.dockerignore

```
*/node_modules
*/.env
```

Dockerfile

```
FROM node:18-alpine AS frontend-builder

WORKDIR /build
COPY frontend/package*.json ./
RUN npm install
COPY frontend/. .
RUN npm run build



FROM node:18-alpine

WORKDIR /app
COPY backend/package*.json backend/prisma ./
RUN npm install --omit=dev && npx prisma generate
COPY backend/. .
COPY --from=frontend-builder /build/dist ./public

EXPOSE 3000
CMD node src/index.js
```

README.md

SkoolWorkshop

Inventory management system for [skoolworkshop](https://skoolworkshop.nl).
Keeps track of products stored in the warehouse and shows which products are needed for a workshop. The scrumboard of this project can be found [here.](https://scrum-aei.avans.nl:8443/secure/RapidBoard.jspa)

Backend

The backend is written in nodejs using [express](https://expressjs.com/) and [prisma](https://www.prisma.io).
Use the following commands to create and import the database. If not using docker edit the `.env` file to match your database settings.
```bash
$ docker run -d -e MARIADB_ALLOW_EMPTY_ROOT_PASSWORD=true -p 3306:3306 mariadb:latest
$ npx prisma db push
```

Then use the following commands to start the backend development server.
```bash
$ npm install
$ npm run serve
```

Frontend

The frontend is created using [vue](https://vuejs.org/), [pinia](https://pinia.vuejs.org/) and [bootstrap](https://getbootstrap.com/).
Use the following commands to start the frontend development server.
```bash
$ npm install
$ npm run serve
```

backend/.env

```
PORT=3000
DATABASE_URL="mysql://root@localhost:3306/skoolworkshop"
```

backend/.eslintrc.cjs

```
module.exports = {
  env: {
    browser: false,
    es2021: true
  },
  extends: 'standard',
  overrides: [
  ],
  parserOptions: {
    ecmaVersion: 'latest',
    sourceType: 'module'
  },
  rules: {
    indent: ['error', 4],
    'no-unused-expressions': 'off'
  }
}
```

backend/package.json

```
{
  "name": "skoolworkshop",
  "version": "0.0.1",
  "main": "src/index.js",
  "type": "module",
  "scripts": {
    "serve": "nodemon",
    "lint": "eslint --fix --ext .js .",
    "test": "mocha 'test/**/*.test.js' --exit"
  },
  "dependencies": {
    "@prisma/client": "^4.14.1",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "express-async-errors": "^3.1.1",
    "jsonschema": "^1.4.1",
    "tracer": "^1.1.6"
  },
  "devDependencies": {
    "chai": "^4.3.7",
    "eslint": "^8.41.0",
    "eslint-config-standard": "^17.0.0",
    "eslint-plugin-import": "^2.27.5",
    "eslint-plugin-n": "^16.0.0",
    "eslint-plugin-promise": "^6.1.1",
    "mocha": "^10.2.0",
    "nodemon": "^2.0.22",
    "prisma": "^4.14.1",
    "sinon": "^15.1.0"
  }
}
```

backend/prisma/schema.prisma

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")
}

model Workshop {
  id        Int    @id @default(autoincrement())
  name      String @unique
  groupSize Int
}

model Product {
  id       Int     @id @default(autoincrement())
  name     String  @unique
  stock    Int
  minStock Int
  code     String? @unique
}
```

backend/src/controller/ProductController.js

```
import { HttpError } from './error/HttpError.js'
import { PostProductRequest } from './request/product/PostProductRequest.js'
import { PutProductRequest } from './request/product/PutProductRequest.js'

export class ProductController {
    constructor (db) {
        this.db = db
    }

    async all (req, res) {
        const products = await this.db.product.findMany()
        if (!products.length) {
            throw new HttpError(404, 'no products found')
        }

        res.status(200).send(products)
    }

    async get (req, res) {
        const id = req.params.id
        const product = await this.db.product.findUnique({
            where: { id: parseInt(id) }
        })

        if (!product) {
            throw new HttpError(404, 'product not found')
        }

        res.status(200).send(product)
    }

    async post (req, res) {
        const product = new PostProductRequest(req).data()

        try {
            const result = await this.db.product.create({ data: product })
            res.status(201).send(result)
        } catch (err) {
            if (err.code === 'P2002') {
                throw new HttpError(400, 'product already exists')
            }
            throw new HttpError(500, 'could not create product')
```

```
        }
    }

    async put (req, res) {
        const id = req.params.id
        const product = new PutProductRequest(req).data()

        try {
            const result = await this.db.product.update({
                where: { id: parseInt(id) },
                data: product
            })

            res.status(200).send(result)
        } catch (err) {
            if (err.code === 'P2025') {
                throw new HttpError(404, 'product not found')
            }
            throw new HttpError(500, 'could not edit product')
        }
    }

    async delete (req, res) {
        const id = req.params.id
        try {
            await this.db.product.delete({
                where: { id: parseInt(id) }
            })

            res.status(200).send({ message: 'product removed' })
        } catch (err) {
            if (err.code === 'P2025') {
                throw new HttpError(404, 'product not found')
            }
            throw new HttpError(500, 'could not delete product')
        }
    }
}
```

backend/src/controller/WorkshopController.js

```
import { HttpError } from './error/HttpError.js'
import { PutWorkshopRequest } from './request/workshop/PutWorkshopRequest.js'
import { PostWorkshopRequest } from './request/workshop/
PostWorkshopRequest.js'

export class WorkshopController {
    constructor (db) {
        this.db = db
    }

    async all (req, res) {
        const workshops = await this.db.workshop.findMany()
        if (!workshops.length) {
            throw new HttpError(404, 'no workshops found')
        }

        res.status(200).send(workshops)
    }

    async get (req, res) {
        const id = req.params.id
        const workshop = await this.db.workshop.findUnique({
            where: { id: parseInt(id) }
        })

        if (!workshop) {
            throw new HttpError(404, 'workshop not found')
        }

        res.status(200).send(workshop)
    }

    async put (req, res) {
        const id = req.params.id
        const workshop = new PutWorkshopRequest(req).data()

        try {
            const result = await this.db.workshop.update({
                where: { id: parseInt(id) },
                data: workshop
            })
```

```
            res.status(200).send(result)
        } catch (err) {
            if (err.code === 'P2025') {
                throw new HttpError(404, 'workshop not found')
            }
            throw new HttpError(500, 'could not edit workshop')
        }
    }

    async delete (req, res) {
        const id = req.params.id
        try {
            await this.db.workshop.delete({
                where: { id: parseInt(id) }
            })

            res.status(200).send({ message: 'workshop removed' })
        } catch (err) {
            if (err.code === 'P2025') {
                throw new HttpError(404, 'workshop not found')
            }
            throw new HttpError(500, 'could not remove workshop')
        }
    }

    async post (req, res) {
        const workshop = new PostWorkshopRequest(req).data()

        try {
            const result = await this.db.workshop.create({ data: workshop })
            res.status(201).send(result)
        } catch (err) {
            if (err.code === 'P2002') {
                throw new HttpError(400, 'workshop already exists')
            }
            throw new HttpError(500, 'could not create workshop')
        }
    }
}
```

backend/src/controller/error/HttpError.js

```
/**
 * HttpError represents an error with a status code. It is used
 * to send errors to the client.
 */
export class HttpError extends Error {
    constructor (status, message, data = null) {
        super(message)
        this.status = status
        this.data = data
    }
}
```

backend/src/controller/request/Request.js

```
import { validate } from 'jsonschema'
import { HttpError } from '../error/HttpError.js'

