Kristine Canete **IT - INTPROG32** **BSIT - 3**

Node.JS + MySQL - Boilerplate API with Email Sign Up & Verification, Authentication & Forgot Password

Objectives

The project aims to build a boilerplate sign up and authentication API with Node.js and MySQL that includes:

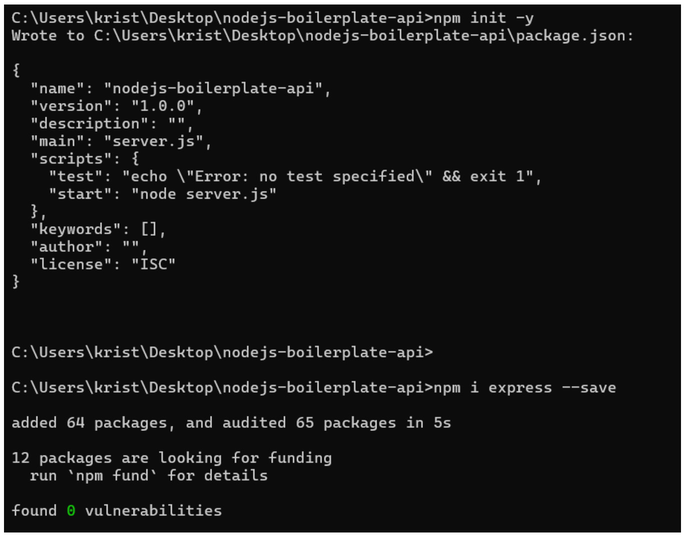
- Email sign up and verification
- JWT authentication with refresh tokens
- Role based authorization with support for two roles (User & Admin)
- Forgot password and reset password functionality
- Account management (CRUD) routes with role based access control
- Swagger api documentation route

Required Tools

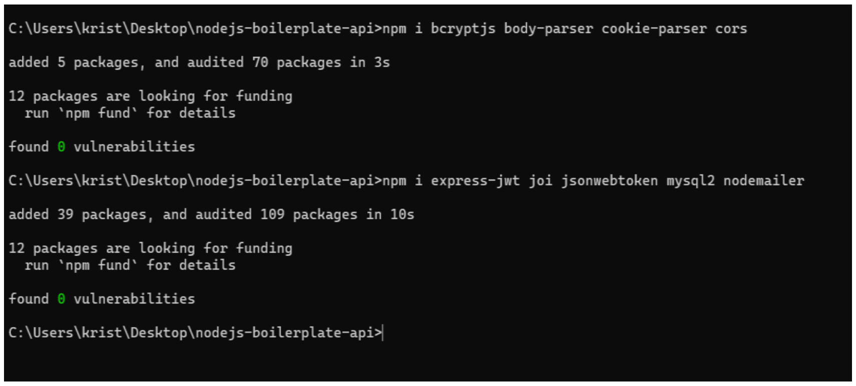
- NodeJS - an open source server environment. It allows us to run JavaScript on the server.
- Visual Studio Code -A code editor to view and edit the API code
- MySQL - an instance for the API to connect to and store data.
- Ethereal - a fake SMTP service, mostly aimed at Nodemailer and EmailEngine users. We're using it to randomly generate emails and test our API's authentication capability.
- Git and GitHub - to initialize and save our API project remotely.

Initializing the Project

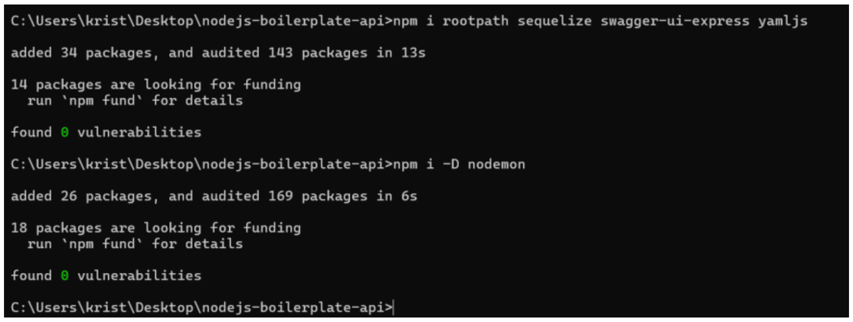
- Create a new folder from your local machine and name it however you want.
- Within the folder directory run the terminal and initialize the node package manager by typing in npm init
- Then after that follow the rest of the process below to install the required dependencies.



```
C:\Users\krist\Desktop\nodejs-boilerplate-api>npm init -y
Wrote to C:\Users\krist\Desktop\nodejs-boilerplate-api\package.json:

{
  "name": "nodejs-boilerplate-api",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}



C:\Users\krist\Desktop\nodejs-boilerplate-api>

C:\Users\krist\Desktop\nodejs-boilerplate-api>npm i express --save

added 64 packages, and audited 65 packages in 5s

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```



```
C:\Users\krist\Desktop\nodejs-boilerplate-api>npm i bcryptjs body-parser cookie-parser cors

added 5 packages, and audited 70 packages in 3s

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\krist\Desktop\nodejs-boilerplate-api>npm i express-jwt joi jsonwebtoken mysql2 nodemailer

added 39 packages, and audited 109 packages in 10s

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\krist\Desktop\nodejs-boilerplate-api>
```

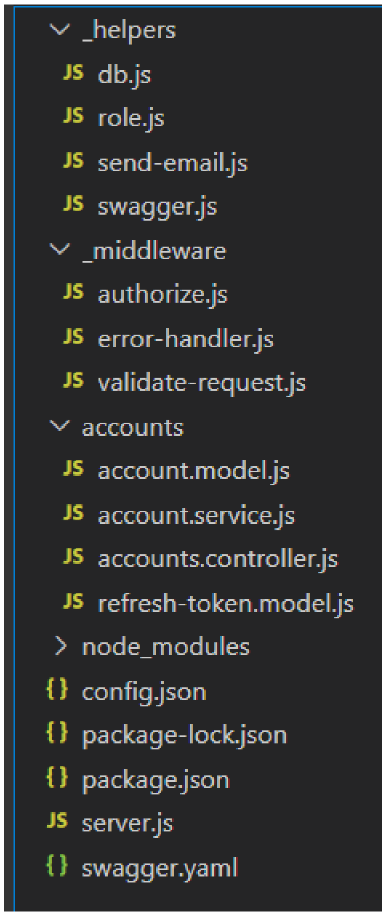


```
C:\Users\krist\Desktop\nodejs-boilerplate-api>npm i rootpath sequelize swagger-ui-express yamljs

added 34 packages, and audited 143 packages in 13s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\krist\Desktop\nodejs-boilerplate-api>npm i -D nodemon

added 26 packages, and audited 169 packages in 6s

18 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\krist\Desktop\nodejs-boilerplate-api>
```

Project Structure

The Project is composed of multiple files and directories, each serving their own purpose.

- Feature folders (accounts)
- Non-feature/Shared Component folders (_helpers, _middleware)
- Config.json
- Server.js
- Swagger.yaml



```
_helpers
    db.js
    role.js
    send-email.js
    swagger.js
_middleware
    authorize.js
    error-handler.js
    validate-request.js
accounts
    account.model.js
    account.service.js
    accounts.controller.js
    refresh-token.model.js
node_modules
config.json
package-lock.json
package.json
server.js
swagger.yaml
```

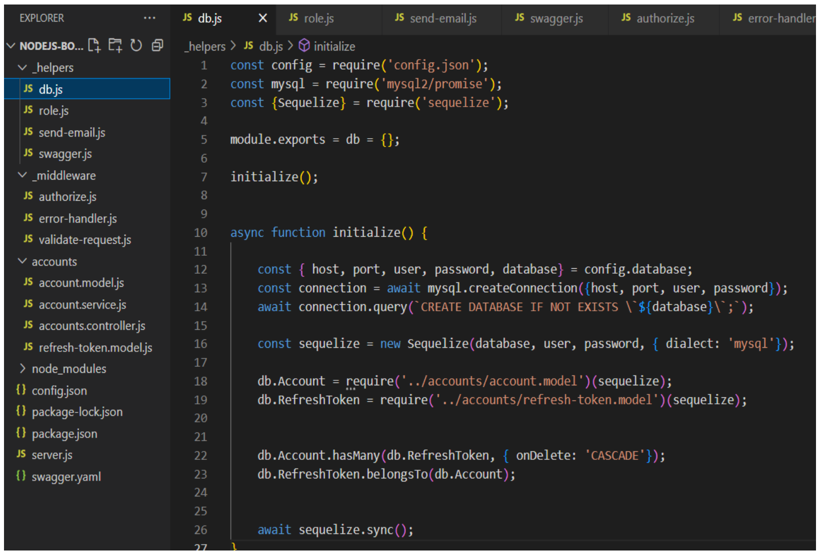
-Path: /_helpers Contents :
- db.js
- role.js
- send-email.js
- swagger.js

-Path: /_middleware Contents :
- authorize.js
- error-handler.js
- validate-request.js

-Path: /_accounts Contents :
- account.model.js
- account.service.js
- accounts.controller.js
- refresh-token.model.js

-Config.json
-Package.json
-server.js
-swagger.yaml

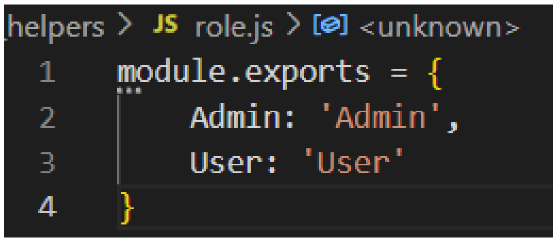
MySQL Database Wrapper (*Path: /_helpers/db.js*)



```javascript
const config = require('config.json');
const mysql = require('mysql2/promise');
const {Sequelize} = require('sequelize');

module.exports = db = {};

initialize();


async function initialize() {

    const { host, port, user, password, database} = config.database;
    const connection = await mysql.createConnection({host, port, user, password});
    await connection.query(`CREATE DATABASE IF NOT EXISTS \`${database}\`;`);

    const sequelize = new Sequelize(database, user, password, { dialect: 'mysql'});

    db.Account = require('../accounts/account.model')(sequelize);
    db.RefreshToken = require('../accounts/refresh-token.model')(sequelize);


    db.Account.hasMany(db.RefreshToken, { onDelete: 'CASCADE'});
    db.RefreshToken.belongsTo(db.Account);


    await sequelize.sync();
}
```

Connects to MySQL using Sequelize to handle functions like handling database records by representing the data as objects.

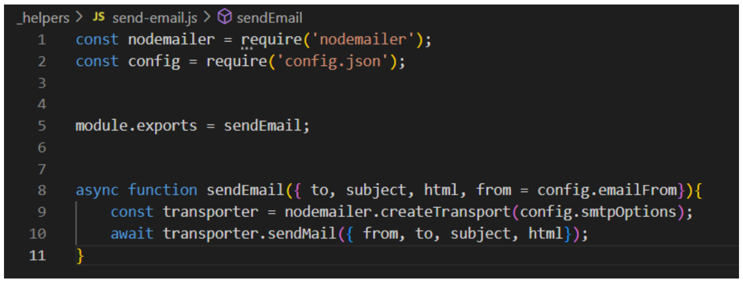
Role Object / Enum (*Path: /_helpers/role.js*)



```
helpers > role.js > <unknown>
module.exports = {
    Admin: 'Admin',
    User: 'User'
}
```

```
Defines all the roles in
the project application.
Using it as an enum to
avoid passing roles as
strings. Therefore we use
it as Role.Admin or
Role.User
```

Send Email Helper (*Path: /_helpers/send-email.js*)

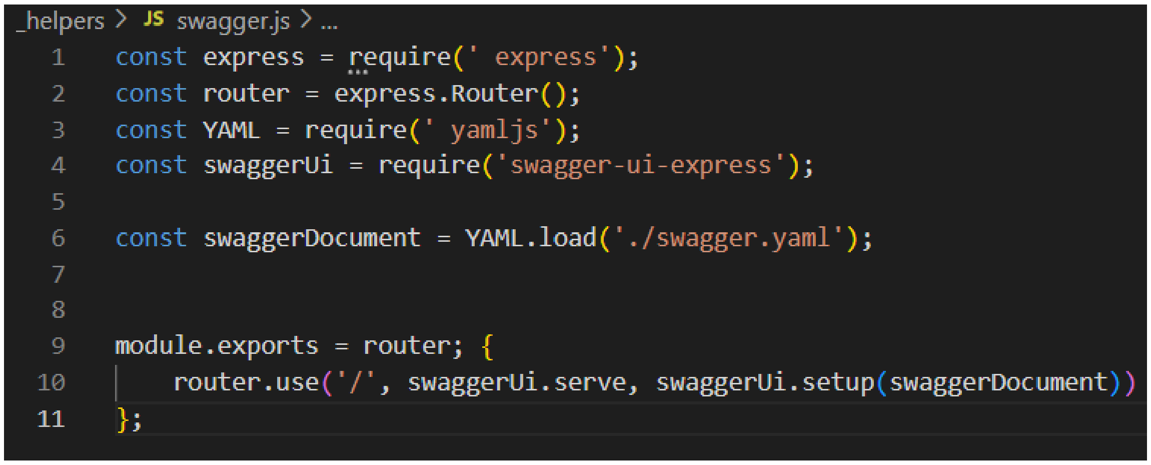


