VILNIUSCODINGSCHOOL

Node.js Auth

Startiniai failai

Susikuriame duomenų bazę (node2) ir Collection (users) su Mongo DB, tada app.js faile pridedame savo URL jungtį.

Auth Routes

Naudosime MVC (model, view, controller)

/signup	GET	signup puslapis
/login	GET	login puslapis
/signup	POST	sukurti naują vartotoją DB
/login	POST	autentikuoti dabartinį vartotoją
/logout	POST	atjungti vartotoją

Sukuriame controllers/authController.js

```
module.exports.signup_get = (req, res) => {
    res.render('signup')
}

module.exports.login_get = (req, res) => {
    res.render('login')
}

module.exports.signup_post = (req, res) => {
    res.send('new signup')
}

module.exports.login_post = (req, res) => {
    res.send('user login')
}
```

CommonJS

```
export function signup_get(req, res) {
    res.render('signup')
}

export function login_get(req, res) {
    res.render('login')
}

export function signup_post(req, res) {
    res.send('new signup')
}

export function login_post(req, res) {
    res.send('user login')
}
```

ES moduliai

Controller struktūra bus tokia. Netrukus aprašysime kiekvieną detaliau.

Sukuriame routes/authRoutes.js

```
const {Router} = require('express')
const authController = require('../controllers/authController')

const router = Router()

router.get('/signup', authController.signup_get)
router.post('/signup', authController.signup_post)
router.get('/login', authController.login_get)
router.post('/login', authController.login_post)

module.exports = router
```

Šiems render atvaizdavimams reikalingi nauji view:

```
module.exports.signup_get = (req, res) => {
    res.render('signup')
}
```

```
module.exports.login_get = (req, res) => {
    res.render('login')
}
```

Sukuriame views/signup.ejs ir views/login.ejs

```
<%- include('partials/header') %>

<h1>sign up</h1>

<%- include('partials/footer') %>
```

```
<%- include('partials/header') %>

<h1>login</h1>

<%- include('partials/footer') %>
```

Registruojame routes į app.js

```
const authRoutes = require('./routes/authRoutes')
```

```
// routes
app.get('/', (req, res) => res.render('home'));
app.get('/smoothies', (req, res) => res.render('smoothies'));
app.use(authRoutes)
```

Testuojame:
Naršyklėje turėtų veikti /signup ir /login puslapiai

Reikalingas greitas būdas patikrinti šiuos du:

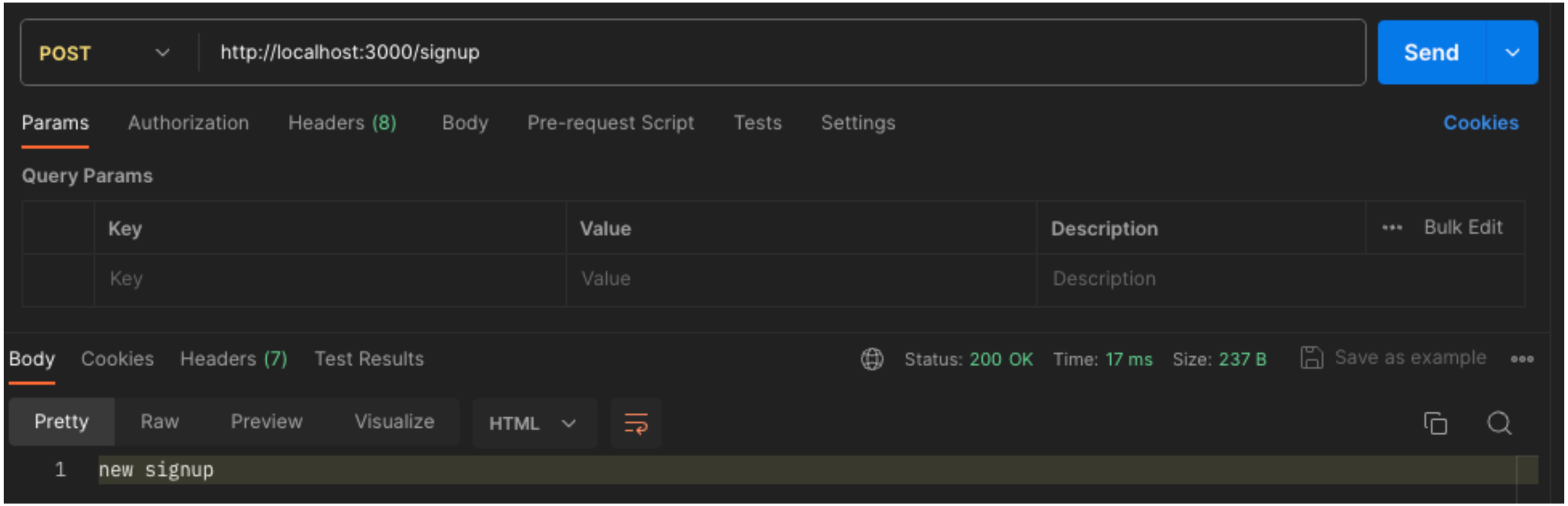
```
module.exports.signup_post = (req, res) => {
    res.send('new signup')
}

module.exports.login_post = (req, res) => {
    res.send('user login')
}
```