/**
 * Request represents a request. It provides request validation.
 * @abstract
 */
export class Request {
    schema = {}

    constructor (req) {
        this.req = req
    }

    data () {
        const result = validate(this.req, this.schema)
        if (result.valid) {
            return this.req
        }

        throw new HttpError(
            400,
            'request is invalid',
            result.errors.map((error) => ({
                property: error.property,
                message: error.message
            }))
        )
    }
}
```

backend/src/controller/request/product/PostProductRequest.js

```
import { Request } from '../Request.js'

export class PostProductRequest extends Request {
    schema = {
        type: 'object',
        additionalProperties: false,
        required: ['name', 'stock', 'minStock'],
        properties: {
            name: { type: 'string' },
            stock: { type: 'number' },
            minStock: { type: 'number' },
            code: { type: 'string' }
        }
    }

    constructor (req) {
        super(req.body)
    }
}
```

backend/src/controller/request/product/PutProductRequest.js

```
import { Request } from '../Request.js'

export class PutProductRequest extends Request {
    schema = {
        type: 'object',
        additionalProperties: false,
        required: ['name', 'stock', 'minStock'],
        properties: {
            name: { type: 'string' },
            stock: { type: 'number' },
            minStock: { type: 'number' },
            code: { type: 'string' }
        }
    }

    constructor (req) {
        super(req.body)
    }
}
```

backend/src/controller/request/workshop/PostWorkshopRequest.js

```
import { Request } from '../Request.js'

export class PostWorkshopRequest extends Request {
    schema = {
        type: 'object',
        additionalProperties: false,
        required: ['name', 'groupSize'],
        properties: {
            name: { type: 'string' },
            groupSize: { type: 'number' }
        }
    }

    constructor (req) {
        super(req.body)
    }
}
```

backend/src/controller/request/workshop/PutWorkshopRequest.js

```
import { Request } from '../Request.js'

export class PutWorkshopRequest extends Request {
    schema = {
        type: 'object',
        additionalProperties: false,
        required: ['name', 'groupSize'],
        properties: {
            name: { type: 'string' },
            groupSize: { type: 'number' }
        }
    }

    constructor (req) {
        super(req.body)
    }
}
```

backend/src/index.js

```
import * as dotenv from 'dotenv'
import express from 'express'
import 'express-async-errors'
import { WorkshopController } from './controller/WorkshopController.js'
import { ProductController } from './controller/ProductController.js'
import { PrismaClient } from '@prisma/client'
import { colorConsole } from 'tracer'
import { AccessLogger } from './middleware/AccessLogger.js'
import { ErrorHandler } from './middleware/ErrorHandler.js'
import { UnknownRouteHandler } from './middleware/UnknownRouteHandler.js'
dotenv.config()

const db = new PrismaClient()
const logger = colorConsole()
const middleware = {
    accessLogger: new AccessLogger(logger),
    errorHandler: new ErrorHandler(logger, process.env.NODE_ENV ===
'production'),
    unknownRouteHandler: new UnknownRouteHandler()
}
const controller = {
    workshop: new WorkshopController(db),
    product: new ProductController(db)
}

// Create express app and register middleware.
const app = express()
    .use(express.json())
    .use(express.static('public'))
    .use((req, res, next) => middleware.accessLogger.exec(req, res, next))

// Register routes.
app
    .get('/api/workshops', (req, res) => controller.workshop.all(req, res))
    .post('/api/workshops', (req, res) => controller.workshop.post(req, res))
    .get('/api/workshops/:id', (req, res) => controller.workshop.get(req,
res))
    .put('/api/workshops/:id', (req, res) => controller.workshop.put(req,
res))
    .delete('/api/workshops/:id', (req, res) =>
controller.workshop.delete(req, res))
```

```
app
    .get('/api/products', (req, res) => controller.product.all(req, res))
    .get('/api/products/:id', (req, res) => controller.product.get(req, res))
    .post('/api/products', (req, res) => controller.product.post(req, res))
    .put('/api/products/:id', (req, res) => controller.product.put(req, res))
    .delete('/api/products/:id', (req, res) => controller.product.delete(req,
res))

// Register error handlers.
app
    .use((err, req, res, next) => middleware.errorHandler.exec(err, req, res,
next))
    .use((req, res, next) => middleware.unknownRouteHandler.exec(req, res,
next))

app.listen(process.env.PORT, () => console.log(`listening on port
${process.env.PORT}`))
```

backend/src/middleware/AccessLogger.js

```
/**
 * AccessLogger logs all requests to the console.
 */
export class AccessLogger {
    constructor (logger) {
        this.logger = logger
    }

    exec (req, res, next) {
        this.logger.log(req.ip, ' ', req.method, ' ', req.path)
        next()
    }
}
```

backend/src/middleware/ErrorHandler.js

```
/**
 * ErrorHandler catches and logs errors and sends them to the client.
 */
export class ErrorHandler {
    constructor (logger, production) {
        this.logger = logger
        this.production = production
    }

    exec (err, req, res, next) {
        const status = err.status || 500
        const response = { error: err.message }

        if (this.production && status === 500) {
            // Hide internal error messages when running in production.
            response.error = 'internal server error'
        }

        if (err.data) {
            response.data = err.data
        }

        this.logger.error(req.ip, ' ', req.method, ' ', req.path, '\n',
err.stack)
        res.status(status).send(response)
    }
}
```

backend/src/middleware/UnknownRouteHandler.js

```
import { join } from 'path'

/**
 * UnknownRouteHandler handles all unknown routes. It returns
 * an error message or index.html when route is not an API.
 */
export class UnknownRouteHandler {
    exec (req, res, next) {
        if (req.path.startsWith('/api')) {
            res.status(404).json({ error: 'API route not found' })
            return
        }

        // always return the index.html when route is not an API.
        // vue-router will direct the user to the correct page.
        res.sendFile(join(process.cwd(), 'public', 'index.html'))
    }
}
```

backend/test/controller/ProductController.test.js

```
import { describe, it } from 'mocha'
import { expect } from 'chai'
import sinon from 'sinon'
import { ProductController } from '../../src/controller/ProductController.js'

describe('controller/ProductController', () => {
    const products = [
        { id: 1, name: 'Product 1', stock: 100, minStock: 10, code:
'012345678' },
        { id: 2, name: 'Product 2', stock: 200, minStock: 20, code:
'123456789' }
    ]

    describe('all', () => {
        it('should return a list of products', async () => {
            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { product: { findMany:
sinon.stub().returns(products) } }
            const controller = new ProductController(db)

            await controller.all({}, res)
            expect(db.product.findMany.calledOnce).to.be.true
            expect(res.status.calledOnceWith(200)).to.be.true
            expect(res.send.calledOnceWith(products)).to.be.true
        })

        it('should return 404 if no products are found', async () => {
            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { product: { findMany: sinon.stub().returns([]) } }
            const controller = new ProductController(db)

            try {
                await controller.all({}, res)
                expect.fail('should have thrown an error')
            } catch (err) {
                expect(err.message).to.equal('no products found')
                expect(db.product.findMany.calledOnce).to.be.true
            }
        })
    })
```