```
_helpers > send-email.js > sendEmail
const nodemailer = require('nodemailer');
const config = require('config.json');


module.exports = sendEmail;


async function sendEmail({ to, subject, html, from = config.emailFrom}){
    const transporter = nodemailer.createTransport(config.smtpOptions);
    await transporter.sendMail({ from, to, subject, html});
}
```

```
Simplifies sending
emails in the
application.
Used by the account
service to send account
verification and
password reset emails.
```

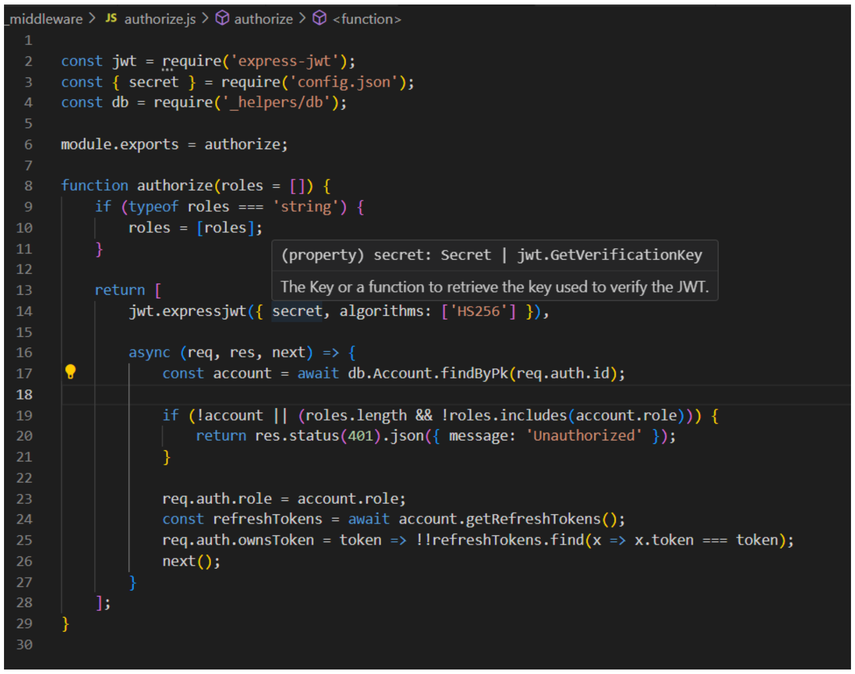
Swagger API Docs Route Handler (*Path: /_helpers/swagger.js*)

```
Auto-generates Swagger UI documentation based
on the swagger.yaml file from the /api-docs
path of the api.
```



```
_helpers > swagger.js > ...
const express = require(' express');
const router = express.Router();
const YAML = require(' yamljs');
const swaggerUi = require('swagger-ui-express');

const swaggerDocument = YAML.load('./swagger.yaml');


module.exports = router; {
    router.use('/', swaggerUi.serve, swaggerUi.setup(swaggerDocument))
};
```

Authorize Middleware (*Path: /_middleware/authorize.js*)



_middleware > authorize.js > authorize > <function>

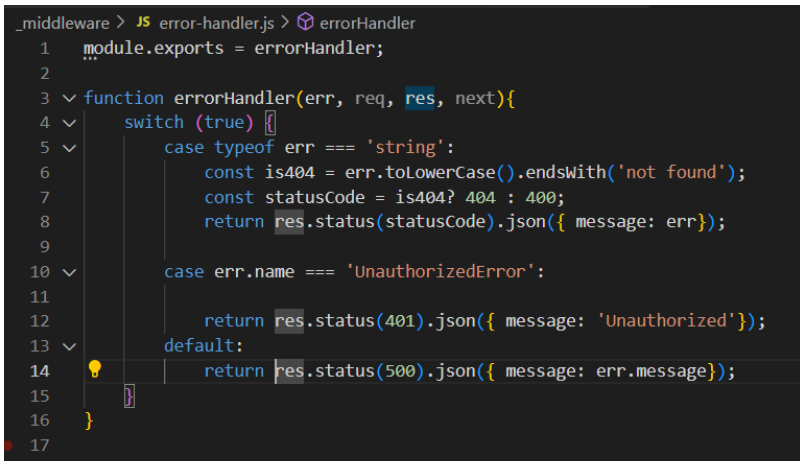
```js

const jwt = require('express-jwt');
const { secret } = require('config.json');
const db = require('_helpers/db');

module.exports = authorize;

function authorize(roles = []) {
    if (typeof roles === 'string') {
        roles = [roles];
    }

    return [
        jwt.expressjwt({ secret, algorithms: ['HS256'] }),

        async (req, res, next) => {
            const account = await db.Account.findByPk(req.auth.id);

            if (!account || (roles.length && !roles.includes(account.role))) {
                return res.status(401).json({ message: 'Unauthorized' });
            }

            req.auth.role = account.role;
            const refreshTokens = await account.getRefreshTokens();
            req.auth.ownsToken = token => !!refreshTokens.find(x => x.token === token);
            next();
        }
    ];
}

```

(property) secret: Secret | jwt.GetVerificationKey

The Key or a function to retrieve the key used to verify the JWT.

Added to restrict access to any route which only authenticated users with specified roles can access. It is used by the accounts controller to handle authorization to CRUD routes as well as revoke token routes.

Global Error Handler Middleware (*Path:/_middleware/error-handler.js*)

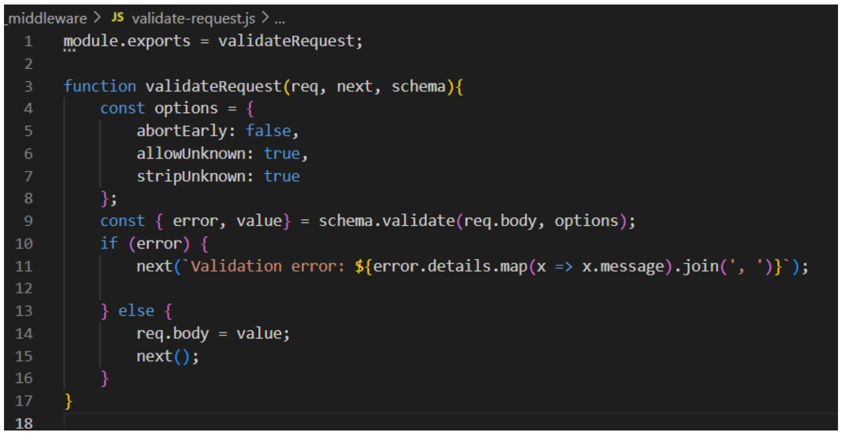


_middleware > error-handler.js > errorHandler

```js
module.exports = errorHandler;

function errorHandler(err, req, res, next){
    switch (true) {
        case typeof err === 'string':
            const is404 = err.toLowerCase().endsWith('not found');
            const statusCode = is404? 404 : 400;
            return res.status(statusCode).json({ message: err});

        case err.name === 'UnauthorizedError':

            return res.status(401).json({ message: 'Unauthorized'});
        default:
            return res.status(500).json({ message: err.message});
    }
}

```

Catches all errors and removes the need for duplicated error handling code throughout the boilerplate application.

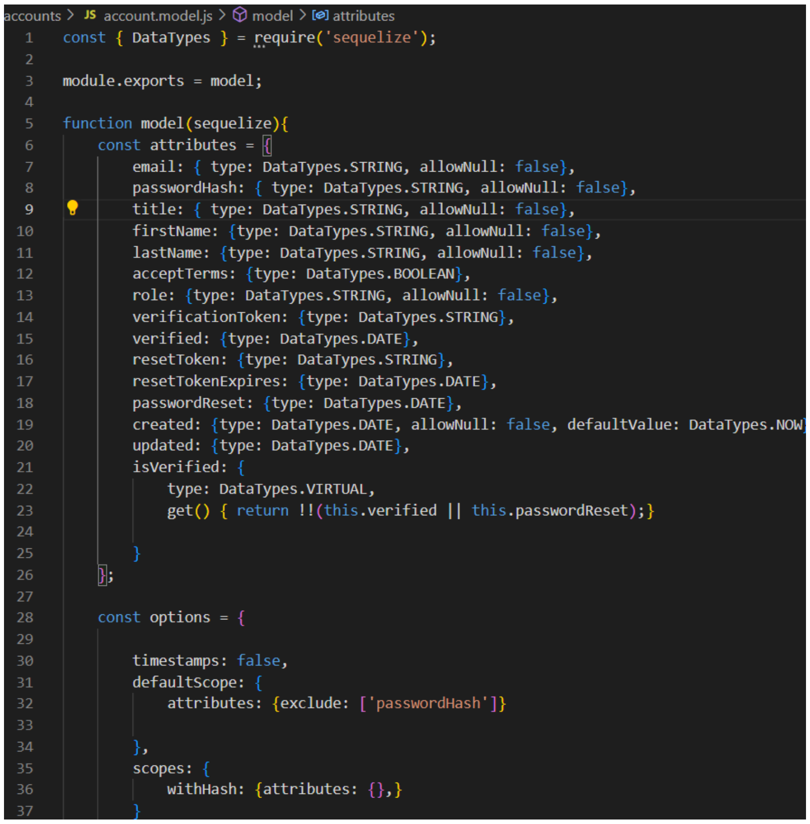
Validate Request Middleware (*Path:/_middleware/validate-request.js*)



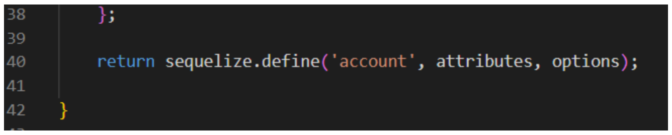
```javascript
module.exports = validateRequest;

function validateRequest(req, next, schema){
    const options = {
        abortEarly: false,
        allowUnknown: true,
        stripUnknown: true
    };
    const { error, value} = schema.validate(req.body, options);
    if (error) {
        next(`Validation error: ${error.details.map(x => x.message).join(', ')}`);

    } else {
        req.body = value;
        next();
    }
}

```

Validates the body of a request against a Joi schema object. Used by the accounts controller.

Sequelize Account Model (*Path:/accounts/account.model.js*)



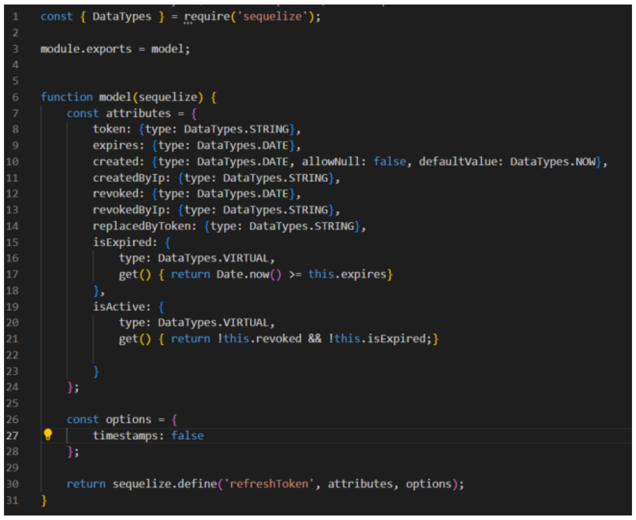
```javascript
const { DataTypes } = require('sequelize');

module.exports = model;

function model(sequelize){
    const attributes = {
        email: { type: DataTypes.STRING, allowNull: false},
        passwordHash: { type: DataTypes.STRING, allowNull: false},
        title: { type: DataTypes.STRING, allowNull: false},
        firstName: {type: DataTypes.STRING, allowNull: false},
        lastName: {type: DataTypes.STRING, allowNull: false},
        acceptTerms: {type: DataTypes.BOOLEAN},
        role: {type: DataTypes.STRING, allowNull: false},
        verificationToken: {type: DataTypes.STRING},
        verified: {type: DataTypes.DATE},
        resetToken: {type: DataTypes.STRING},
        resetTokenExpires: {type: DataTypes.DATE},
        passwordReset: {type: DataTypes.DATE},
        created: {type: DataTypes.DATE, allowNull: false, defaultValue: DataTypes.NOW
        updated: {type: DataTypes.DATE},
        isVerified: {
            type: DataTypes.VIRTUAL,
            get() { return !!(this.verified || this.passwordReset);}

        }
    };

    const options = {

        timestamps: false,
        defaultScope: {
            attributes: {exclude: ['passwordHash']}

        },
        scopes: {
            withHash: {attributes: {},}
        }
```



```javascript
    };

    return sequelize.define('account', attributes, options);