Tam galime panaudoti Postman: postman.com

Parsisiunčiame ir instaliuojame Postman Desktop Agent.

Kai Postman Desktop Agent jau yra aktyvus, Postman tinklalapyje testuojame:




```
// middleware
app.use(express.static('public'));
app.use(express.json())
```

app.js

Ima bet kokius json duomenis, kuriuos gauname per užklausą ir juos parsina į JS objektą, kurį galime naudoti kode.

Testuosime ar šie POST veikia tinkamai.

```
module.exports.signup_post = (req, res) => {
    const {email, password} = req.body
    console.log(email, password)
    res.send('new signup')
}

module.exports.login_post = (req, res) => {
    const {email, password} = req.body
    console.log(email, password)
    res.send('user login')
}
```

Postman

POST http://localhost:3000/login Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2     "email": "andrius@digioapps.com",
3     "password": "Test1234"
4 }
5
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 40 ms Size: 237 B Save as example

Pretty Raw Preview Visualize HTML

```
1 user login
```

Terminale: `andrius@digioapps.com Test1234`

Postman

POST http://localhost:3000/signup Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2     "email": "andrius@digioapps.com",
3     "password": "Test1234"
4 }
5
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 5 ms Size: 237 B Save as example

Pretty Raw Preview Visualize HTML

```
1 new signup
```

Terminale: `andrius@digioapps.com Test1234`

Naujų vartotojų kūrimas DB

Sukuriame models/User.js

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

const userSchema = new mongoose.Schema({
    email: {
        type: String,
        required: true,
        unique: true,
        lowercase: true
    },
    password: {
        type: String,
        required: true,
        minlength: 6
    }
})

// parametras turėtų būti vienaskaita mūsų Collection pavadinimo Mongo DB
const User = mongoose.model('user', userSchema)

module.exports = User
```

Naujų vartotojų kūrimas DB

authController.js

```
module.exports.signup_post = async (req, res) => {
    const {email, password} = req.body
    try {
        const user = await User.create({email, password})
        res.status(201).json(user)
    } catch (err) {
        console.log(err)
        res.status(400).send('Error: user not created')
    }
}
```

Naujų vartotojų kūrimas DB

Testuojame su Postman.

POST http://localhost:3000/signup Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

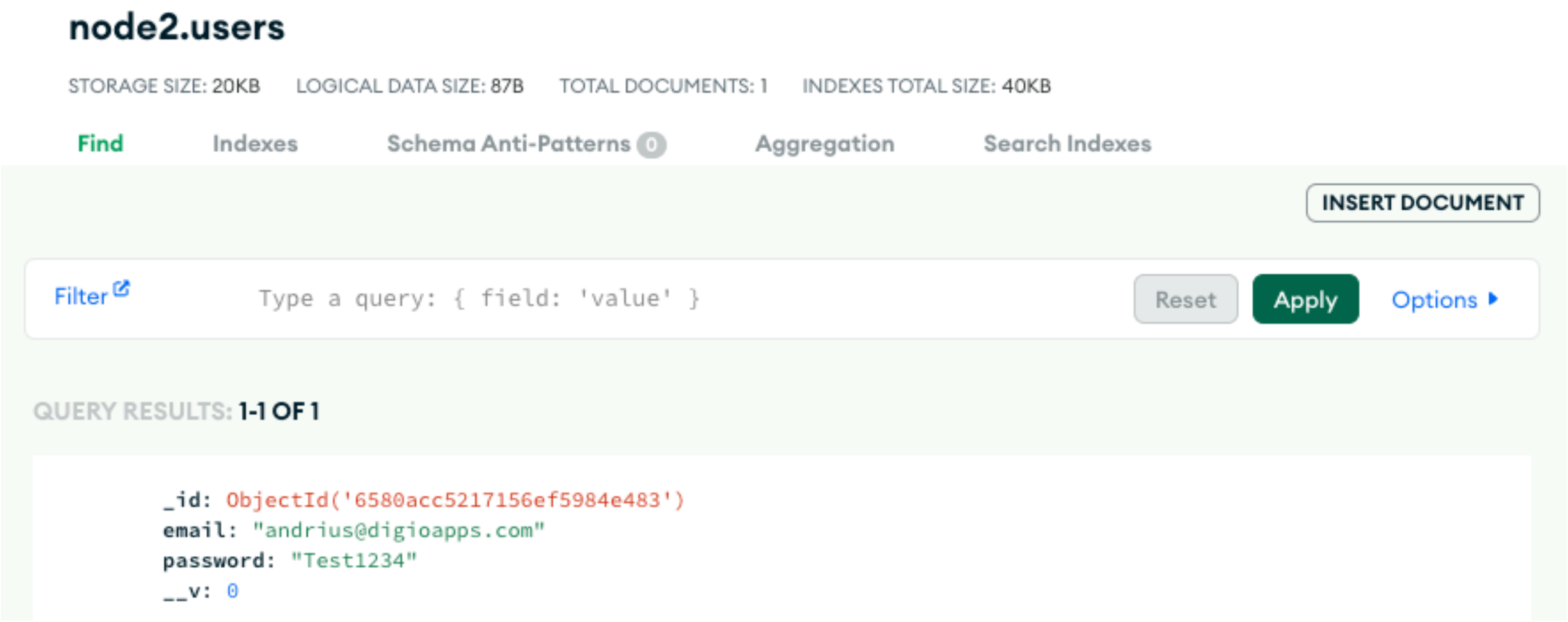
```
1 {
2     "email": "andrius@digioapps.com",
3     "password": "Test1234"
4 }
5
```

Body Cookies Headers (7) Test Results Status: 201 Created Time: 109 ms Size: 336 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2     "_id": "6580acc5217156ef5984e483",
3     "email": "andrius@digioapps.com",
4     "password": "Test1234",
5     "__v": 0
6 }
```