```
    describe('get', () => {
        it('should return a specific product', async () => {
            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { product: { findUnique:
sinon.stub().returns(products[0]) } }
            const controller = new ProductController(db)

            await controller.get({ params: 1 }, res)
            expect(db.product.findUnique.calledOnce).to.be.true
            expect(res.status.calledOnceWith(200)).to.be.true
            expect(res.send.calledOnceWith(products[0])).to.be.true
        })

        it('should return 404 if no product is found', async () => {
            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { product: { findUnique: sinon.stub().returns(null) } }
            const controller = new ProductController(db)

            try {
                await controller.get({ params: 1 }, res)
                expect.fail('should have thrown an error')
            } catch (err) {
                expect(err.message).to.equal('product not found')
                expect(db.product.findUnique.calledOnce).to.be.true
            }
        })
    })

    describe('post', () => {
        it('should create a new product', async () => {
            const req = { body: products[0] }
            delete req.body.id

            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { product: { create:
sinon.stub().returns(products[0]) } }
            const controller = new ProductController(db)

            await controller.post(req, res)
            expect(db.product.create.calledOnce).to.be.true
```

```
            expect(res.status.calledOnceWith(201)).to.be.true
            expect(res.send.calledOnceWith(products[0])).to.be.true
        })

        it('should return 400 when there is a conflict', async () => {
            const error = new Error()
            error.code = 'P2002'

            const req = { body: products[0] }
            delete req.body.id

            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { product: { create: sinon.stub().throws(error) } }
            const controller = new ProductController(db)

            try {
                await controller.post(req, res)
                expect.fail('should have thrown an error')
            } catch (err) {
                expect(db.product.create.calledOnce).to.be.true
                expect(err.message).to.equal('product already exists')
                expect(err.status).to.equal(400)
            }
        })
    })

    describe('put', () => {
        it('should update a product', async () => {
            const product = { ...products[0], name: 'Product 1 updated' }
            const req = { params: { id: 1 }, body: product }
            delete req.body.id

            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { product: { update: sinon.stub().returns(product) } }
            const controller = new ProductController(db)

            await controller.put(req, res)
            expect(db.product.update.calledOnce).to.be.true
            expect(res.status.calledOnceWith(200)).to.be.true
            expect(res.send.calledOnceWith(product)).to.be.true
        })
```

```
        it('should return 404 when a product does not exist', async () => {
            const error = new Error()
            error.code = 'P2025'

            const product = { ...products[0], name: 'Workshop 1 updated' }
            const req = { params: { id: 1 }, body: product }
            delete req.body.id

            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { product: { update: sinon.stub().throws(error) } }
            const controller = new ProductController(db)

            try {
                await controller.put(req, res)
                expect.fail('should have thrown an error')
            } catch (err) {
                expect(db.product.update.calledOnce).to.be.true
                expect(err.message).to.equal('product not found')
                expect(err.status).to.equal(404)
            }
        })
    })

    describe('delete', () => {
        it('should delete a product', async () => {
            const req = { params: { id: 1 } }
            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { product: { delete: sinon.stub() } }
            const controller = new ProductController(db)

            await controller.delete(req, res)
            expect(db.product.delete.calledOnce).to.be.true
            expect(res.status.calledOnceWith(200)).to.be.true
            expect(res.send.calledOnceWith({ message: 'product
removed' })).to.be.true
        })

        it('should return 404 when a product does not exist', async () => {
            const error = new Error()
            error.code = 'P2025'

            const req = { params: { id: 1 } }
```

```
            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { product: { delete: sinon.stub().throws(error) } }
            const controller = new ProductController(db)

            try {
                await controller.delete(req, res)
                expect.fail('should have thrown an error')
            } catch (err) {
                expect(db.product.delete.calledOnce).to.be.true
                expect(err.message).to.equal('product not found')
                expect(err.status).to.equal(404)
            }
        })
    })
})
```

backend/test/controller/WorkshopController.test.js

```
import { describe, it } from 'mocha'
import { expect } from 'chai'
import sinon from 'sinon'
import { WorkshopController } from '../../src/controller/
WorkshopController.js'

describe('controller/WorkshopController', () => {
    const workshops = [
        { id: 1, name: 'Workshop 1', groupSize: 10 },
        { id: 2, name: 'Workshop 2', groupSize: 25 }
    ]

    describe('all', () => {
        it('should return a list of workshops', async () => {
            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { workshop: { findMany:
sinon.stub().returns(workshops) } }
            const controller = new WorkshopController(db)

            await controller.all({}, res)
            expect(db.workshop.findMany.calledOnce).to.be.true
            expect(res.status.calledOnceWith(200)).to.be.true
            expect(res.send.calledOnceWith(workshops)).to.be.true
        })

        it('should return 404 if no workshops are found', async () => {
            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { workshop: { findMany: sinon.stub().returns([]) } }
            const controller = new WorkshopController(db)

            try {
                await controller.all({}, res)
                expect.fail('should have thrown an error')
            } catch (err) {
                expect(err.message).to.equal('no workshops found')
                expect(db.workshop.findMany.calledOnce).to.be.true
            }
        })
    })
```

```
    describe('get', () => {
        it('should return a specific workshop', async () => {
            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { workshop: { findUnique:
sinon.stub().returns(workshops[0]) } }
            const controller = new WorkshopController(db)

            await controller.get({ params: 1 }, res)
            expect(db.workshop.findUnique.calledOnce).to.be.true
            expect(res.status.calledOnceWith(200)).to.be.true
            expect(res.send.calledOnceWith(workshops[0])).to.be.true
        })

        it('should return 404 if no workshop is found', async () => {
            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { workshop: { findUnique:
sinon.stub().returns(null) } }
            const controller = new WorkshopController(db)

            try {
                await controller.get({ params: 1 }, res)
                expect.fail('should have thrown an error')
            } catch (err) {
                expect(err.message).to.equal('workshop not found')
                expect(db.workshop.findUnique.calledOnce).to.be.true
            }
        })
    })

    describe('post', () => {
        it('should create a new workshop', async () => {
            const req = { body: workshops[0] }
            delete req.body.id

            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { workshop: { create:
sinon.stub().returns(workshops[0]) } }
            const controller = new WorkshopController(db)

            await controller.post(req, res)
            expect(db.workshop.create.calledOnce).to.be.true
```

```
            expect(res.status.calledOnceWith(201)).to.be.true
            expect(res.send.calledOnceWith(workshops[0])).to.be.true
        })

        it('should return 400 when there is a conflict', async () => {
            const error = new Error()
            error.code = 'P2002'

            const req = { body: workshops[0] }
            delete req.body.id

            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { workshop: { create: sinon.stub().throws(error) } }
            const controller = new WorkshopController(db)

            try {
                await controller.post(req, res)
                expect.fail('should have thrown an error')
            } catch (err) {
                expect(db.workshop.create.calledOnce).to.be.true
                expect(err.message).to.equal('workshop already exists')
                expect(err.status).to.equal(400)
            }
        })
    })

    describe('put', () => {
        it('should update a workshop', async () => {
            const workshop = { ...workshops[0], name: 'Workshop 1 updated' }
            const req = { params: { id: 1 }, body: workshop }
            delete req.body.id

            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { workshop: { update:
sinon.stub().returns(workshop) } }
            const controller = new WorkshopController(db)

            await controller.put(req, res)
            expect(db.workshop.update.calledOnce).to.be.true
            expect(res.status.calledOnceWith(200)).to.be.true
            expect(res.send.calledOnceWith(workshop)).to.be.true
        })
```

```
        it('should return 404 when a workshop does not exist', async () => {
            const error = new Error()
            error.code = 'P2025'

            const workshop = { ...workshops[0], name: 'Workshop 1 updated' }
            const req = { params: { id: 1 }, body: workshop }
            delete req.body.id