}
```

Uses Sequelize to define the schema for the accounts table in the MySQL database. The exported Sequelize model object gives full access to perform CRUD operations on accounts in MySQL.

Sequelize Refresh Token Model
(*Path:/accounts/refresh-token.model.js***)**

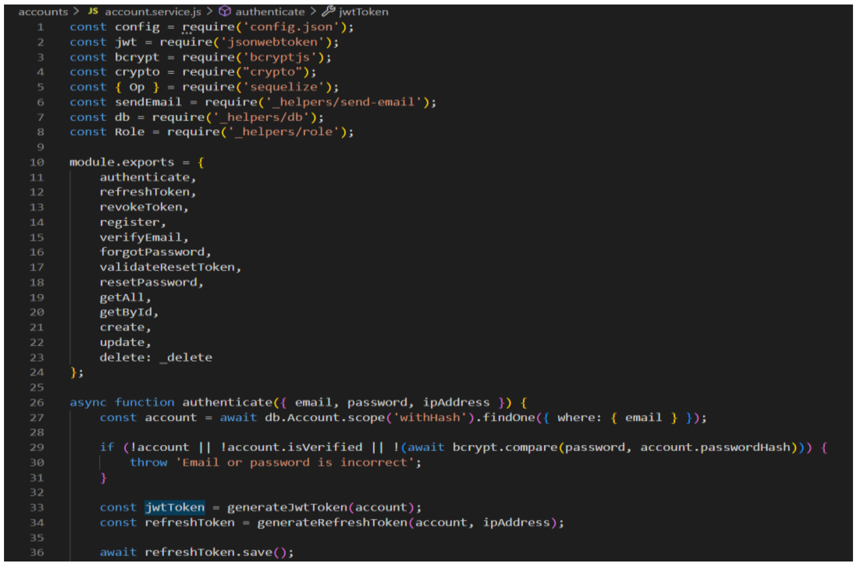


```
const { DataTypes } = require('sequelize');

module.exports = model;


function model(sequelize) {
    const attributes = {
        token: {type: DataTypes.STRING},
        expires: {type: DataTypes.DATE},
        created: {type: DataTypes.DATE, allowNull: false, defaultValue: DataTypes.NOW},
        createdByIp: {type: DataTypes.STRING},
        revoked: {type: DataTypes.DATE},
        revokedByIp: {type: DataTypes.STRING},
        replacedByToken: {type: DataTypes.STRING},
        isExpired: {
            type: DataTypes.VIRTUAL,
            get() { return Date.now() >= this.expires}
        },
        isActive: {
            type: DataTypes.VIRTUAL,
            get() { return !this.revoked && !this.isExpired;}

        }
    };

    const options = {
        timestamps: false
    };

    return sequelize.define('refreshToken', attributes, options);
}
```

```
Uses Sequelize to define
the schema for the
refreshTokens table in
the MySQL
database. The exported
Sequelize model object
gives full access to
perform CRUD
operations on refresh
tokens in MySQL
```

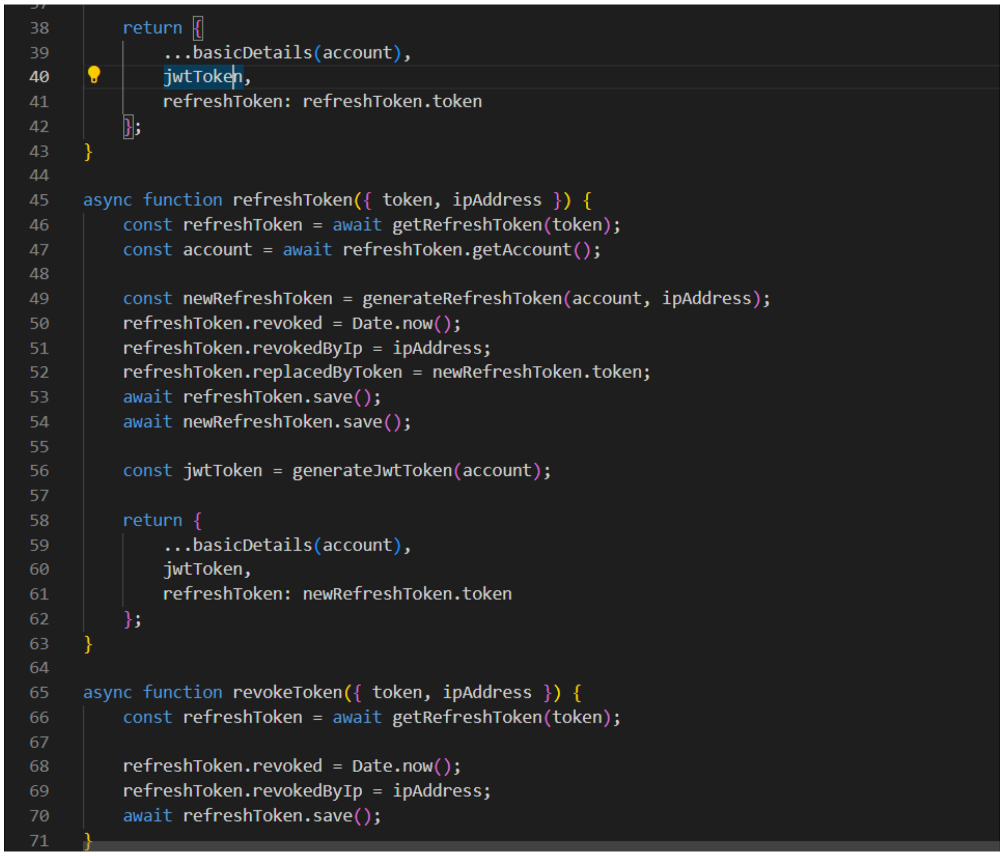
Account Service (*Path:/accounts/account.service.js*)

The service encapsulates all interaction with the Sequelize account models and exposes a simple set of methods which are used by the accounts controller.

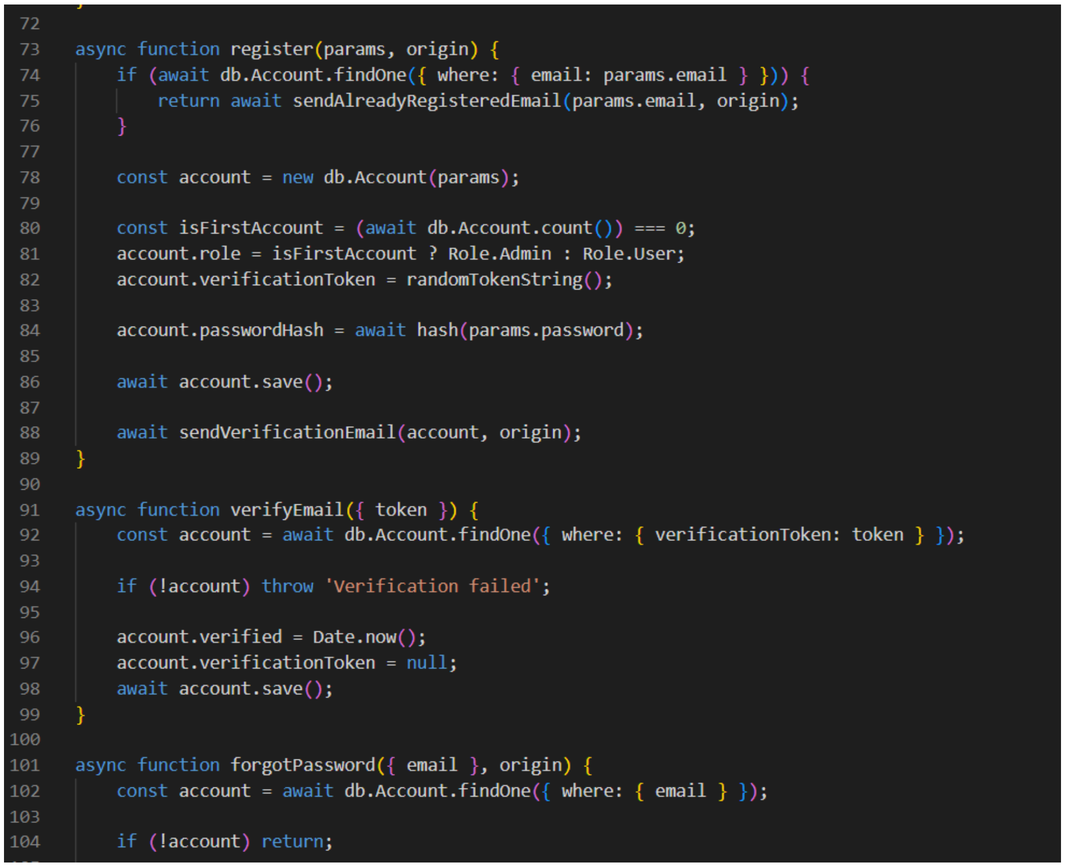


accounts > JS account.service.js > authenticate > jwtToken

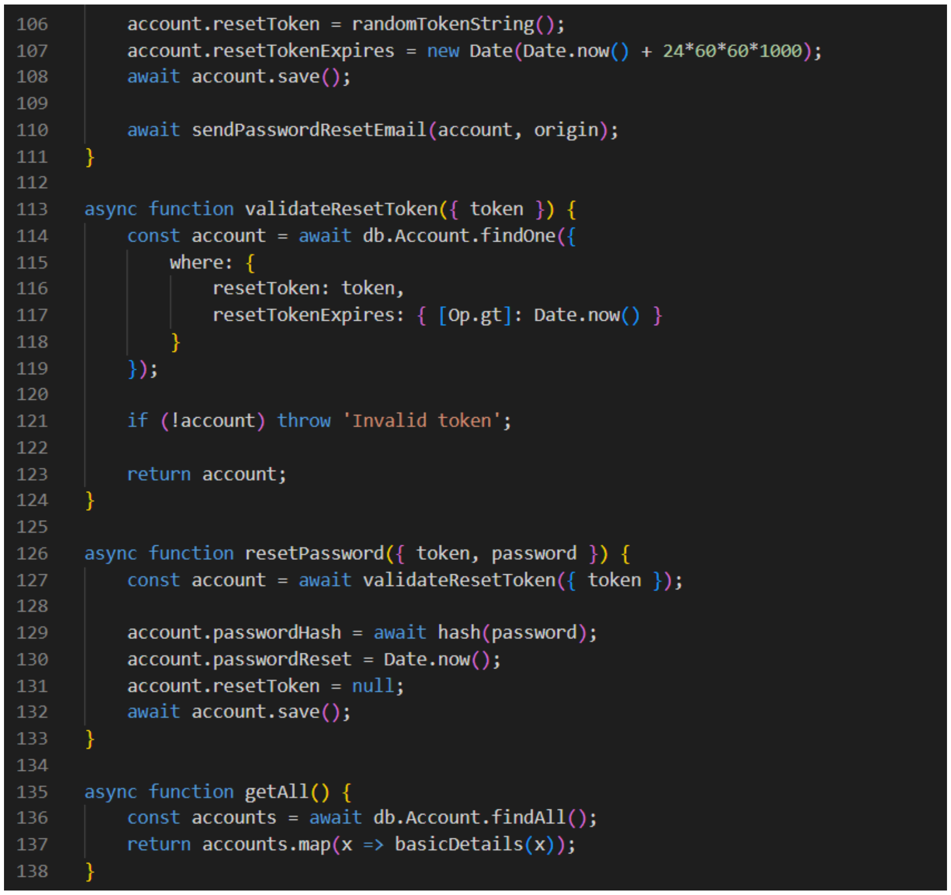
```
const config = require('config.json');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
const crypto = require("crypto");
const { Op } = require('sequelize');
const sendEmail = require('_helpers/send-email');
const db = require('_helpers/db');
const Role = require('_helpers/role');

module.exports = {
    authenticate,
    refreshToken,
    revokeToken,
    register,
    verifyEmail,
    forgotPassword,
    validateResetToken,
    resetPassword,
    getAll,
    getById,
    create,
    update,
    delete: _delete
};

async function authenticate({ email, password, ipAddress }) {
    const account = await db.Account.scope('withHash').findOne({ where: { email } });

    if (!account || !account.isVerified || !(await bcrypt.compare(password, account.passwordHash))) {
        throw 'Email or password is incorrect';
    }

    const jwtToken = generateJwtToken(account);
    const refreshToken = generateRefreshToken(account, ipAddress);