Naujų vartotojų kūrimas DB



Slaptažodžiai turėtų būti šifruojami. Padarysime tai vėliau.

Tikslesnis klaidų atvaizdavimas - User.js

Visos klaidos turėjo vienodą pranešimą: Error: user not created

El. pašto validavimui: npm install validator

```
const mongoose = require('mongoose')
const { isEmail } = require('validator')


const userSchema = new mongoose.Schema({
  email: {
    type: String,
    required: [true, 'Please enter an email'],
    unique: true,
    lowercase: true,
    validate: [isEmail, 'Please enter a valid email']
  },
  password: {
    type: String,
    required: [true, 'Please enter a password'],
    minlength: [6, 'Minimum password length is 6 characters']
  }
})
```

```
// parametras turėtų būti vienaskaita mūsų Collection pavadinimo Mongo DB
const User = mongoose.model('user', userSchema)

module.exports = User
```

Tikslesnis klaidų atvaizdavimas - authController.js

```
const User = require('../models/User')
...

const handleErrors = (err) => {
    let errors = {email: '', password: ''}
    if(err.code === 11000) {
        errors.email = 'That email is already registered'
        return errors
    }

    if (err.message.includes('user validation failed')) {
        Object.values(err.errors).forEach(({properties}) => {
            errors[properties.path] = properties.message
        })
    }
    return errors
}
```

```
module.exports.signup_post = async (req, res) => {
    const {email, password} = req.body
    try {
        const user = await User.create({email, password})
        res.status(201).json(user)
    } catch (err) {
        const errors = handleErrors(err)
        res.status(404).json({errors})
    }
}
```

Mongoose hooks - slaptažodžių šifravimas

npm install bcrypt

hAjfg Test1234 — Šifravimo algoritmas — $d8skv;v...

Salt

```
const bcrypt = require('bcrypt')
```

User.js

```
// vykdome funkciją prieš išsaugant į db
userSchema.pre('save', async function(next) {
    const salt = await bcrypt.genSalt()
    this.password = await bcrypt.hash(this.password, salt)
    next()
})
```

POST http://localhost:3000/signup Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "email": "amber@digioapps.com",
    "password": "Test1234"
}
```

Body Cookies Headers (7) Test Results Status: 201 Created Time: 198 ms Size: 387 B Save as example

Pretty Raw Preview Visualize JSON

```
{
    "_id": "6580bca593f744f382758323",
    "email": "amber@digioapps.com",
    "password": "$2b$10$.SDYbgVoudQI3VdCbHiMuelJ/OJ35ylGO4HNVnaariYwSE7APmXLC",
    "__v": 0
}
```