            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { workshop: { update: sinon.stub().throws(error) } }
            const controller = new WorkshopController(db)

            try {
                await controller.put(req, res)
                expect.fail('should have thrown an error')
            } catch (err) {
                expect(db.workshop.update.calledOnce).to.be.true
                expect(err.message).to.equal('workshop not found')
                expect(err.status).to.equal(404)
            }
        })
    })

    describe('delete', () => {
        it('should delete a workshop', async () => {
            const req = { params: { id: 1 } }
            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { workshop: { delete: sinon.stub() } }
            const controller = new WorkshopController(db)

            await controller.delete(req, res)
            expect(db.workshop.delete.calledOnce).to.be.true
            expect(res.status.calledOnceWith(200)).to.be.true
            expect(res.send.calledOnceWith({ message: 'workshop
removed' })).to.be.true
        })

        it('should return 404 when a workshop does not exist', async () => {
            const error = new Error()
            error.code = 'P2025'
```

```
            const req = { params: { id: 1 } }
            const res = { status: sinon.stub().returnsThis(), send:
sinon.stub() }
            const db = { workshop: { delete: sinon.stub().throws(error) } }
            const controller = new WorkshopController(db)

            try {
                await controller.delete(req, res)
                expect.fail('should have thrown an error')
            } catch (err) {
                expect(db.workshop.delete.calledOnce).to.be.true
                expect(err.message).to.equal('workshop not found')
                expect(err.status).to.equal(404)
            }
        })
    })
})
```

backend/test/controller/request/product/PostProductRequest.test.js

```
import { describe, it } from 'mocha'
import { expect } from 'chai'
import { PostProductRequest } from '../../../../src/controller/request/
product/PostProductRequest.js'

describe('controller/request/product/PostProductRequest', () => {
    it('should accept valid requests', async () => {
        const req = {
            body: {
                name: 'product 1',
                stock: 10,
                minStock: 5,
                code: '0123456789'
            }
        }

        const data = new PostProductRequest(req).data()
        expect(data).to.deep.equal(req.body)
    })
})
```

backend/test/controller/request/product/PutProductRequest.test.js

```
import { describe, it } from 'mocha'
import { expect } from 'chai'
import { PutProductRequest } from '../../../../src/controller/request/product/
PutProductRequest.js'

describe('controller/request/product/PutProductRequest', () => {
    it('should accept valid requests', async () => {
        const req = {
            body: {
                name: 'product 1',
                stock: 10,
                minStock: 5,
                code: '123456789'
            }
        }

        const data = new PutProductRequest(req).data()
        expect(data).to.deep.equal(req.body)
    })
})
```

backend/test/controller/request/workshop/PostWorkshopRequest.test.js

```
import { describe, it } from 'mocha'
import { expect } from 'chai'
import { PostWorkshopRequest } from '../../../../src/controller/request/
workshop/PostWorkshopRequest.js'

describe('controller/request/workshop/PostWorkshopRequest', () => {
    it('should accept valid requests', async () => {
        const req = {
            body: {
                name: 'workshop 1',
                groupSize: 10
            }
        }

        const data = new PostWorkshopRequest(req).data()
        expect(data).to.deep.equal(req.body)
    })
})
```

backend/test/controller/request/workshop/PutWorkshopRequest.test.js

```
import { describe, it } from 'mocha'
import { expect } from 'chai'
import { PutWorkshopRequest } from '../../../../src/controller/request/
workshop/PutWorkshopRequest.js'

describe('controller/request/workshop/PutWorkshopRequest', () => {
    it('should accept valid requests', async () => {
        const req = {
            body: {
                name: 'Workshop 1',
                groupSize: 10
            }
        }

        const data = new PutWorkshopRequest(req).data()
        expect(data).to.deep.equal(req.body)
    })
})
```

backend/test/middleware/AccessLogger.test.js

```
import { describe, it } from 'mocha'
import { expect } from 'chai'
import sinon from 'sinon'
import { AccessLogger } from '../../src/middleware/AccessLogger.js'

describe('middleware/AccessLogger', () => {
    it('should log requests', async () => {
        const logger = { log: sinon.stub() }
        const next = sinon.stub()
        const req = { ip: '192.168.1.1', method: 'GET', path: '/api/
workshops' }

        const accessLogger = new AccessLogger(logger)
        await accessLogger.exec(req, {}, next)

        expect(logger.log.calledOnceWith(req.ip, ' ', req.method, ' ',
req.path)).to.be.true
        expect(next.calledOnce).to.be.true
    })
})
```

backend/test/middleware/ErrorHandler.test.js

```
import { describe, it } from 'mocha'
import { expect } from 'chai'
import sinon from 'sinon'
import { ErrorHandler } from '../../src/middleware/ErrorHandler.js'

describe('middleware/ErrorHandler', () => {
    it('should handle errors', async () => {
        const logger = { error: sinon.stub() }
        const err = {
            status: 500,
            message: 'error message',
            stack: 'error stack'
        }
        const req = { ip: '192.168.1.1', method: 'GET', path: '/api/
workshops' }
        const res = { status: sinon.stub().returnsThis(), send: sinon.stub() }
        const next = sinon.stub()

        const errorHandler = new ErrorHandler(logger, false)
        await errorHandler.exec(err, req, res, next)

        expect(res.status.calledOnceWith(err.status)).to.be.true
        expect(res.send.calledOnceWith({ error: err.message })).to.be.true
        expect(logger.error.calledOnceWith(req.ip, ' ', req.method, ' ',
req.path, '\n', err.stack)).to.be.true
        expect(next.calledOnce).to.be.false
    })

    it('should hide error messages in production', async () => {
        const logger = { error: sinon.stub() }
        const err = {
            status: 500,
            message: 'error message',
            stack: 'error stack'
        }
        const req = { ip: '192.168.1.1', method: 'GET', path: '/api/
workshops' }
        const res = { status: sinon.stub().returnsThis(), send: sinon.stub() }
        const next = sinon.stub()

        const errorHandler = new ErrorHandler(logger, true)
        await errorHandler.exec(err, req, res, next)
```

```
        expect(res.status.calledOnceWith(err.status)).to.be.true
        expect(res.send.calledOnceWith({ error: 'internal server
error' })).to.be.true
        expect(logger.error.calledOnceWith(req.ip, ' ', req.method, ' ',
req.path, '\n', err.stack)).to.be.true
        expect(next.calledOnce).to.be.false
    })
})
```

backend/test/middleware/UnknownRouteHandler.test.js

```
import { describe, it } from 'mocha'
import { expect } from 'chai'
import sinon from 'sinon'
import { UnknownRouteHandler } from '../../src/middleware/
UnknownRouteHandler.js'
import { join } from 'path'

describe('middleware/UnknownRouteHandler', () => {
    it('should return route unknown message for unknown API endpoints', async
() => {
        const next = sinon.stub()
        const req = { path: '/api/workshops' }
        const res = { status: sinon.stub().returnsThis(), json: sinon.stub() }

        const unknownRouteHandler = new UnknownRouteHandler()
        await unknownRouteHandler.exec(req, res, next)

        expect(res.status.calledOnceWith(404)).to.be.true
        expect(res.json.calledOnceWith({ error: 'API route not
found' })).to.be.true
        expect(next.calledOnce).to.be.false
    })

    it('should return index.html for unknown non-api routes', async () => {
        const next = sinon.stub()
        const req = { path: '/products' }
        const res = { sendFile: sinon.stub() }

        const unknownRouteHandler = new UnknownRouteHandler()
        await unknownRouteHandler.exec(req, res, next)

        expect(res.sendFile.calledOnceWith(join(process.cwd(), 'public',
'index.html'))).to.be.true
        expect(next.calledOnce).to.be.false
    })
})
```

frontend/.eslintrc.cjs