    await refreshToken.save();
```



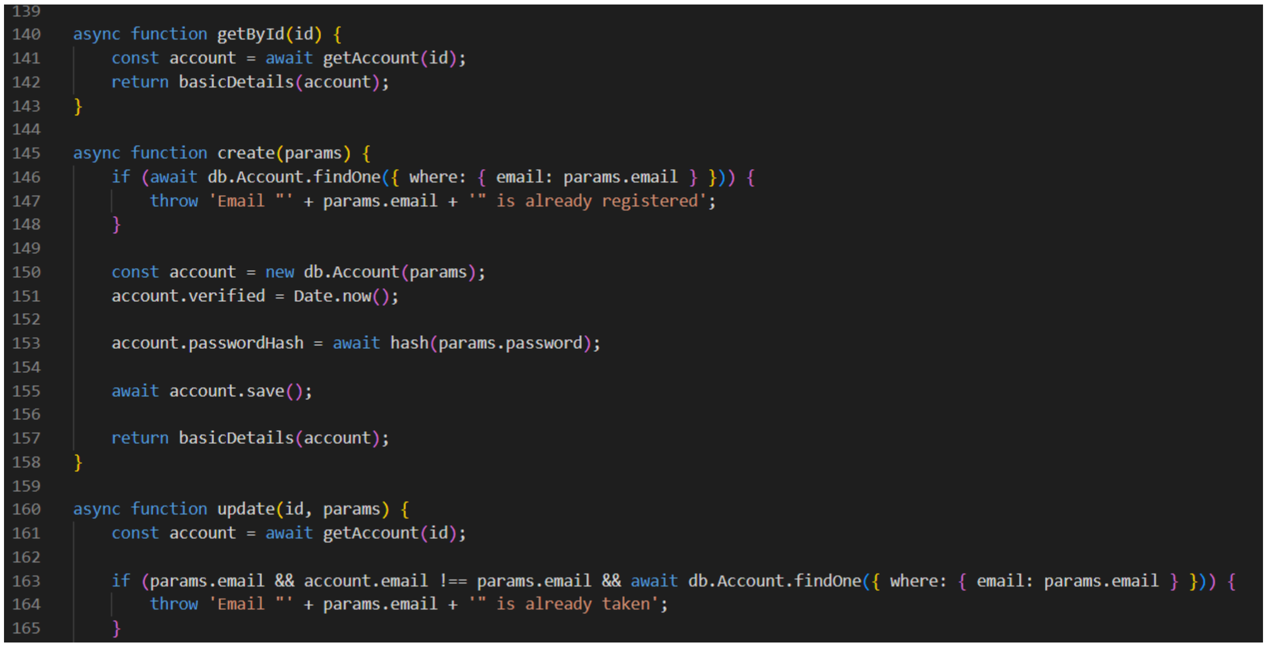
```
    return {
        ...basicDetails(account),
        jwtToken,
        refreshToken: refreshToken.token
    };
}

async function refreshToken({ token, ipAddress }) {
    const refreshToken = await getRefreshToken(token);
    const account = await refreshToken.getAccount();

    const newRefreshToken = generateRefreshToken(account, ipAddress);
    refreshToken.revoked = Date.now();
    refreshToken.revokedByIp = ipAddress;
    refreshToken.replacedByToken = newRefreshToken.token;
    await refreshToken.save();
    await newRefreshToken.save();

    const jwtToken = generateJwtToken(account);

    return {
        ...basicDetails(account),
        jwtToken,
        refreshToken: newRefreshToken.token
    };
}

async function revokeToken({ token, ipAddress }) {
    const refreshToken = await getRefreshToken(token);

    refreshToken.revoked = Date.now();
    refreshToken.revokedByIp = ipAddress;
    await refreshToken.save();
}
```



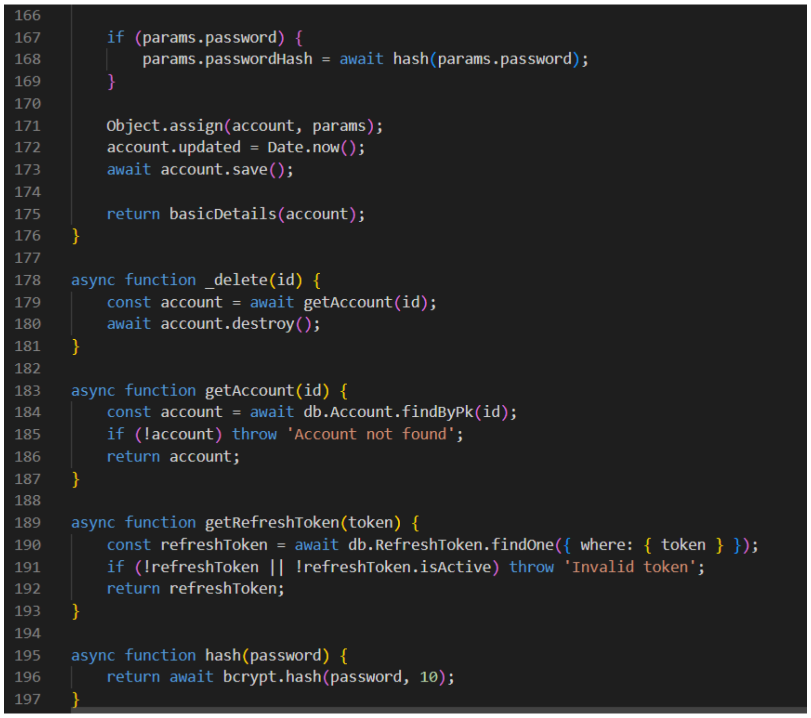
```

async function register(params, origin) {
    if (await db.Account.findOne({ where: { email: params.email } })) {
        return await sendAlreadyRegisteredEmail(params.email, origin);
    }

    const account = new db.Account(params);

    const isFirstAccount = (await db.Account.count()) === 0;
    account.role = isFirstAccount ? Role.Admin : Role.User;
    account.verificationToken = randomTokenString();

    account.passwordHash = await hash(params.password);

    await account.save();

    await sendVerificationEmail(account, origin);
}

async function verifyEmail({ token }) {
    const account = await db.Account.findOne({ where: { verificationToken: token } });

    if (!account) throw 'Verification failed';

    account.verified = Date.now();
    account.verificationToken = null;
    await account.save();
}

async function forgotPassword({ email }, origin) {
    const account = await db.Account.findOne({ where: { email } });

    if (!account) return;
```



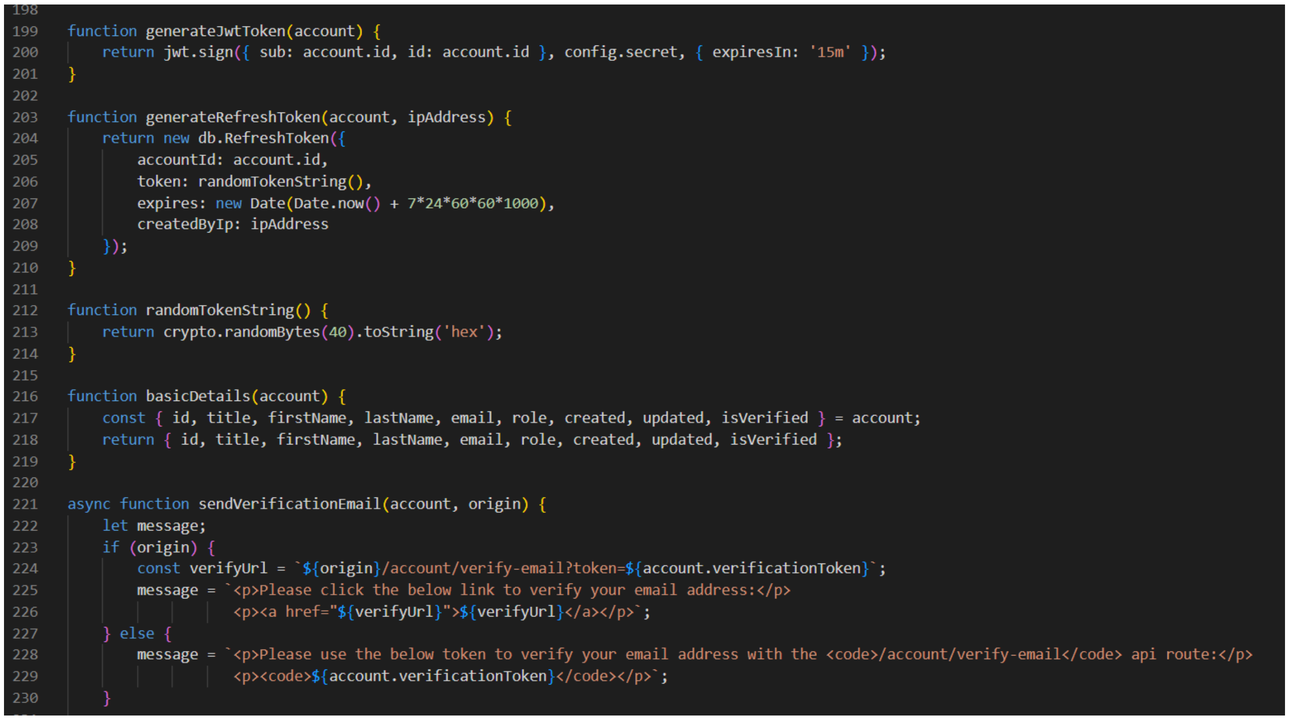
```
    account.resetToken = randomTokenString();
    account.resetTokenExpires = new Date(Date.now() + 24*60*60*1000);
    await account.save();

    await sendPasswordResetEmail(account, origin);
}

async function validateResetToken({ token }) {
    const account = await db.Account.findOne({
        where: {
            resetToken: token,
            resetTokenExpires: { [Op.gt]: Date.now() }
        }
    });

    if (!account) throw 'Invalid token';

    return account;
}

async function resetPassword({ token, password }) {
    const account = await validateResetToken({ token });

    account.passwordHash = await hash(password);
    account.passwordReset = Date.now();
    account.resetToken = null;
    await account.save();
}

async function getAll() {
    const accounts = await db.Account.findAll();
    return accounts.map(x => basicDetails(x));
}
```



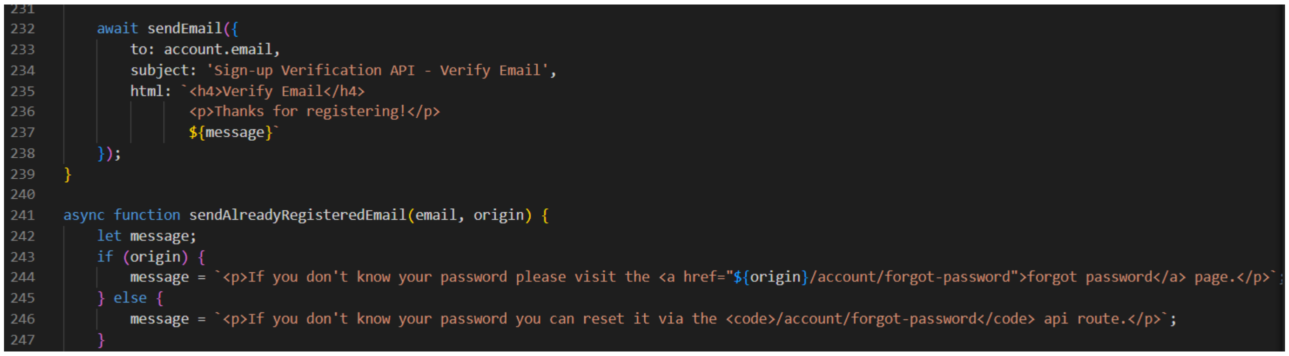
```

async function getById(id) {
    const account = await getAccount(id);
    return basicDetails(account);
}

async function create(params) {
    if (await db.Account.findOne({ where: { email: params.email } })) {
        throw 'Email "' + params.email + '" is already registered';
    }

    const account = new db.Account(params);
    account.verified = Date.now();

    account.passwordHash = await hash(params.password);

    await account.save();

    return basicDetails(account);
}

async function update(id, params) {
    const account = await getAccount(id);

    if (params.email && account.email !== params.email && await db.Account.findOne({ where: { email: params.email } })) {
        throw 'Email "' + params.email + '" is already taken';
    }
```



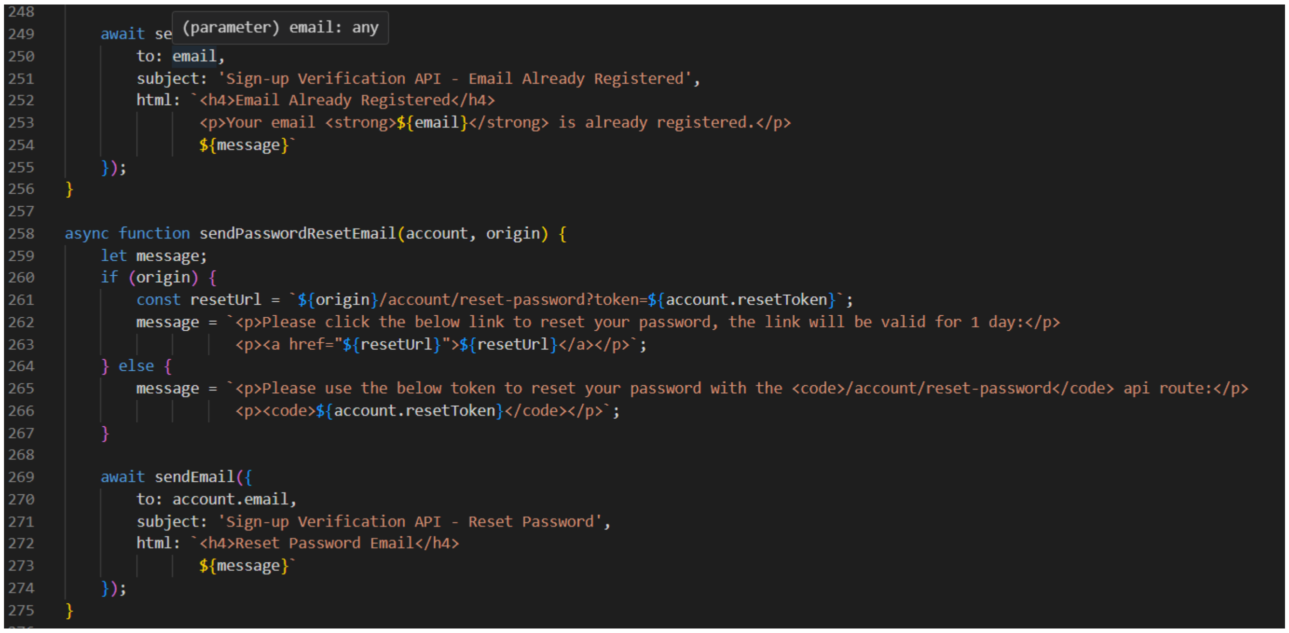
```
    if (params.password) {
        params.passwordHash = await hash(params.password);
    }

