateForm = async (req, res) => {
  try {
    const form = await Form.findByIdAndUpdate(req.params.id, req.body, { new: true });
    if (!form) {
      return res.status(404).json({ message: 'Form not found' });
    }
    res.status(200).json(form);
  } catch (err) {
    res.status(400).json({ message: 'Error updating form', error: err });
  }
};
```

این تابع برای اپدیت کردن یک فرم هست. ابتدا فرم مورد نظر را با کمک ایدی درون ریکوست پیدا کرده اگر وجود نداشته باشد ارور می دهیم. همچنین دقت کنید که از فایند اند اپدیت استفاده کرده ایم و بادی ریکوست را برای فیلد های آن به آن داده ایم.

اگر فرم مورد نظر پیدا شود خود به خود اپدیت می شود.

```
exports.deleteForm = async (req, res) => {
  try {
    const form = await Form.findByIdAndDelete(req.params.id);
    if (!form) {
      return res.status(404).json({ message: 'Form not found' });
    }
    res.status(200).json({ message: 'Form deleted' });
  } catch (err) {
    res.status(400).json({ message: 'Error deleting form', error: err });
  }
};
```

این تابع برای پاک کردن فرم ها هست. برای این کار از فایند اند دیلیت استفاده می کنیم. اگر فرم مورد نظر پیدا نشود ارور متناسب می دهیم.

```
exports.togglePublishedStatus = async (req, res) => {
  try {
    const { id } = req.params;

    if (!mongoose.Types.ObjectId.isValid(id)) {
      return res.status(400).json({ message: 'Invalid form ID format' });
    }


    const form = await Form.findById(id);
    if (!form) {
      return res.status(404).json({ message: 'Form not found' });
    }

    form.published = !form.published;
    await form.save();

    res.status(200).json({ message: 'Published status updated', form });
  } catch (err) {
    console.error('Error toggling published status:', err);
    res.status(500).json({ message: 'Error toggling published status', error: err.message });
  }
};
```

این تابع وضعیت انتشار فرم را تاگل می کند.

ابتدا فرمت ایدی را چک کرده اگر درست باشد آن را با استفاده از فایند پیدا می کنیم.

اگر پیدا شود published را برابر با نقیض آن قرار داده و آن را تاگل می کنیم.

```
exports.getPublishedForms = async (req, res) => {
  try {
    const publishedForms = await Form.find({ published: true });

    if (!publishedForms || publishedForms.length === 0) {
      return res.status(404).json({ message: 'No published forms found' });
    }

    res.status(200).json(publishedForms);
  } catch (err) {
    console.error('Error retrieving published forms:', err);
    res.status(500).json({ message: 'Error retrieving published forms', error: err.message });
  }
};
```

این تابع برای پیدا کردن همه ی فرم هایی است که انتشار داده شده اند.

برای این کار فرم هایی را پیدا می کنیم که شرط انتشار آن ها برقرار باشد و سپس آن ها را برمی گردانیم. اگر فرمی پیدا نشود [] را برمی گردانیم.

حال به سراغ کنترلر فیلد ها می رویم.

```
exports.createField = async (req, res) => {
  try {
    const field = new Field(req.body);
    await field.save();

    const form = await Form.findById(req.body.form);
    form.fields.push(field);
    await form.save();

    res.status(201).json(field);
  } catch (err) {
    res.status(400).json({ message: 'Error creating field', error: err });
  }
};

```

این تابع برای ساختن فیلد جدید هست. ابتدا فیلد جدید را می سازد سپس ایدی فرم آن را از بادی ریکوست می گیرد و آن را پیدا می کند. بعد فیلد ساخته شده را به فرم پیدا شده پوش کرده و آن را سیو می کند.

```
exports.getFieldsByFormId = async (req, res) => {
    try {
      const { formId } = req.params;

      if (!mongoose.Types.ObjectId.isValid(formId)) {
        return res.status(400).json({ message: 'Invalid form ID format' });
      }

      const fields = await Field.find({ form: formId });

      if (!fields || fields.length === 0) {
        return res.status(404).json({ message: 'No fields found for this form' });
      }

      console.log('Fields retrieved:', fields);
      res.status(200).json(fields);
    } catch (err) {
      console.error('Error fetching fields:', err);
      res.status(500).json({ message: 'Error fetching fields', error: err.message });
    }
  };


```

این تابع برای نشان دادن فیلد های یک فرم هست. پس از چک کردن فرمت ایدی فرم آن را پیدا کرده و چک می کند اگر فیلدی ندارد یا اندازه ارایه فیلد های آن 0 هست ارور متناسب می دهد.

در غیر این صورت فیلد های پیدا شده از آن فرم را بر میگرداند.

```
44
45  exports.updateField = async (req, res) => {
46    try {
47      const field = await Field.findByIdAndUpdate(req.params.id, req.body, { new: true });
48      if (!field) {
49        return res.status(404).json({ message: 'Field not found' });
50      }
51      res.status(200).json(field);
52    } catch (err) {
53      res.status(400).json({ message: 'Error updating field', error: err });
54    }
55  };
56
```

این تابع برای اپدیت کردن یک فیلد هست. از فایند اند اپدیت استفاده کرده و فیلد با ایدیی که در پارامتر های ریکوست بوده اپدیت می کند. در صورت پیدا نشدن فیلد با این ایدی ارور متناسب می دهد.

```
exports.deleteField = async (req, res) => {
    try {
      const { id } = req.params;

      console.log('Deleting field with ID:', id);


      if (!mongoose.Types.ObjectId.isValid(id)) {
        return res.status(400).json({ message: 'Invalid ID format' });
      }


      const field = await Field.findById(id);
      if (!field) {
        return res.status(404).json({ message: 'Field not found' });
      }

      await Field.findByIdAndDelete(id);

      await Form.findByIdAndUpdate(field.form, {
        $pull: { fields: id }
      });

      console.log('Field deleted and removed from form:', id);
      res.status(200).json({ message: 'Field deleted' });
    } catch (err) {
      console.error('Error deleting field:', err);
      res.status(500).json({ message: 'Error deleting field', error: err.message });
    }
  };
```

در آخر این تابع برای پاک کردن یک فیلد هست. ابتدا ایدی آن را از ریکوست استخراج کرده و سپس فرمت آن را چک می کند. بعد از آن فیلد را با استفاده از ایدی پیدا کرده و در صورت وجود داشتن آن روی آن فایند اند دیلیت می زند.