```
module.exports = {
    env: {
        browser: true,
        es2021: true
    },
    extends: [
        'plugin:vue/vue3-essential',
        'standard'
    ],
    overrides: [
    ],
    parserOptions: {
        ecmaVersion: 'latest',
        sourceType: 'module'
    },
    plugins: [
        'vue'
    ],
    rules: {
        indent: ['error', 4],
        'vue/multi-word-component-names': 'off'
    },
    ignorePatterns: ['node_modules', 'dist']
}
```

frontend/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Warehouse</title>
  </head>
  <body>
    <div id="app"></div>
    <script type="module" src="/src/main.js"></script>
  </body>
</html>
```

frontend/package.json

```
{
  "name": "skoolworkshop",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "serve": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "lint": "eslint --fix --ext .js,.vue ."
  },
  "dependencies": {
    "@fortawesome/fontawesome-svg-core": "^6.4.0",
    "@fortawesome/free-solid-svg-icons": "^6.4.0",
    "@fortawesome/vue-fontawesome": "^3.0.3",
    "bootstrap": "^5.2.3",
    "pinia": "^2.1.3",
    "vue": "^3.2.47",
    "vue-barcode-reader": "^1.0.3",
    "vue-router": "^4.2.1"
  },
  "devDependencies": {
    "@vitejs/plugin-vue": "^4.1.0",
    "eslint": "^8.41.0",
    "eslint-config-standard": "^17.0.0",
    "eslint-plugin-import": "^2.27.5",
    "eslint-plugin-n": "^16.0.0",
    "eslint-plugin-promise": "^6.1.1",
    "eslint-plugin-vue": "^9.14.0",
    "sass": "^1.62.1",
    "vite": "^4.3.8"
  }
}
```

frontend/src/App.vue

```
<script setup>
import NavigationBar from './component/layout/NavigationBar.vue'
import { onErrorCaptured, ref } from 'vue'
import ErrorNotification from './component/layout/ErrorNotification.vue'

// handle exceptions from components
const message = ref('')
onErrorCaptured((error) => {
    message.value = error.message
    setTimeout(() => { message.value = '' }, 5000)
})
</script>

<template>
  <div>
    <div class="content">
      <suspense>
        <router-view/>
      </suspense>
    </div>

    <navigation-bar/>
    <error-notification :message="message" :shown="!!message" @close="message
= ''"/>
  </div>
</template>
```

frontend/src/component/input/NumberInput.vue

```
<script setup>
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'
import { ref, watch } from 'vue'

const emit = defineEmits(['update:value'])
const props = defineProps({
    name: {
        type: String,
        required: true
    },
    value: {
        type: Number,
        default: 1
    }
})

// create a copy of the value prop to be able to edit it
// without directly changing the prop
const value = ref(props.value)
watch(() => props.value, (newValue) => {
    value.value = newValue
})

function update () {
    emit('update:value', value.value)
}
</script>

<template>
  <div class="d-flex align-items-center p-2 border-bottom">
    <span class="mx-3">{{ name }}</span>

    <div class="ms-auto d-flex align-items-center">
      <div role="button" @click="value = Math.max(value - 1, 0); update()">
        <font-awesome-icon
            :icon="['fas', 'minus']"
            class="p-3 mx-2 rounded-3 hover-darken" />
      </div>

      <input type="number" class="form-control-plaintext" style="width: 2rem"
v-model="value" @input="update" />
```

```
      <div role="button" @click="value += 1; update()">
        <font-awesome-icon
            :icon="['fas', 'plus']"
            class="p-3 mx-2 rounded-3 hover-darken" />
      </div>
    </div>
  </div>
</template>
```

frontend/src/component/input/TextInput.vue

```
<script setup>
import { ref, watch } from 'vue'
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'

const emit = defineEmits(['update:value'])
const props = defineProps({
    name: {
        type: String,
        required: true
    },
    value: {
        type: String,
        default: ''
    }
})

// create a copy of the value prop to be able to edit it
// without directly changing the prop
const value = ref(props.value)
watch(() => props.value, (newValue) => {
    value.value = newValue
})

const edit = ref(value.value === '')

function update () {
    if (!edit.value) {
        edit.value = true
        return
    }

    if (!value.value) value.value = props.value
    if (value.value) {
        emit('update:value', value.value)
        edit.value = false
    }
}
</script>

<template>
  <div class="d-flex align-items-center p-2 border-bottom">
    <span class="mx-3">{{ name }}</span>
```

```
    <div class="ms-auto d-flex align-items-center">
      <span v-if="!edit">{{ value }}</span>
      <input v-else type="text" class="form-control" v-model="value"
@keydown.enter="update" @focusout="update" autofocus/>
      <div role="button" @click="update">
        <font-awesome-icon
            :icon="['fas', 'pen']"
            class="p-3 mx-2 rounded-3 hover-darken"
            :class="{'bg-secondary': edit}"/>
      </div>
    </div>
  </div>
</template>

<style>
/* hide number input arrows */
input::-webkit-outer-spin-button,
input::-webkit-inner-spin-button {
  -webkit-appearance: none;
  margin: 0;
}

input[type=number] {
  -moz-appearance: textfield;
}
</style>
```

frontend/src/component/layout/ErrorNotification.vue

```
<script setup>
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'

const emit = defineEmits(['close'])
const props = defineProps({
    shown: {
        type: Boolean,
        required: true
    },
    message: {
        type: String,
        required: true
    }
})
</script>

<template>
  <div class="alert alert-danger fixed-top m-3 d-flex align-items-center" v-
if="props.shown">
    <strong>Error!</strong>
    <span class="ms-1">{{ props.message }}</span>
    <button class="btn ms-auto alert-link" @click="emit('close')">
      <font-awesome-icon :icon="['fas', 'x']" />
    </button>
  </div>
</template>
```

frontend/src/component/layout/NavigationBar.vue

```
<script setup>
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'
import { useRoute } from 'vue-router'

const route = useRoute()
</script>

<template>
  <div class="navigation-bar">
    <img src="../../assets/logo.png" alt="logo" class="d-none d-sm-block px-3
py-3 border-bottom" style="height: 4rem" />

    <router-link to="/workshops"
                 class="d-flex justify-content-center align-items-center
w-100"
                 :class="{'text-primary': route.meta.nav === 'workshop'}">
      <font-awesome-icon :icon="['fas', 'people-group']" class="fa-2x p-3"/>
    </router-link>

    <router-link to="/scan"
                 class="d-flex d-sm-none justify-content-center align-items-
center bg-primary rounded-circle"
                 style="height: 4rem; width: 4rem; margin-top: -1rem">
      <font-awesome-icon :icon="['fas', 'qrcode']" class="fa-2x p-3"/>
    </router-link>

    <router-link to="/products"
                 class="d-flex justify-content-center align-items-center
w-100"
                 :class="{'text-primary': route.meta.nav === 'product'}">
      <font-awesome-icon :icon="['fas', 'boxes-stacked']" class="fa-2x p-3"/>
    </router-link>
  </div>
</template>
```