    Object.assign(account, params);
    account.updated = Date.now();
    await account.save();

    return basicDetails(account);
}

async function _delete(id) {
    const account = await getAccount(id);
    await account.destroy();
}

async function getAccount(id) {
    const account = await db.Account.findByPk(id);
    if (!account) throw 'Account not found';
    return account;
}

async function getRefreshToken(token) {
    const refreshToken = await db.RefreshToken.findOne({ where: { token } });
    if (!refreshToken || !refreshToken.isActive) throw 'Invalid token';
    return refreshToken;
}

async function hash(password) {
    return await bcrypt.hash(password, 10);
}
```



```

function generateJwtToken(account) {
    return jwt.sign({ sub: account.id, id: account.id }, config.secret, { expiresIn: '15m' });
}

function generateRefreshToken(account, ipAddress) {
    return new db.RefreshToken({
        accountId: account.id,
        token: randomTokenString(),
        expires: new Date(Date.now() + 7*24*60*60*1000),
        createdByIp: ipAddress
    });
}

function randomTokenString() {
    return crypto.randomBytes(40).toString('hex');
}

function basicDetails(account) {
    const { id, title, firstName, lastName, email, role, created, updated, isVerified } = account;
    return { id, title, firstName, lastName, email, role, created, updated, isVerified };
}

async function sendVerificationEmail(account, origin) {
    let message;
    if (origin) {
        const verifyUrl = `${origin}/account/verify-email?token=${account.verificationToken}`;
        message = `<p>Please click the below link to verify your email address:</p>
                   <p><a href="${verifyUrl}">${verifyUrl}</a></p>`;
    } else {
        message = `<p>Please use the below token to verify your email address with the <code>/account/verify-email</code> api route:</p>
                   <p><code>${account.verificationToken}</code></p>`;
    }
```



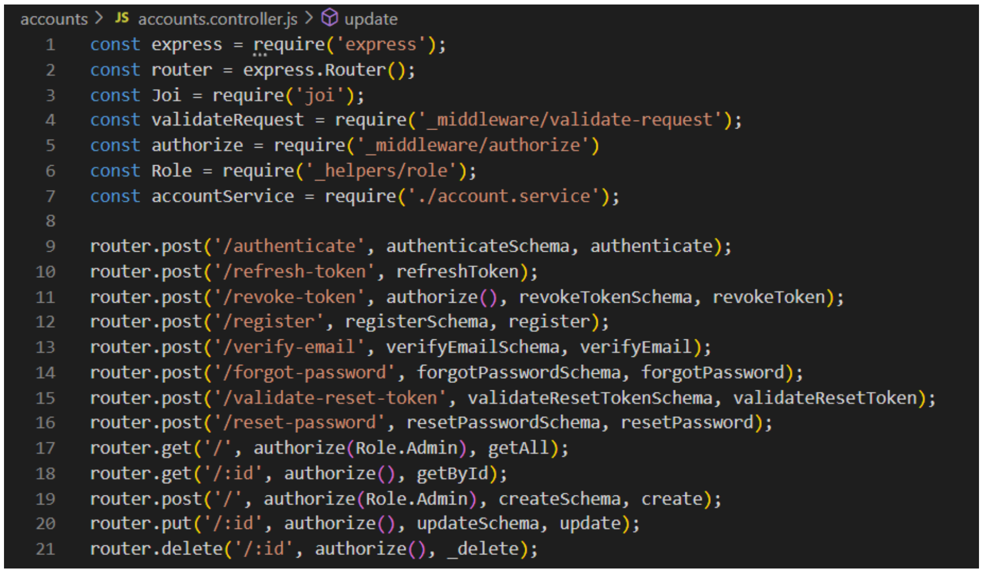
```

    await sendEmail({
        to: account.email,
        subject: 'Sign-up Verification API - Verify Email',
        html: `<h4>Verify Email</h4>
               <p>Thanks for registering!</p>
               ${message}`
    });
}

async function sendAlreadyRegisteredEmail(email, origin) {
    let message;
    if (origin) {
        message = `<p>If you don't know your password please visit the <a href="${origin}/account/forgot-password">forgot password</a> page.</p>`;
    } else {
        message = `<p>If you don't know your password you can reset it via the <code>/account/forgot-password</code> api route.</p>`;
    }
```



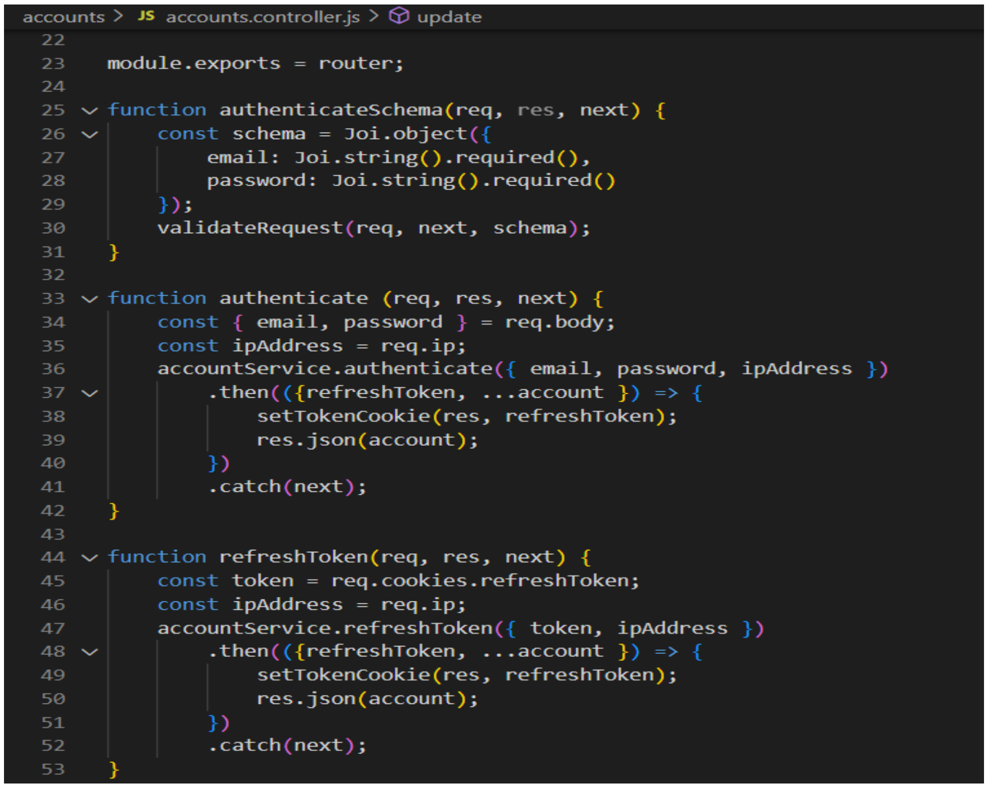
```
    await se (parameter) email: any
        to: email,
        subject: 'Sign-up Verification API - Email Already Registered',
        html: `<h4>Email Already Registered</h4>
               <p>Your email <strong>${email}</strong> is already registered.</p>
               ${message}`
    });
}

async function sendPasswordResetEmail(account, origin) {
    let message;
    if (origin) {
        const resetUrl = `${origin}/account/reset-password?token=${account.resetToken}`;
        message = `<p>Please click the below link to reset your password, the link will be valid for 1 day:</p>
                   <p><a href="${resetUrl}">${resetUrl}</a></p>`;
    } else {
        message = `<p>Please use the below token to reset your password with the <code>/account/reset-password</code> api route:</p>
                   <p><code>${account.resetToken}</code></p>`;
    }

    await sendEmail({
        to: account.email,
        subject: 'Sign-up Verification API - Reset Password',
        html: `<h4>Reset Password Email</h4>
               ${message}`
    });
}
```

Accounts Controller (*Path:/accounts/accounts.controller.js*)

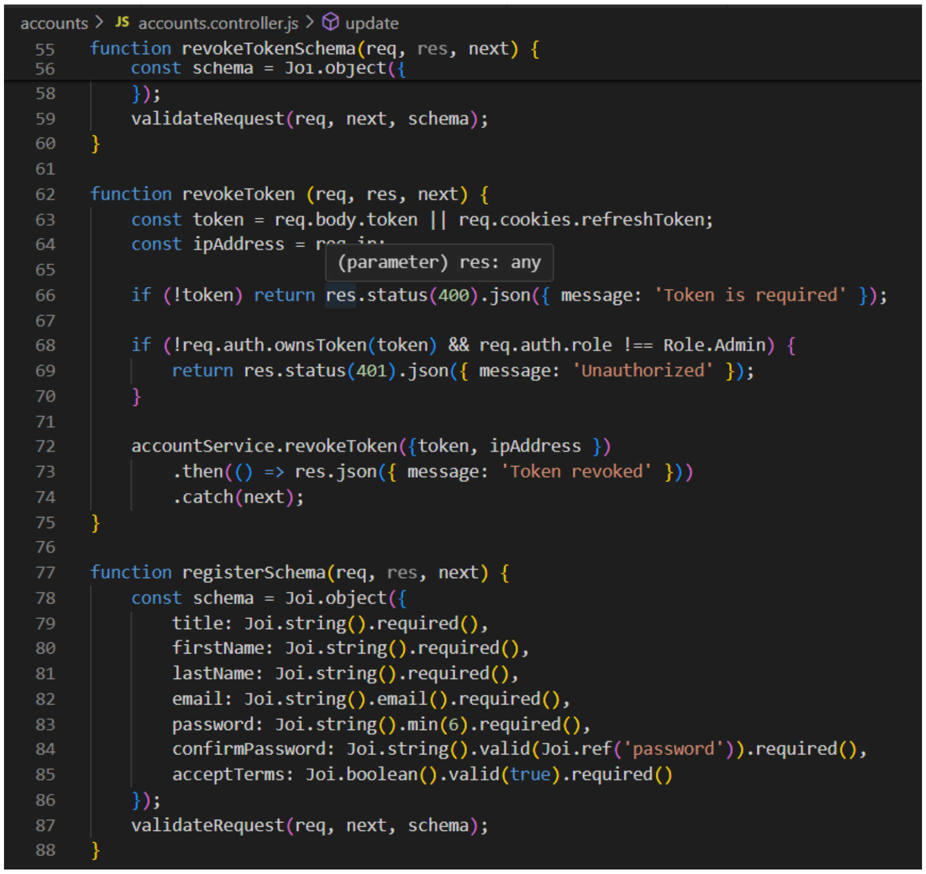
Defines all /accounts routes for the Node.js + MySQL boilerplate api, the route definitions are grouped together at the top of the file and the implementation functions are below, followed by local helper functions. The controller is bound to the /accounts path in the main server.js file.



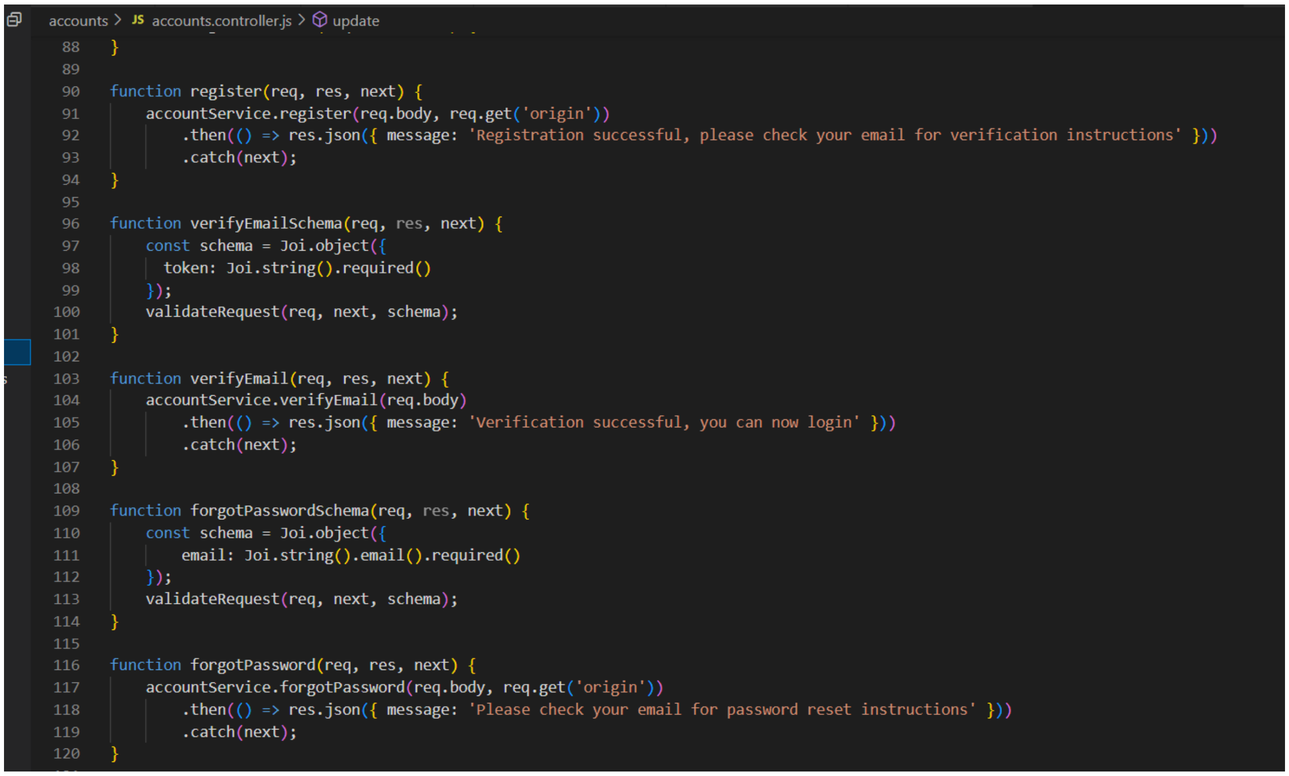
```
const express = require('express');
const router = express.Router();
const Joi = require('joi');
const validateRequest = require('_middleware/validate-request');
const authorize = require('_middleware/authorize')
const Role = require('_helpers/role');
const accountService = require('./account.service');