```
_id: ObjectId('6580bca593f744f382758323')
email: "amber@digioapps.com"
password: "$2b$10$.SDYbgVoudQI3VdCbHiMuelJ/OJ35ylGO4HNVnaariYwSE7APmXLC"
__v: 0
```

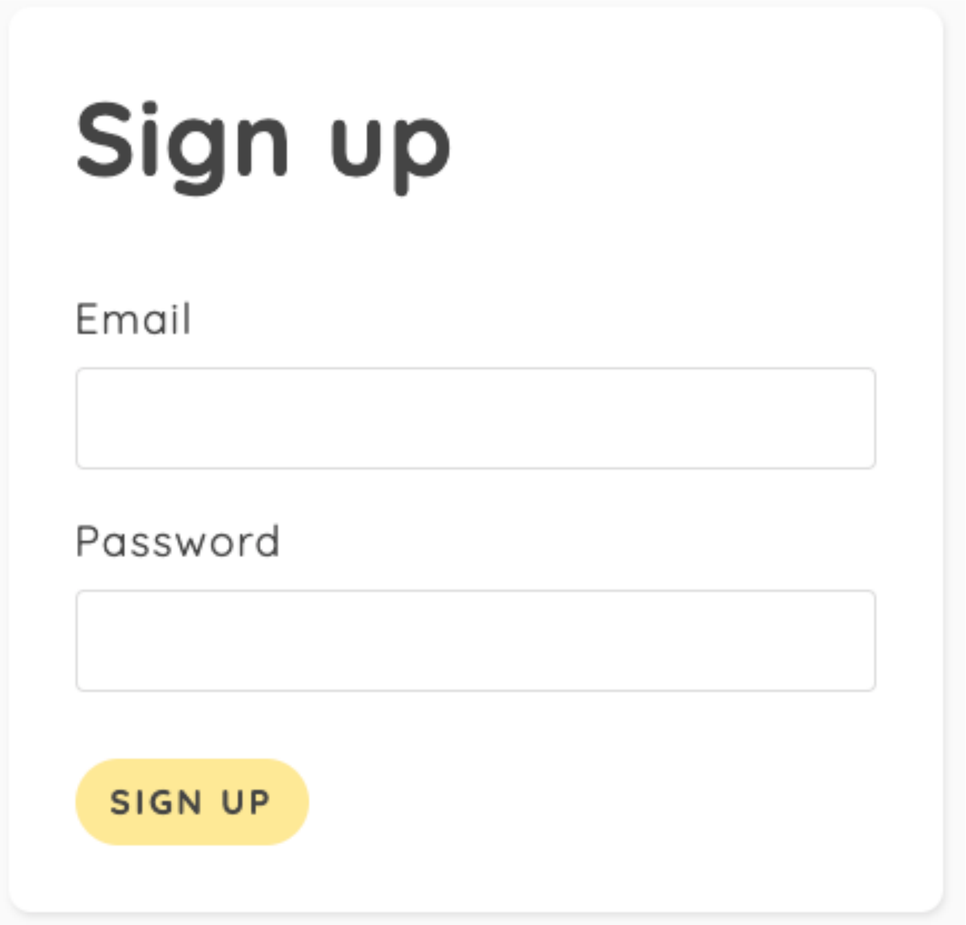
signup.ejs

```
<%- include('partials/header') %>

<form>
    <h2>Sign up</h2>
    <label for="email">Email</label>
    <input type="email" name="email" required>
    <div class="email error"></div>
    <label for="password">Password</label>
    <input type="password" name="password" required>
    <div class="password error"></div>
    <button>Sign up</button>
</form>

<script>
    const form = document.querySelector('form')
    form.addEventListener('submit', (e) => {
        e.preventDefault()
        const email = form.email.value
        const password = form.password.value
    })
</script>