frontend/src/component/product/ProductItem.vue

```
<script setup>
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'

const emit = defineEmits(['click', 'delete'])
const props = defineProps({
    product: {
        type: Object,
        required: true
    },
    edit: {
        type: Boolean,
        default: false
    }
})
</script>

<template>
  <router-link
       class="d-flex align-items-center border-bottom hover-darken"
       @click="emit('click', props.product)"
       :to="`/products/${props.product.id}`">

    <!-- image and title -->
    <font-awesome-icon :icon="['fas', 'box']" class="fa-3x img border p-3
ms-1 me-3 my-3"/>
    <span class="h5"> {{ props.product.name }} </span>

    <div v-if="!props.edit" class="ms-auto">
      <!-- product stock status -->
      <font-awesome-icon v-if="props.product.stock >=
props.product.minStock" :icon="['fas', 'check']" class="fa-1x rounded-circle
bg-success p-1 m-4 text-white" style="width:20px;height:20px;"/>
      <font-awesome-icon v-else :icon="['fas', 'xmark']" class="fa-1x rounded-
circle bg-danger p-1 m-4 text-white" style="width:20px;height:20px;"/>
    </div>
    <div v-else class="ms-auto">
      <!-- edit mode buttons -->
      <button class="btn p-2 hover-darken" @click.prevent="emit('delete',
product)">
        <font-awesome-icon :icon="['fas', 'trash']" class="scale-up-center fa-
xl rounded-circle p-3 bg-danger text-white"/>
      </button>
```

```
    </div>
  </router-link>
</template>
```

frontend/src/component/workshop/WorkshopItem.vue

```
<script setup>
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'

const emit = defineEmits(['click', 'delete'])
const props = defineProps({
    workshop: {
        type: Object,
        required: true
    },
    edit: {
        type: Boolean,
        default: false
    }
})
</script>

<template>
  <router-link
       class="d-flex align-items-center border-bottom hover-darken"
       @click="emit('click', props.workshop)"
       :to="`/workshops/${props.workshop.id}`">

    <!-- image and title -->
    <font-awesome-icon :icon="['fas', 'people-robbery']" class="fa-3x img
border p-3 ms-1 me-3 my-3"/>
    <span class="h5"> {{ props.workshop.name }} </span>

    <div v-if="!props.edit" class="ms-auto">
      <!-- workshop stock status -->
      <font-awesome-icon :icon="['fas', 'check']" class="fa-1x rounded-circle
p-1 m-4 bg-success text-white" style="width:20px;height:20px;"/>
    </div>
    <div v-else class="ms-auto">
      <!-- edit mode buttons -->
      <button class="btn p-2 hover-darken" @click.prevent="emit('delete',
workshop)">
        <font-awesome-icon :icon="['fas', 'trash']" class="scale-up-center fa-
xl rounded-circle p-3 bg-danger text-white"/>
      </button>
    </div>
  </router-link>
</template>
```

frontend/src/icons.js

```
import { library } from '@fortawesome/fontawesome-svg-core'
import {
    faCircleInfo,
    faCheck,
    faPenToSquare,
    faTrash,
    faPlus,
    faMinus,
    faXmark,
    faPen,
    faX,
    faPeopleGroup,
    faPeopleRobbery,
    faBoxesStacked,
    faBox,
    faQrcode,
    faCaretLeft,
    faFloppyDisk
} from '@fortawesome/free-solid-svg-icons'

library.add(
    faCircleInfo,
    faCheck,
    faPenToSquare,
    faTrash,
    faPlus,
    faMinus,
    faXmark,
    faPen,
    faX,
    faPeopleGroup,
    faPeopleRobbery,
    faBoxesStacked,
    faBox,
    faQrcode,
    faCaretLeft,
    faFloppyDisk
)
```

frontend/src/main.js

```
import { createApp } from 'vue'
import App from './App.vue'
import { createPinia } from 'pinia'
import router from './router/router.js'
import './style/styles.scss'
import './icons.js'

createApp(App)
    .use(router)
    .use(createPinia())
    .mount('#app')
```

frontend/src/router/router.js

```
import { createRouter, createWebHistory } from 'vue-router'

const routes = [
    {
        path: '/',
        alias: '/workshops',
        name: 'workshops',
        meta: { nav: 'workshop' },
        component: () => import('../views/Workshops.vue')
    },
    {
        path: '/workshops/:id',
        name: 'workshop-details',
        meta: { nav: 'workshop' },
        component: () => import('../views/WorkshopDetails.vue')
    },
    {
        path: '/workshops/new',
        name: 'workshop-create',
        role: 'workshop',
        meta: { nav: 'workshop' },
        component: () => import('../views/WorkshopCreate.vue')
    },
    {
        path: '/products',
        name: 'products',
        meta: { nav: 'product' },
        component: () => import('../views/Products.vue')
    },
    {
        path: '/products/new',
        name: 'product-create',
        meta: { nav: 'product' },
        component: () => import('../views/ProductCreate.vue')
    },
    {
        path: '/products/:id',
        name: 'product-details',
        meta: { nav: 'product' },
        component: () => import('../views/ProductDetails.vue')
    },
    {
```

```
        path: '/scan',
        name: 'scan',
        meta: { nav: 'scan' },
        component: () => import('../views/Scan.vue')
    }
]

export default createRouter({
    history: createWebHistory(),
    routes
})
```

frontend/src/store/productStore.js

```
import { defineStore } from 'pinia'
import { API } from '../util/Api.js'

export const useProductStore = defineStore('product', {
    state: () => ({
        fetched: false,
        products: []
    }),
    actions: {
        async fetch (force = false) {
            if (this.fetched && !force) return

            const { response, ok } = await API.Req('GET', '/api/products')
            if (ok) {
                this.products = response
                this.fetched = true
            } else {
                this.products = []
            }
        },

        async get (id) {
            const product = this.products.find(item => item.id === id)
            if (product) return product

            const { response, ok } = await API.Req('GET', `/api/products/${id}
`)
            if (ok) {
                this.products.push(response)
                return response
            } else {
                throw new Error(response.error)
            }
        },

        async create (product) {
            const { response, ok } = await API.Req('POST', '/api/products',
{ body: product })
            if (ok) {
                this.products.push(response)
            } else {
                throw new Error(response.error)
```

```
            }
        },

        async update (data, id) {
            const { response, ok } = await API.Req('PUT', `/api/products/${id}
`, { body: data })
            if (ok) {
                const idx = this.products.findIndex(p => p.id === data.id)
                this.products[idx] = response
            } else {
                throw new Error(response.error)
            }
        },

        async delete (id) {
            const { response, ok } = await API.Req('DELETE', `/api/products/
${id}`)
            if (ok) {
                this.products = this.products.filter(w => w.id !== id)
            } else {
                throw new Error(response.error)
            }
        },

        search (query) {
            return this.products.filter(product =>
product.name.toLowerCase().includes(query.toLowerCase()))
        },

        findCode (code) {
            return this.products.find(product => product.code === code)
        }
    }
})
```

frontend/src/store/workshopStore.js

```
import { defineStore } from 'pinia'
import { API } from '../util/Api.js'

export const useWorkshopStore = defineStore('workshop', {
    state: () => ({
        fetched: false,
        workshops: []
    }),
    actions: {
        async fetch (force = false) {
            if (this.fetched && !force) return

            const { response, ok } = await API.Req('GET', '/api/workshops')
            if (ok) {
                this.workshops = response
                this.fetched = true
            } else {
                this.workshops = []
            }
        },

        async get (id) {
            const workshop = this.workshops.find(item => item.id === id)
            if (workshop) return workshop

            const { response, ok } = await API.Req('GET', `/api/workshops/
${id}`)
            if (ok) {
                this.workshops.push(response)
                return response
            } else {
                throw new Error(response.error)
            }
        },

        async create (workshop) {
            const { response, ok } = await API.Req('POST', '/api/workshops',
{ body: workshop })
            if (ok) {
                this.workshops.push(response)
            } else {
                throw new Error(response.error)
```

```
            }
        },

        async update (data, id) {
            const { response, ok } = await API.Req('PUT', `/api/workshops/
${id}`, { body: data })
            if (ok) {
                const idx = this.workshops.findIndex(p => p.id === data.id)
                this.workshops[idx] = response
            } else {
                throw new Error(response.error)
            }
        },

        async delete (id) {
            const { response, ok } = await API.Req('DELETE', `/api/workshops/
${id}`)
            if (ok) {
                this.workshops = this.workshops.filter(w => w.id !== id)
            } else {
                throw new Error(response.error)
            }
        },

        search (query) {
            return this.workshops.filter(workshop =>
workshop.name.toLowerCase().includes(query.toLowerCase()))
        }
    }
})
```