router.post('/authenticate', authenticateSchema, authenticate);
router.post('/refresh-token', refreshToken);
router.post('/revoke-token', authorize(), revokeTokenSchema, revokeToken);
router.post('/register', registerSchema, register);
router.post('/verify-email', verifyEmailSchema, verifyEmail);
router.post('/forgot-password', forgotPasswordSchema, forgotPassword);
router.post('/validate-reset-token', validateResetTokenSchema, validateResetToken);
router.post('/reset-password', resetPasswordSchema, resetPassword);
router.get('/', authorize(Role.Admin), getAll);
router.get('/:id', authorize(), getById);
router.post('/', authorize(Role.Admin), createSchema, create);
router.put('/:id', authorize(), updateSchema, update);
router.delete('/:id', authorize(), _delete);
```



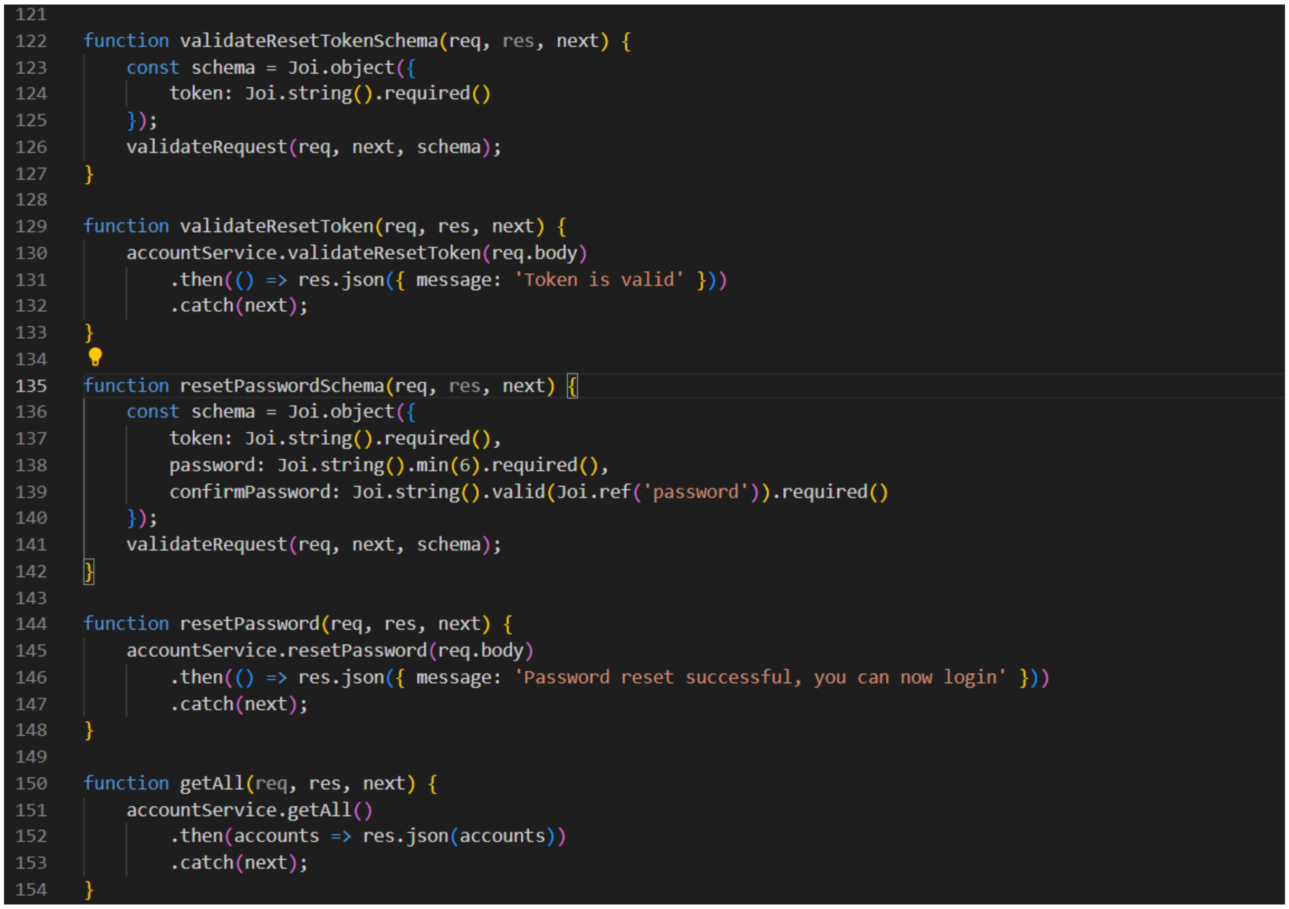
```
module.exports = router;

function authenticateSchema(req, res, next) {
    const schema = Joi.object({
        email: Joi.string().required(),
        password: Joi.string().required()
    });
    validateRequest(req, next, schema);
}

function authenticate (req, res, next) {
    const { email, password } = req.body;
    const ipAddress = req.ip;
    accountService.authenticate({ email, password, ipAddress })
        .then(({refreshToken, ...account }) => {
            setTokenCookie(res, refreshToken);
            res.json(account);
        })
        .catch(next);
}

function refreshToken(req, res, next) {
    const token = req.cookies.refreshToken;
    const ipAddress = req.ip;
    accountService.refreshToken({ token, ipAddress })
        .then(({refreshToken, ...account }) => {
            setTokenCookie(res, refreshToken);
            res.json(account);
        })
        .catch(next);
}
```



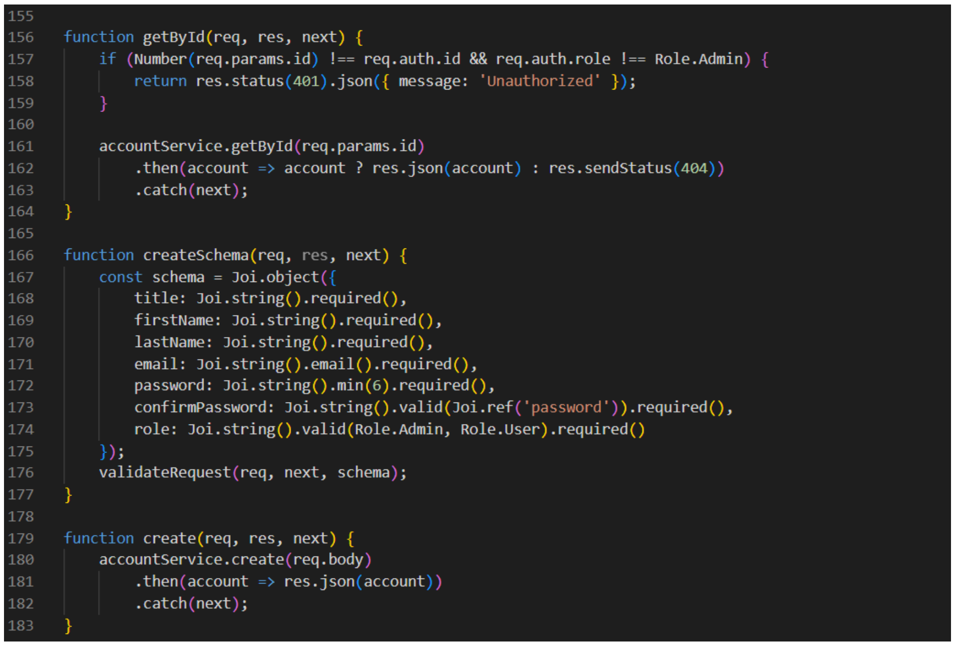
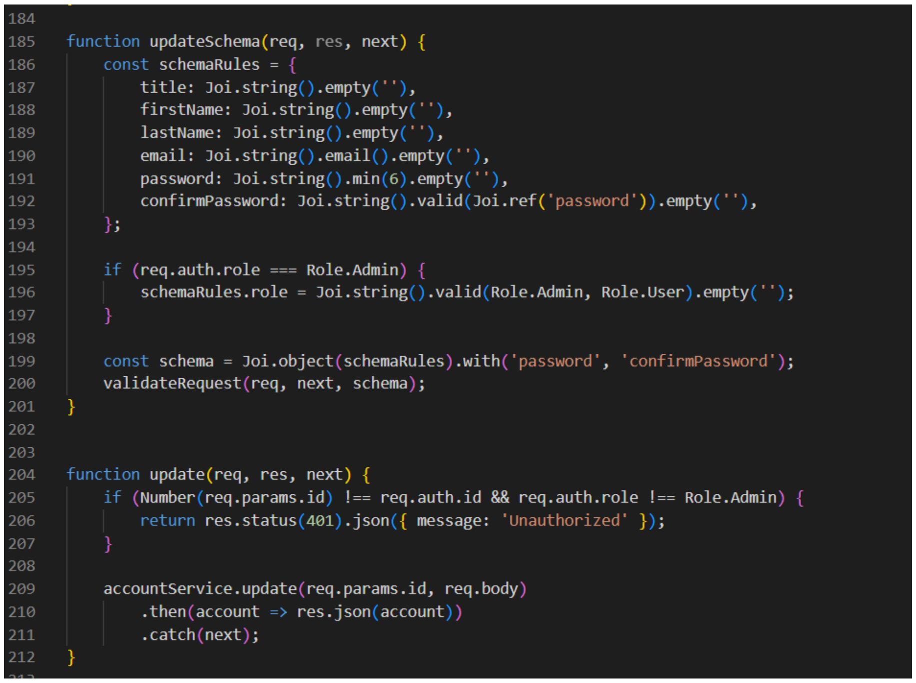
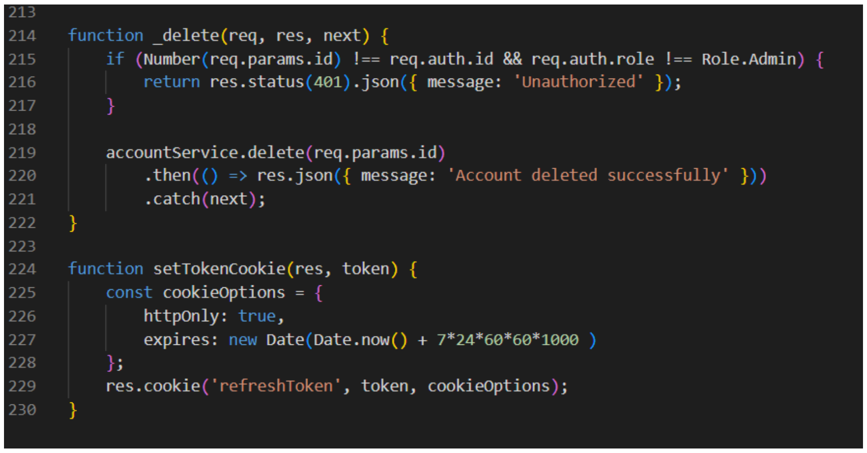
```
function revokeTokenSchema(req, res, next) {
    const schema = Joi.object({
    });
    validateRequest(req, next, schema);
}

function revokeToken (req, res, next) {
    const token = req.body.token || req.cookies.refreshToken;
    const ipAddress = req.ip;

    if (!token) return res.status(400).json({ message: 'Token is required' });

    if (!req.auth.ownsToken(token) && req.auth.role !== Role.Admin) {
        return res.status(401).json({ message: 'Unauthorized' });
    }

    accountService.revokeToken({token, ipAddress })
        .then(() => res.json({ message: 'Token revoked' }))
        .catch(next);
}

function registerSchema(req, res, next) {
    const schema = Joi.object({
        title: Joi.string().required(),
        firstName: Joi.string().required(),
        lastName: Joi.string().required(),
        email: Joi.string().email().required(),
        password: Joi.string().min(6).required(),
        confirmPassword: Joi.string().valid(Joi.ref('password')).required(),
        acceptTerms: Joi.boolean().valid(true).required()
    });
    validateRequest(req, next, schema);
}
```