<%- include('partials/footer') %>
```



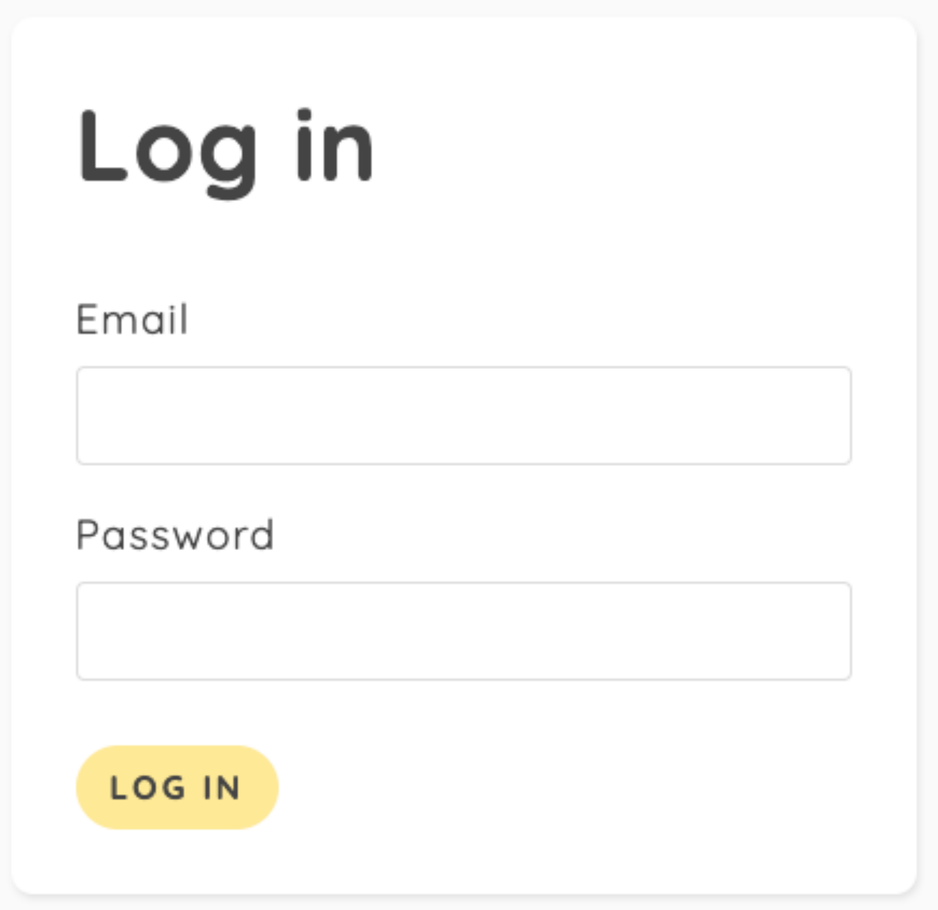
login.ejs

```
<%- include('partials/header') %>

<form>
    <h2>Log in</h2>
    <label for="email">Email</label>
    <input type="email" name="email" required>
    <div class="email error"></div>
    <label for="password">Password</label>
    <input type="password" name="password" required>
    <div class="password error"></div>
    <button>Log in</button>
</form>

<script>
    const form = document.querySelector('form')
    form.addEventListener('submit', (e) => {
        e.preventDefault()
        const email = form.email.value
        const password = form.password.value
    })
</script>

<%- include('partials/footer') %>
```



Cookies

Duomenų išsaugojimas vartotojo naršyklėje

name=amber, age=18, isStudent=true

Kaskart, kai vartotojas daro užklausą į serverį, cookies duomenys keliauja kartu su užklausa.

npm install cookie-parser

(po šio pvz, ištriname)

```
const cookieParser = require('cookie-parser')
```

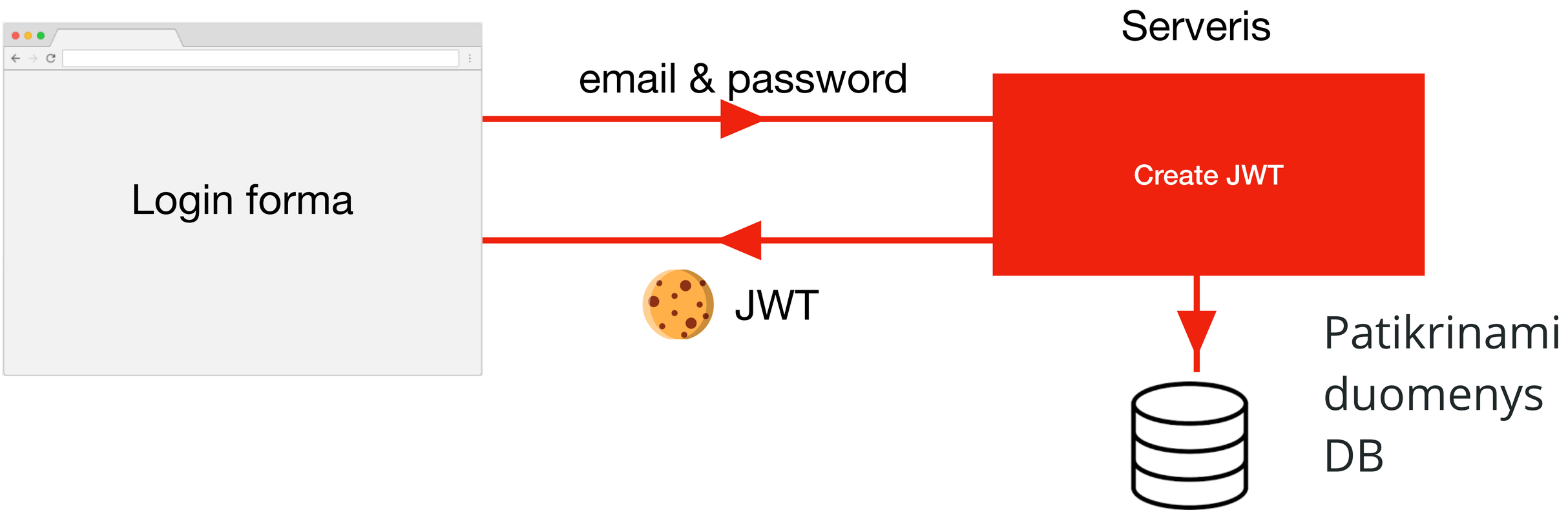
```
// middleware
app.use(express.static('public'))
app.use(express.json())
app.use(cookieParser())
```

app.js