frontend/src/style/animations.scss

```
.scale-up-center {
  animation: scale-up-center 0.05s ease-in both;
}

@keyframes scale-up-center {
  0% {
    transform: scale(0.5);
  }
  100% {
    transform: scale(1);
  }
}
```

frontend/src/style/layout.scss

```
$navigation-bar-mobile-size: 4.5rem;
$navigation-bar-desktop-size: 5rem;

.navigation-bar {
  position: fixed;
  bottom: 0;
  left: 0;
  right: 0;
  height: $navigation-bar-mobile-size;

  display: flex;
  background: white;
  border-top: $border-width $border-style $border-color !important;
}

.content {
  @extend .container;
  overflow-y: auto;
  height: calc(100vh - $navigation-bar-mobile-size);

  scrollbar-width: none;
}

.content::-webkit-scrollbar {
  display: none;
}

/** navigation bar on mobile */
@include media-breakpoint-up(sm) {
  .navigation-bar {
    top: 0;
    width: $navigation-bar-desktop-size;
    right: unset;
    height: unset;

    flex-direction: column;
    border-top: unset !important;
    border-right: $border-width $border-style $border-color !important;
  }
}

@include media-breakpoint-up(sm) {
```

```
.content {
    padding-left: $navigation-bar-desktop-size;
    height: 100vh;
  }
}

.box {
  box-shadow: 0 0.5rem 1rem rgba(0, 0, 0, 0.15) !important;
  border-radius: var(--bs-border-radius-2xl) !important;
  overflow: hidden;
  margin: 0;
}

.box-header {
  height: 4.5rem;
}
```

frontend/src/style/styles.scss

```
$body-bg: #fafafa;
$input-focus-border-color: #f49700;
$input-focus-box-shadow: unset;

$theme-colors: (
  'primary': #f49700,
  'success': #28a745,
  'secondary': #ededed,
  'danger': #dc3545,
  'black': #444444,
  'green': #8EA604,
  'red': #DF2935,
  'grey': #666666,
  'light-grey': #CACACA,
  'white': #FFFFFF
);

$container-max-widths: (
        sm: 540px,
        md: 720px,
        lg: 960px
);

@import "~bootstrap/scss/bootstrap";
@import "layout";
@import "animations";

a {
  color: unset !important;
  text-decoration: none !important;
}

.hover-darken:hover {
  background-color: #0002 !important;
}

.card {
  background-color: #ededed;
}

.img {
  background-color: white;
```

```
  border-radius: 4em;
}

.card img {
  width: 100px;
  margin: auto;
  padding: 0.5em;
  border-radius: 0.7em;
}

.overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.5);
  display: flex;
  justify-content: center;
  align-items: center;
}

.centered-rectangle {
  background-color: white;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  border-radius: 2em;
}

.search {
  border-radius: 0;
  border: none;
}
```

frontend/src/util/Api.js

```
export class API {
    /**
     * Req executes a http request and returns the response.
     * @param url http destination url
     * @param token bearer token
     * @param method http method (POST, PUT etc...)
     * @param body body of request
     * @param headers additional headers of request
     * @returns {Promise<{response: *, ok: boolean, status: number}>}
     */
    static async Req (method, url, { body = null, headers = new Headers() } =
{}) {
        if (body) headers.append('Content-Type', 'application/json')
        const response = await fetch(url, {
            headers,
            method,
            body: body ? JSON.stringify(body) : undefined
        })

        const contentType = response.headers.get('content-type')
        return {
            response: contentType && contentType.indexOf('application/json') !
== -1 ? await response.json() : await response.text(),
            ok: response.ok,
            status: response.status
        }
    }
}
```

frontend/src/views/ProductCreate.vue

```
<script setup>
import { useProductStore } from '../store/productStore.js'
import { useRouter } from 'vue-router'
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'
import TextInput from '../component/input/TextInput.vue'
import NumberInput from '../component/input/NumberInput.vue'
import { ref } from 'vue'

const router = useRouter()
const productStore = useProductStore()

const product = ref({
    name: '',
    stock: 0,
    minStock: 0
})

async function create () {
    if (product.value.name === '') throw new Error('name is empty')
    await productStore.create(product.value)
    await router.back()
}
</script>

<template>
  <div class="row box-header">
    <div class="d-flex align-items-center m-0" style="width: min-content">
      <a class="btn p-2 bg-secondary hover-darken" @click="$router.back()">
        <font-awesome-icon :icon="['fas', 'caret-left']" class="fa-xl"
style="width: 24px"/>
      </a>
    </div>

    <div class="col d-flex align-items-center">
      <h3 class="m-0">New Product</h3>
    </div>
  </div>

  <div class="row box bg-white border-top">
    <text-input name="Name" v-model:value="product.name" />
    <number-input name="Stock" v-model:value="product.stock" />
    <number-input name="Minimum Stock" v-model:value="product.minStock" />
```

```
    <button class="m-3 ms-auto btn p-2 bg-primary d-flex justify-content-
center" @click="create" style="width: 10rem">
      <font-awesome-icon :icon="['fas', 'floppy-disk']" class="fa-xl" />
      <span class="h5 m-0 ms-3">Create</span>
    </button>
  </div>
</template>
```

frontend/src/views/ProductDetails.vue

```
<script setup>
import { useProductStore } from '../store/productStore.js'
import { useRoute } from 'vue-router'
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'
import TextInput from '../component/input/TextInput.vue'
import NumberInput from '../component/input/NumberInput.vue'

const route = useRoute()
const productStore = useProductStore()

const productId = Number(route.params.id)
const product = await productStore.get(productId)

async function save () {
    const { id, ...data } = product
    await productStore.update(data, id)
}
</script>

<template>
  <div class="row box-header">
    <div class="d-flex align-items-center m-0" style="width: min-content">
      <a class="btn p-2 bg-secondary hover-darken" @click="$router.back()">
        <font-awesome-icon :icon="['fas', 'caret-left']" class="fa-xl"
style="width: 24px"/>
      </a>
    </div>

    <div class="col d-flex align-items-center">
      <h3 class="m-0">Product Info</h3>
    </div>
  </div>

  <div class="row box bg-white border-top">
    <text-input name="Name" v-model:value="product.name" @update:value="save"/
>
    <number-input name="Stock" v-model:value="product.stock"
@update:value="save"/>
    <number-input name="Minimum Stock" v-model:value="product.minStock"
@update:value="save"/>
  </div>
</template>
```