```
}

function register(req, res, next) {
    accountService.register(req.body, req.get('origin'))
        .then(() => res.json({ message: 'Registration successful, please check your email for verification instructions' }))
        .catch(next);
}

function verifyEmailSchema(req, res, next) {
    const schema = Joi.object({
      token: Joi.string().required()
    });
    validateRequest(req, next, schema);
}

function verifyEmail(req, res, next) {
    accountService.verifyEmail(req.body)
        .then(() => res.json({ message: 'Verification successful, you can now login' }))
        .catch(next);
}

function forgotPasswordSchema(req, res, next) {
    const schema = Joi.object({
        email: Joi.string().email().required()
    });
    validateRequest(req, next, schema);
}

function forgotPassword(req, res, next) {
    accountService.forgotPassword(req.body, req.get('origin'))
        .then(() => res.json({ message: 'Please check your email for password reset instructions' }))
        .catch(next);
}
```

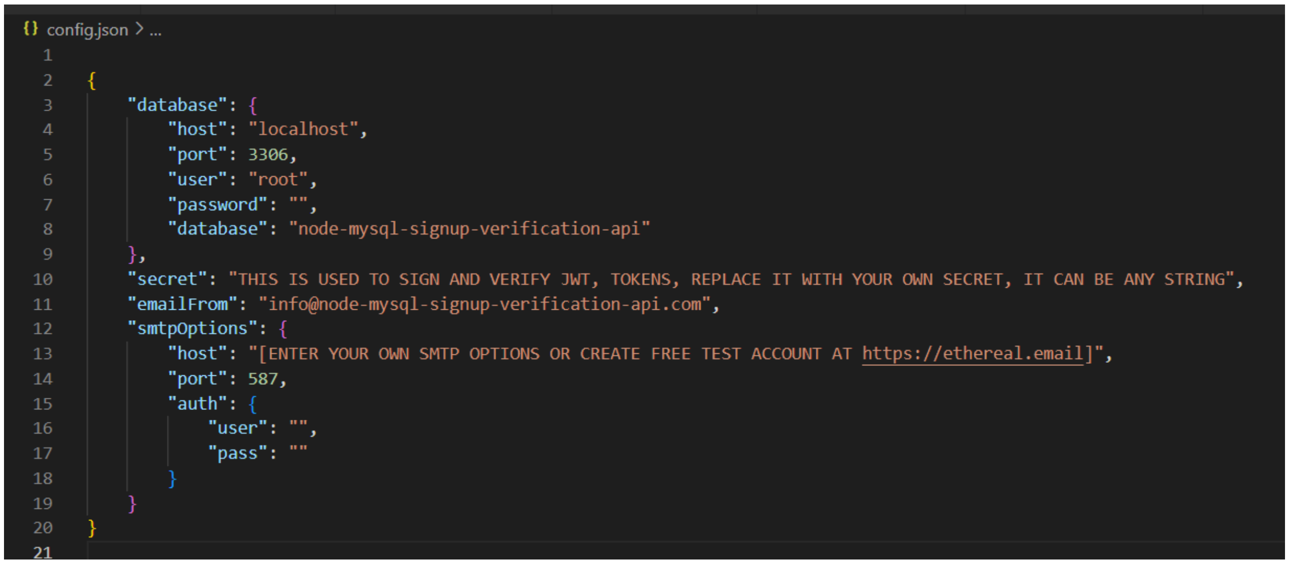


```

function validateResetTokenSchema(req, res, next) {
    const schema = Joi.object({
        token: Joi.string().required()
    });
    validateRequest(req, next, schema);
}

function validateResetToken(req, res, next) {
    accountService.validateResetToken(req.body)
        .then(() => res.json({ message: 'Token is valid' }))
        .catch(next);
}

function resetPasswordSchema(req, res, next) {
    const schema = Joi.object({
        token: Joi.string().required(),
        password: Joi.string().min(6).required(),
        confirmPassword: Joi.string().valid(Joi.ref('password')).required()
    });
    validateRequest(req, next, schema);
}

function resetPassword(req, res, next) {
    accountService.resetPassword(req.body)
        .then(() => res.json({ message: 'Password reset successful, you can now login' }))
        .catch(next);
}

function getAll(req, res, next) {
    accountService.getAll()
        .then(accounts => res.json(accounts))
        .catch(next);
}
```


```javascript

function getById(req, res, next) {
    if (Number(req.params.id) !== req.auth.id && req.auth.role !== Role.Admin) {
        return res.status(401).json({ message: 'Unauthorized' });
    }

    accountService.getById(req.params.id)
        .then(account => account ? res.json(account) : res.sendStatus(404))
        .catch(next);
}

function createSchema(req, res, next) {
    const schema = Joi.object({
        title: Joi.string().required(),
        firstName: Joi.string().required(),
        lastName: Joi.string().required(),
        email: Joi.string().email().required(),
        password: Joi.string().min(6).required(),
        confirmPassword: Joi.string().valid(Joi.ref('password')).required(),
        role: Joi.string().valid(Role.Admin, Role.User).required()
    });
    validateRequest(req, next, schema);
}

function create(req, res, next) {
    accountService.create(req.body)
        .then(account => res.json(account))
        .catch(next);
}
```

```javascript

function updateSchema(req, res, next) {
    const schemaRules = {
        title: Joi.string().empty(''),
        firstName: Joi.string().empty(''),
        lastName: Joi.string().empty(''),
        email: Joi.string().email().empty(''),
        password: Joi.string().min(6).empty(''),
        confirmPassword: Joi.string().valid(Joi.ref('password')).empty(''),
    };

    if (req.auth.role === Role.Admin) {
        schemaRules.role = Joi.string().valid(Role.Admin, Role.User).empty('');
    }

    const schema = Joi.object(schemaRules).with('password', 'confirmPassword');
    validateRequest(req, next, schema);
}


function update(req, res, next) {
    if (Number(req.params.id) !== req.auth.id && req.auth.role !== Role.Admin) {
        return res.status(401).json({ message: 'Unauthorized' });
    }

    accountService.update(req.params.id, req.body)
        .then(account => res.json(account))
        .catch(next);
}
```

```javascript

function _delete(req, res, next) {
    if (Number(req.params.id) !== req.auth.id && req.auth.role !== Role.Admin) {
        return res.status(401).json({ message: 'Unauthorized' });
    }

    accountService.delete(req.params.id)
        .then(() => res.json({ message: 'Account deleted successfully' }))
        .catch(next);
}

function setTokenCookie(res, token) {
    const cookieOptions = {
        httpOnly: true,
        expires: new Date(Date.now() + 7*24*60*60*1000 )
    };
    res.cookie('refreshToken', token, cookieOptions);
}
```

API Conffig (*Path:/config.json*)

Contains configuration data for the boilerplate api, it includes the database connection options for the MySQL database, the secret used for signing and verifying JWT tokens, the emailFrom address used to send emails, and the smtpOptions used to connect and authenticate with an email server.

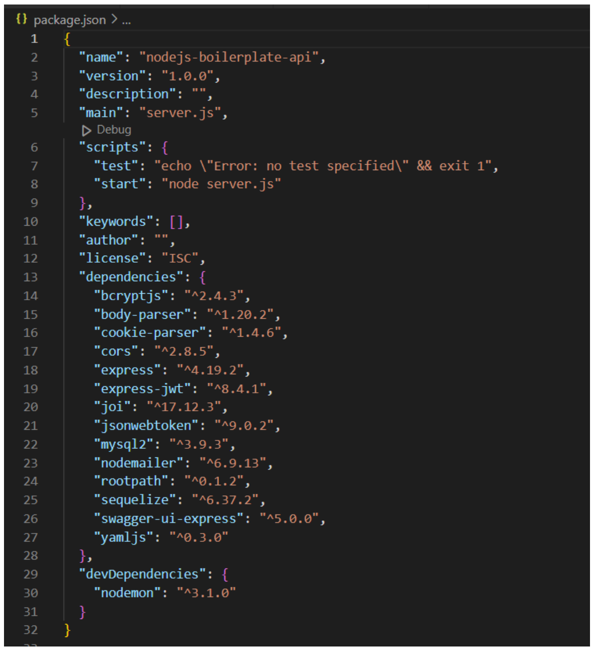


config.json > ...

```json

{
    "database": {
        "host": "localhost",
        "port": 3306,
        "user": "root",
        "password": "",
        "database": "node-mysql-signup-verification-api"
    },
    "secret": "THIS IS USED TO SIGN AND VERIFY JWT, TOKENS, REPLACE IT WITH YOUR OWN SECRET, IT CAN BE ANY STRING",
    "emailFrom": "info@node-mysql-signup-verification-api.com",
    "smtpOptions": {
        "host": "[ENTER YOUR OWN SMTP OPTIONS OR CREATE FREE TEST ACCOUNT AT https://ethereal.email]",
        "port": 587,
        "auth": {
            "user": "",
            "pass": ""
        }
    }
}
```

Package.json (*Path:/package.json*)

The package.json file contains project configuration information including package dependencies which get installed when you run npm install.

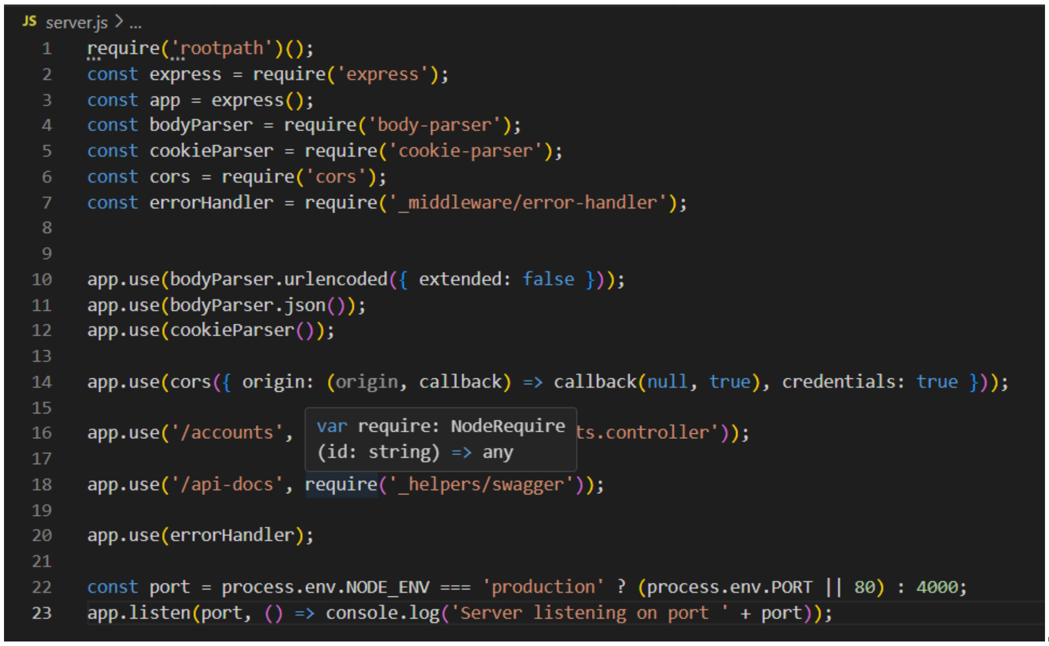


package.json > ...

```json
{
  "name": "nodejs-boilerplate-api",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.20.2",
    "cookie-parser": "^1.4.6",
    "cors": "^2.8.5",
    "express": "^4.19.2",
    "express-jwt": "^8.4.1",
    "joi": "^17.12.3",
    "jsonwebtoken": "^9.0.2",
    "mysql2": "^3.9.3",
    "nodemailer": "^6.9.13",
    "rootpath": "^0.1.2",
    "sequelize": "^6.37.2",
    "swagger-ui-express": "^5.0.0",
    "yamljs": "^0.3.0"
  },
  "devDependencies": {
    "nodemon": "^3.1.0"
  }
}
```

Server Startup File (*Path:/server.js*)

The server.js file is the entry point into the boilerplate Node.js api, it configures application middleware, binds controllers to routes and starts the Express web server for the api



server.js > ...