```
// cookies
app.get('/set-cookies', (req, res) => {
    res.cookie('newUser', false)
    res.cookie('isStudent', true, {maxAge: 1000 * 60 * 60 * 24, secure: true})
    res.send('You got cookies!')
})

app.get('/read-cookies', (req, res) => {
    const cookies = req.cookies
    res.json(cookies)
})
```

JSON Web Tokens (JWT)



JSON Web Tokens (JWT)

Headers

Pasako serveriui, kokio tipo parašas yra naudojamas (meta)

Payload

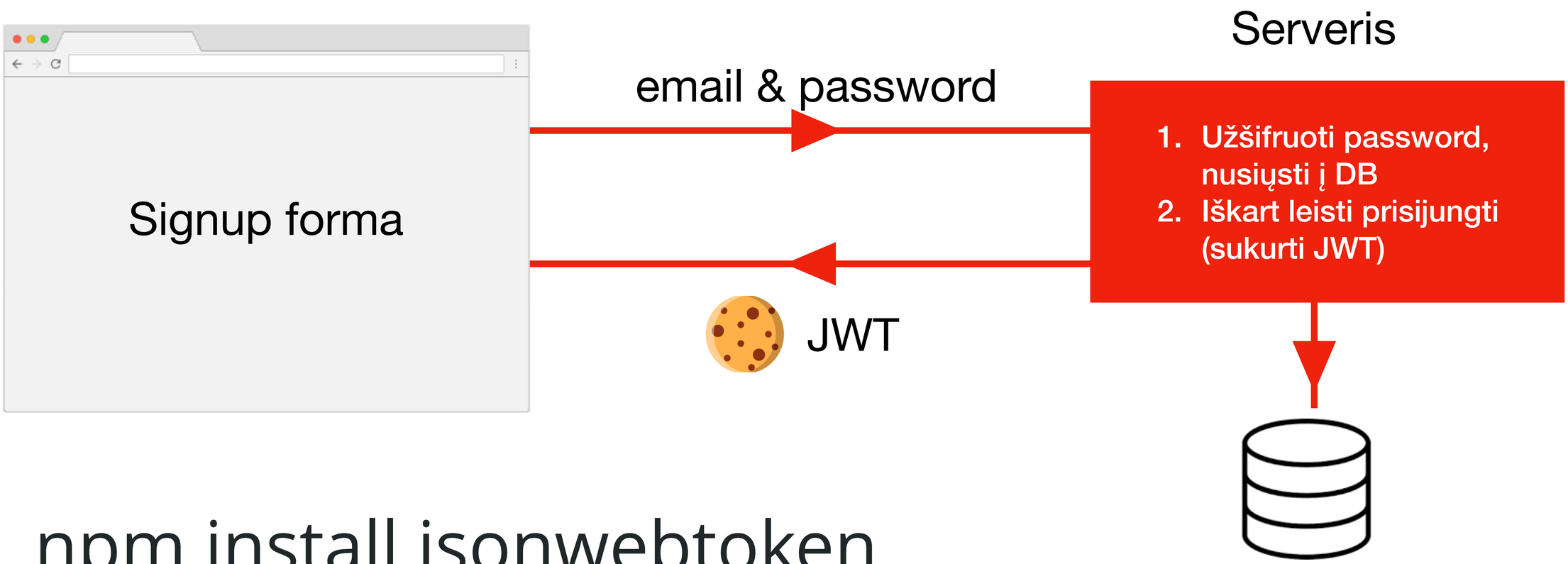
Naudojama vartotojo identifikavimui (pvz., talpina user id)

Signature

Padaro token saugiu (panašiai kaip autentikavimo spaudas)

Rezultatas: šių trijų dalių junginys, suformuojantis token

JSON Web Tokens (JWT): ką darysim



npm install jsonwebtoken

signup.ejs