frontend/src/views/Products.vue

```
<script setup>
import { useProductStore } from '../store/productStore.js'
import ProductItem from '../component/product/ProductItem.vue'
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'
import { ref, computed } from 'vue'

const edit = ref(false)
const productStore = useProductStore()
productStore.fetch()

const search = ref('')
const filteredProducts = computed(() => productStore.search(search.value))

async function remove (product) {
    await productStore.delete(product.id)
}
</script>

<template>
  <div class="row box-header">
    <div class="col-2 d-flex align-items-center">
      <h3 class="m-2">Products</h3>
    </div>

    <div class="col-10 d-flex align-items-center justify-content-end">
      <!-- action buttons -->
      <router-link class="btn p-3 hover-darken" to="/products/new">
        <font-awesome-icon :icon="['fas', 'plus']" class="fa-xl"/>
      </router-link>

      <button class="btn p-3 hover-darken" :class="{'bg-primary': edit}"
@click="edit = !edit">
        <font-awesome-icon :icon="['fas', 'pen-to-square']" class="fa-xl"/>
      </button>
    </div>
  </div>

  <div class="row box bg-white border-top">
    <div class="p-0 input-group align-items-end">
      <input type="text" v-model="search" placeholder="Search products..."
class="form-control search p-4">
      <router-link to="/scan"
```

```
                    class="d-flex justify-content-center align-items-center bg-
primary h-100"
                    style="height: 3rem; width: 5rem; margin-top: -1rem">
          <font-awesome-icon :icon="['fas', 'qrcode']" class="fa-2x"/>
        </router-link>
      </div>

      <!-- product list -->
      <ProductItem
          v-for="product in filteredProducts"
          :key="product.id"
          :product="product"
          :edit="edit"
          @delete="remove"/>
    </div>
</template>
```

frontend/src/views/Scan.vue

```
<script setup>
import { useProductStore } from '../store/productStore.js'
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'
import { StreamBarcodeReader } from 'vue-barcode-reader'
import router from '../router/router.js'

const productStore = useProductStore()
productStore.fetch()

function onDecode (result) {
    const product = productStore.findCode(result)
    if (product) {
        router.push('/products/' + product.id)
    } else {
        throw Error('unknown product')
    }
}
</script>

<template>
  <div class="d-flex justify-content-center">
    <h2 class="pt-4">Scan A QR / Barcode</h2>
    <div class="position-absolute d-flex flex-column">
      <div class="position-relative pe-3">
        <router-link to="/workshops" class="btn float-end hover-darken mt-4
mb-4">
          <font-awesome-icon :icon="['fas', 'x']"/>
        </router-link>
      </div>
      <stream-barcode-reader @decode="onDecode" class="ps-3 pe-3"/>
    </div>
  </div>
</template>
```

frontend/src/views/WorkshopCreate.vue

```
<script setup>
import { useRouter } from 'vue-router'
import { useWorkshopStore } from '../store/workshopStore.js'
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'
import TextInput from '../component/input/TextInput.vue'
import NumberInput from '../component/input/NumberInput.vue'
import { ref } from 'vue'

const router = useRouter()
const workshopStore = useWorkshopStore()

const workshop = ref({
    name: '',
    groupSize: 0
})

async function create () {
    if (workshop.value.name === '') throw new Error('name is empty')
    await workshopStore.create(workshop.value)
    await router.back()
}
</script>

<template>
  <div class="row box-header">
    <div class="d-flex align-items-center m-0" style="width: min-content">
      <a class="btn p-2 bg-secondary hover-darken" @click="$router.back()">
        <font-awesome-icon :icon="['fas', 'caret-left']" class="fa-xl"
style="width: 24px" />
      </a>
    </div>

    <div class="col d-flex align-items-center">
      <h3 class="m-0">New Workshop</h3>
    </div>
  </div>

  <div class="row box bg-white border-top">
    <text-input name="Name" v-model:value="workshop.name" />
    <number-input name="Group size" v-model:value="workshop.groupSize" />

    <button class="m-3 ms-auto btn p-2 bg-primary d-flex justify-content-
```

```
center" @click="create" style="width: 10rem">
      <font-awesome-icon :icon="['fas', 'floppy-disk']" class="fa-xl" />
      <span class="h5 m-0 ms-3">Create</span>
    </button>
  </div>
</template>
```

frontend/src/views/WorkshopDetails.vue

```
<script setup>
import { useRoute } from 'vue-router'
import { useWorkshopStore } from '../store/workshopStore.js'
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'
import TextInput from '../component/input/TextInput.vue'
import NumberInput from '../component/input/NumberInput.vue'

const route = useRoute()
const workshopStore = useWorkshopStore()

const workshopId = Number(route.params.id)
const workshop = await workshopStore.get(workshopId)

async function save () {
    const { id, ...data } = workshop
    await workshopStore.update(data, id)
}
</script>

<template>
  <div class="row box-header">
    <div class="d-flex align-items-center m-0" style="width: min-content">
      <a class="btn p-2 bg-secondary hover-darken" @click="$router.back()">
        <font-awesome-icon :icon="['fas', 'caret-left']" class="fa-xl"
style="width: 24px" />
      </a>
    </div>

    <div class="col d-flex align-items-center">
      <h3 class="m-0">Workshop Info</h3>
    </div>
  </div>

  <div class="row box bg-white border-top">
    <text-input name="Name" v-model:value="workshop.name"
@update:value="save" />
    <number-input name="Group size" v-model:value="workshop.groupSize"
@update:value="save" />
  </div>
</template>
```

frontend/src/views/Workshops.vue

```
<script setup>
import { useWorkshopStore } from '../store/workshopStore.js'
import WorkshopItem from '../component/workshop/WorkshopItem.vue'
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'
import { ref, computed } from 'vue'

const edit = ref(false)
const workshopStore = useWorkshopStore()
workshopStore.fetch()

const search = ref('')
const filteredWorkshops = computed(() => workshopStore.search(search.value))

async function remove (workshop) {
    await workshopStore.delete(workshop.id)
}
</script>

<template>
  <div class="row box-header">
    <div class="col-2 d-flex align-items-center">
      <h3 class="m-2">Workshops</h3>
    </div>

    <div class="col-10 d-flex align-items-center justify-content-end">
      <!-- action buttons -->
      <router-link class="btn p-3 hover-darken" to="/workshops/new">
        <font-awesome-icon :icon="['fas', 'plus']" class="fa-xl"/>
      </router-link>

      <button class="btn p-3 hover-darken" :class="{ 'bg-primary': edit }"
@click="edit = !edit">
        <font-awesome-icon :icon="['fas', 'pen-to-square']" class="fa-xl"/>
      </button>
    </div>
  </div>

  <div class="row box bg-white border-top">
    <div class="p-0 input-group align-items-end">
      <input type="text" v-model="search" placeholder="Search workshops..."
class="form-control search p-4">
      <router-link to="/scan"
```

```
                    class="d-flex justify-content-center align-items-center bg-
primary h-100"
                    style="height: 3rem; width: 5rem; margin-top: -1rem">
          <font-awesome-icon :icon="['fas', 'qrcode']" class="fa-2x"/>
        </router-link>
      </div>

      <!-- workshop list -->
      <WorkshopItem v-for="workshop in filteredWorkshops"
                    :key="workshop.id"
                    :workshop="workshop"
                    :edit="edit"
                    @delete="remove"/>
    </div>
</template>
```

frontend/vite.config.js

```
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'
import * as path from 'path'

// https://vitejs.dev/config/
export default defineConfig({
    plugins: [vue()],
    resolve: {
        alias: {
            '~bootstrap': path.resolve(__dirname, 'node_modules/bootstrap')
        }
    },
    server: {
        proxy: {
            '/api': { target: 'http://localhost:3000' }
        }
    }
})
```