```
<script>
    const form = document.querySelector('form')
    form.addEventListener('submit', async (e) => {
        e.preventDefault()
        const email = form.email.value
        const password = form.password.value
        try {
            const res = await fetch('/signup', {
                method: 'POST',
                body: JSON.stringify({email, password}),
                headers: {'Content-Type': 'application/json'}
            })
        } catch(err) {
            console.log(err)
        }
    })
</script>
```

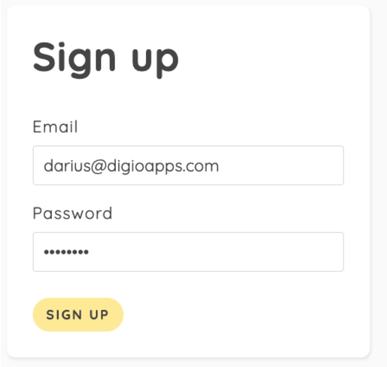
authController.js

```
const jwt = require('jsonwebtoken')
```

```
const maxAge = 3 * 24 * 60 * 60 // trys dienos
const createToken = (id) => {
    return jwt.sign({id}, 'slaptas dalykas', {
        expiresIn: maxAge
    })
}
```

```
module.exports.signup_post = async (req, res) => {
    const {email, password} = req.body
    try {
        const user = await User.create({email, password})
        const token = createToken(user._id)
        res.cookie('jwt', token, {httpOnly: true, maxAge: maxAge * 1000})
        res.status(201).json({user: user._id})
    } catch (err) {
        const errors = handleErrors(err)
        res.status(404).json({errors})
    }
}
```

Testuojame:



Elements | Console | Sources | Network | Performance | Memory | Application | Security | Lighthouse | Recorder | Performance insights

Application
- Manifest
- Service workers
- Storage

Storage
- Local storage
  - http://localhost:3000
- Session storage
- IndexedDB
- Web SQL
- Cookies
  - http://localhost:3000
- Private state tokens
- Interest groups
- Shared storage
- Cache storage

Background services
- Back/forward cache
- Background fetch
- Background sync
- Bounce tracking mitigations
- Notifications
- Payment handler

Filter | Only show cookies with an issue

Name	Value	Domain	Path	Expires /...	Size	HttpOnly	Secure	SameSite	Partition...	Priority
jwt	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY1OD...	localhost	/	2023-12...	174	✓				Medium

Cookie Value | Show URL-decoded

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY1ODBkNjcxMWRmODdmZmIzMTBhNDg3OCIsImlhdCI6MTcwMjk0MjMyMSwiZXhwIjoxNzAzMjAxNTIxfQ.YMhH4mcAOnjliD2cNrm6QnEQ5g5bj3pHyn7emFv4DaE

signup.ejs:

```
<script>
    const form = document.querySelector('form')
    const emailError = document.querySelector('.email.error')
    const passwordError = document.querySelector('.password.error')

    form.addEventListener('submit', async (e) => {
      e.preventDefault()

      // reset errors
      emailError.textContent = ''
      passwordError.textContent = ''

      // get values
      const email = form.email.value
      const password = form.password.value
      try {
        const res = await fetch('/signup', {
          method: 'POST',
          body: JSON.stringify({ email, password }),
          headers: {'Content-Type': 'application/json'}
        })
        const data = await res.json()
        console.log(data)
        if (data.errors) {
          emailError.textContent = data.errors.email
          passwordError.textContent = data.errors.password
        }
        if (data.user) {
          location.assign('/')
        }

      }
      catch (err) {
        console.log(err)
      }

    })
</script>
```

Testuojame:

Užpildžius signup formą, turėtų rodyti klaidas naršyklėje.

Sėkmingai prisijungus, turėtų redirect į pradinį puslapį.

Turėtų sugeneruoti JWT token.

login.ejs

Kopijuojame visą script žymę iš signup.ejs į login.ejs, tik pakeičiame fetch į login

User.js

```
// statiškas metodas vartotojo prisijungimui
userSchema.statics.login = async function(email, password) {
    const user = await this.findOne({ email })
    if (user) {
      const auth = await bcrypt.compare(password, user.password)
      if (auth) {
        return user
      }
      throw Error('incorrect password')
    }
    throw Error('incorrect email')
}
```

authController.js

```
if (err.message === 'incorrect email') {
    errors.email = 'That email is not registered'
}

if (err.message === 'incorrect password') {
    errors.password = 'That password is incorrect'
}
```

```
module.exports.login_post = async (req, res) => {
    const { email, password } = req.body

    try {
        const user = await User.login(email, password)
        const token = createToken(user._id)
        res.cookie('jwt', token, { httpOnly: true, maxAge: maxAge * 1000 })
        res.status(200).json({ user: user._id })
    } catch (err) {
        const errors = handleErrors(err)
        res.status(400).json({ errors })
    }
}
```

Testuojame login:

Rodo klaidas abiems laukeliams.

Prisijungus, nukreipia į pradinį puslapį.

header.ejs

```
<h1><a href="/">Yummy Smoothies</a></h1>
<ul>
    <li><a href="/login">Log in</a></li>
    <li><a href="/signup" class="btn">Sign up</a></li>
</ul>
```

Route apsauga - JWT panaudojimas

Norime, kad /smoothies puslapis būtų pasiekiamas tik prisijungusiems vartotojams. Jei vartotojas turi JWT ir yra tinkamas, galime jam rodyti tą puslapį. Jei neturi JWT, tuomet redirect į Login.

Sukuriame middleware/authMiddleware.js

```
const jwt = require('jsonwebtoken')

const requireAuth = (req, res, next) => {
    const token = req.cookies.jwt
    // tikriname ar JWT egzistuoja ir yra validus
    if (token) {
        jwt.verify(token, 'slaptas dalykas', (err, decodedToken) => {
            if(err) {
                console.log(err.message)
                res.redirect('/login')
            } else {
                console.log(decodedToken)
                next()
            }
        })
    } else {
        res.redirect('/login')
    }
}

module.exports = {requireAuth}
```

app.js

```
const {requireAuth} = require('./middleware/authMiddleware')
```

```
// routes
app.get('/', (req, res) => res.render('home'))
app.get('/smoothies', requireAuth, (req, res) => res.render('smoothies'))
app.use(authRoutes)
```

Testuojame:

1. Ištrynus JWT ir paspaudus ant View Recipes, turėtume atsidurti tiek Login forma.
2. Prisijungus, turėtume atsidurti /smoothies puslapyje
3. Turėtume matyti savo JWT.

Atsijungimas

Kol kas vienintelis būdas atsijungti - ištrinti JWT.

header.ejs

```
<h1><a href="/">Yummy Smoothies</a></h1>
<ul>
    <li><a href="/logout">Logout</a></li>
    <li><a href="/login">Log in</a></li>
    <li><a href="/signup" class="btn">Sign up</a></li>
</ul>
```

authRoutes.js

```
router.get('/logout', authController.logout_get)
```

authController.js

```
module.exports.logout_get = async (req, res) => {
    res.cookie('jwt', '', {maxAge: 1}) // pakeičiame JWT į tuščią eilutę vienai milisekundei
    res.redirect('/')
}
```

Testuojame:

Paspaudus Logout, turėtų pašalinti JWT ir nuvesti mus į pradinį puslapį.

authMiddleware.js

Norime padaryti taip, kad prisijungusio vartotojo duomenys galiotų visuose puslapiuose, kol jis neatsijungs.

```
// tikriname dabartinį vartotoją
const checkUser = (req, res, next) => {
    const token = req.cookies.jwt
    if (token) {
        jwt.verify(token, 'slaptas dalykas', async (err, decodedToken) => {
        if (err) {
            res.locals.user = null
            next()
        } else {
            let user = await User.findById(decodedToken.id)
            res.locals.user = user
            next()
        }
      })
    } else {
        res.locals.user = null
        next()
    }
}

module.exports = {requireAuth, checkUser}
```

app.js

```
const {requireAuth, checkUser} = require('./middleware/authMiddleware')
```

```
// routes
app.get('*', checkUser);
app.get('/', (req, res) => res.render('home'))
app.get('/smoothies', requireAuth, (req, res) => res.render('smoothies'))
app.use(authRoutes)
```

authMiddleware.js

```
const User = require('../models/User')
```

header.ejs

```
<ul>
    <% if (user) { %>
        <li>Welcome, <%= user.email %></li>
        <li><a href="/logout">Log out</a></li>
    <% } else { %>
        <li><a href="/login">Log in</a></li>
        <li><a href="/signup" class="btn">Sign up</a></li>
    <% } %>
</ul>
```