```
require('rootpath')();
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const cookieParser = require('cookie-parser');
const cors = require('cors');
const errorHandler = require('_middleware/error-handler');


app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.use(cookieParser());

app.use(cors({ origin: (origin, callback) => callback(null, true), credentials: true }));

app.use('/accounts', var require: NodeRequire ts.controller'));
                     (id: string) => any
app.use('/api-docs', require('_helpers/swagger'));

app.use(errorHandler);

const port = process.env.NODE_ENV === 'production' ? (process.env.PORT || 80) : 4000;
app.listen(port, () => console.log('Server listening on port ' + port));
```

.

Swagger API Documentation (*Path:/swagger.yaml*)

The YAML documentation is used by the swagger.js helper to automatically generate and serve interactive Swagger UI documentation on the /api-docs route of the boilerplate api. To preview the Swagger UI documentation without running the api simply copy and paste the below YAML into the swagger editor at Swagger Editor.

```
openapi: 3.0.0
info:
  title: Node.js Sign-up and Verification API
  description: Node.js and MySQL - API with email sign-up, verification,
authentication and forgot password
  version: 1.0.0

servers:
  - url: http://localhost:4000
    description: Local development server

paths:
  /accounts/authenticate:
    post:
      summary: Authenticate account credentials and return a JWT token and
a cookie with a refresh token
      description: Accounts must be verified before authenticating.
      operationId: authenticate
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                email:
                  type: string
                  example: "jason@example.com"
                password:
                  type: string
                  example: "pass123"
              required:
                - email
                - password
      responses:
        "200":
          description: Account details, a JWT access token and a refresh
token cookie
          headers:
            Set-Cookie:
              description: "`refreshToken`"
              schema:
                type: string
                example:
refreshToken=51872eca5efedcf424db4cf5afd16a9d00ad25b743a034c9c221afc85d18d
cd5e4ad6e3f08607550; Path=/; Expires=Tue, 16 Jun 2020 09:14:17 GMT;
HttpOnly
          content:
            application/json:
              schema:
                type: object
                properties:
                  id:
                    type: string
                    example: "5eb12e197e06a76ccdefc121"
                  title:
                    type: string
                    example: "Mr"
                  firstName:
                    type: string
                    example: "Jason"
                  lastName:
                    type: string
                    example: "Watmore"
                  email:
                    type: string
                    example: "jason@example.com"
                  role:
                    type: string
                    example: "Admin"
                  created:
                    type: string
                    example: "2020-05-05T09:12:57.848Z"
                  isVerified:
                    type: boolean
                    example: true
                  jwtToken:
                    type: string
                    example:
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI1ZWIxMmUxOTdlMDZhNzZjY2Rl
ZmMxMjEiLCJpZCI6IjVlYjEyZTE5N2UwNmE3NmNjZGVmYzEyMSIsImlhdCI6MTU4ODc1ODE1N3
0.xR9H0STbFOpSkuGA9jHNZOJ6eS7umHHqKRhI807YT1Y"
        "400":
          description: The email or password is incorrect
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: "Email or password is incorrect"
  /accounts/refresh-token:
    post:
      summary: Use a refresh token to generate a new JWT token and a new
refresh token
      description: The refresh token is sent and returned via cookies.
      operationId: refreshToken
      parameters:
        - in: cookie
          name: refreshToken
          description: The `refreshToken` cookie
          schema:
            type: string
            example:
51872eca5efedcf424db4cf5afd16a9d00ad25b743a034c9c221afc85d18dcd5e4ad6e3f08
607550
      responses:
        "200":
          description: Account details, a JWT access token and a new
refresh token cookie
          headers:
            Set-Cookie:
              description: "`refreshToken`"
              schema:
                type: string
                example:
refreshToken=51872eca5efedcf424db4cf5afd16a9d00ad25b743a034c9c221afc85d18d
cd5e4ad6e3f08607550; Path=/; Expires=Tue, 16 Jun 2020 09:14:17 GMT;
HttpOnly
          content:
            application/json:
              schema:
                type: object
                properties:
                  id:
                    type: string
                    example: "5eb12e197e06a76ccdefc121"
                  title:
                    type: string
                    example: "Mr"
                  firstName:
                    type: string
                    example: "Jason"
                  lastName:
                    type: string
                    example: "Watmore"
                  email:
                    type: string
                    example: "jason@example.com"
                  role:
                    type: string
                    example: "Admin"
                  created:
                    type: string
                    example: "2020-05-05T09:12:57.848Z"
                  isVerified:
                    type: boolean
                    example: true
                  jwtToken:
                    type: string
                    example:
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI1ZWIxMmUxOTdlMDZhNzZjY2Rl
ZmMxMjEiLCJpZCI6IjVlYjEyZTE5N2UwNmE3NmNjZGVmYzEyMSIsImlhdCI6MTU4ODc1ODE1N3
0.xR9H0STbFOpSkuGA9jHNZOJ6eS7umHHqKRhI807YT1Y"
        "400":
          description: The refresh token is invalid, revoked or expired
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: "Invalid token"
  /accounts/revoke-token:
    post:
      summary: Revoke a refresh token
      description: Admin users can revoke the tokens of any account,
regular users can only revoke their own tokens.
      operationId: revokeToken
      security:
        - bearerAuth: []
      parameters:
        - in: cookie
          name: refreshToken
          description: The refresh token can be sent in a cookie or the
post body, if both are sent the token in the body is used.
          schema:
            type: string
            example:
51872eca5efedcf424db4cf5afd16a9d00ad25b743a034c9c221afc85d18dcd5e4ad6e3f08
607550
      requestBody:
        content:
          application/json:
            schema:
              type: object
              properties:
                token:
                  type: string
                  example:
"51872eca5efedcf424db4cf5afd16a9d00ad25b743a034c9c221afc85d18dcd5e4ad6e3f0
8607550"
      responses:
        "200":
          description: The refresh token was successfully revoked
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: "Token revoked"
        "400":
          description: The refresh token is invalid
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: "Invalid token"
        "401":
          $ref: "#/components/responses/UnauthorizedError"
  /accounts/register:
    post:
      summary: Register a new user account and send a verification email
      description: The first account registered in the system is assigned
the `Admin` role, other accounts are assigned the `User` role.
      operationId: register
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                title:
                  type: string
                  example: "Mr"
                firstName:
                  type: string
                  example: "Jason"
                lastName:
                  type: string
                  example: "Watmore"
                email:
                  type: string
                  example: "jason@example.com"
                password:
                  type: string
                  example: "pass123"
                confirmPassword:
                  type: string
                  example: "pass123"
                acceptTerms:
                  type: boolean
              required:
                - title
                - firstName
                - lastName
                - email
                - password
                - confirmPassword
                - acceptTerms
      responses:
        "200":
          description: The registration request was successful and a
verification email has been sent to the specified email address
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: "Registration successful, please check your
email for verification instructions"
  /accounts/verify-email:
    post:
      summary: Verify a new account with a verification token received by
email after registration
      operationId: verifyEmail
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                token:
                  type: string
                  example:
"3c7f8d9c4cb348ff95a0b74a1452aa24fc9611bb76768bb9eafeeb826ddae2935f1880bc7
713318f"
              required:
                - token
      responses:
        "200":
          description: Verification was successful so you can now login to
the account
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: "Verification successful, you can now login"
        "400":
          description: Verification failed due to an invalid token
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: "Verification failed"
  /accounts/forgot-password:
    post:
      summary: Submit email address to reset the password on an account
      operationId: forgotPassword
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                email:
                  type: string
                  example: "jason@example.com"
              required:
                - email
      responses:
        "200":
          description: The request was received and an email has been sent
to the specified address with password reset instructions (if the email
address is associated with an account)
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: "Please check your email for password reset
instructions"

  /accounts/validate-reset-token:
    post:
      summary: Validate the reset password token received by email after
submitting to the /accounts/forgot-password route
      operationId: validateResetToken
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                token:
                  type: string
                  example:
"3c7f8d9c4cb348ff95a0b74a1452aa24fc9611bb76768bb9eafeeb826ddae2935f1880bc7
713318f"
              required:
                - token
      responses:
        "200":
          description: Token is valid
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: "Token is valid"
        "400":
          description: Token is invalid
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: "Invalid token"
  /accounts/reset-password:
    post:
      summary: Reset the password for an account
      operationId: resetPassword
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                token:
                  type: string
                  example:
"3c7f8d9c4cb348ff95a0b74a1452aa24fc9611bb76768bb9eafeeb826ddae2935f1880bc7
713318f"
                password:
                  type: string
                  example: "newPass123"
                confirmPassword:
                  type: string
                  example: "newPass123"
              required:
                - token
                - password
                - confirmPassword
      responses:
        "200":
          description: Password reset was successful so you can now login
to the account with the new password
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: "Password reset successful, you can now
login"
        "400":
          description: Password reset failed due to an invalid token
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: "Invalid token"
  /accounts:
    get:
```

```yaml
    summary: Get a list of all accounts
    description: Restricted to admin users.
    operationId: getAllAccounts
    security:
      - bearerAuth: []
    responses:
      "200":
        description: An array of all accounts
        content:
          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  id:
                    type: string
                    example: "5eb12e197e06a76ccdefc121"
                  title:
                    type: string
                    example: "Mr"
                  firstName:
                    type: string
                    example: "Jason"
                  lastName:
                    type: string
                    example: "Watmore"
                  email:
                    type: string
                    example: "jason@example.com"
                  role:
                    type: string
                    example: "Admin"
                  created:
                    type: string
                    example: "2020-05-05T09:12:57.848Z"
                  updated:
                    type: string
                    example: "2020-05-08T03:11:21.553Z"
      "401":
        $ref: "#/components/responses/UnauthorizedError"
  post:
    summary: Create a new account
    description: Restricted to admin users.
    operationId: createAccount
    security:
      - bearerAuth: []
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                title:
                  type: string
                  example: "Mr"
                firstName:
                  type: string
                  example: "Jason"
                lastName:
                  type: string
                  example: "Watmore"
                email:
                  type: string
                  example: "jason@example.com"
                password:
                  type: string
                  example: "pass123"
                confirmPassword:
                  type: string
                  example: "pass123"
                role:
                  type: string
                  enum: [Admin, User]
              required:
                - title
                - firstName
                - lastName
                - email
                - password
                - confirmPassword
                - role
      responses:
        "200":
          description: Account created successfully, verification is not
required for accounts created with this endpoint. The details of the new
account are returned.
          content:
            application/json:
              schema:
                type: object
                properties:
                  id:
                    type: string
                    example: "5eb12e197e06a76ccdefc121"
                  title:
                    type: string
                    example: "Mr"
                  firstName:
                    type: string
                    example: "Jason"
                  lastName:
                    type: string
                    example: "Watmore"
                  email:
                    type: string
                    example: "jason@example.com"
                  role:
                    type: string
                    example: "Admin"
                  created:
                    type: string
                    example: "2020-05-05T09:12:57.848Z"
        "400":

          description: Email is already registered
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: "Email 'jason@example.com' is already
registered"
        "401":
          $ref: "#/components/responses/UnauthorizedError"
  /accounts/{id}:
    parameters:
      - in: path
        name: id
        description: Account id
        required: true
        example: "5eb12e197e06a76ccdefc121"
        schema:
          type: string
    get:
      summary: Get a single account by id
      description: Admin users can access any account, regular users are
restricted to their own account.
      operationId: getAccountById
      security:
        - bearerAuth: []
      responses:
        "200":
          description: Details of the specified account
          content:
            application/json:
              schema:
                type: object
                properties:
                  id:
                    type: string
                    example: "5eb12e197e06a76ccdefc121"
                  title:
                    type: string
                    example: "Mr"
                  firstName:
                    type: string
                    example: "Jason"
                  lastName:
                    type: string
                    example: "Watmore"
                  email:
                    type: string
                    example: "jason@example.com"
                  role:
                    type: string
                    example: "Admin"
                  created:
                    type: string
                    example: "2020-05-05T09:12:57.848Z"
                  updated:
                    type: string
                    example: "2020-05-08T03:11:21.553Z"
        "404":
          $ref: "#/components/responses/NotFoundError"
        "401":
          $ref: "#/components/responses/UnauthorizedError"
    put:
      summary: Update an account
      description: Admin users can update any account including role,
regular users are restricted to their own account and cannot update role.
      operationId: updateAccount
      security:
        - bearerAuth: []
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                title:
                  type: string
                  example: "Mr"
                firstName:
                  type: string
                  example: "Jason"
                lastName:
                  type: string
                  example: "Watmore"
                email:
                  type: string
                  example: "jason@example.com"
                password:
                  type: string
                  example: "pass123"
                confirmPassword:
                  type: string
                  example: "pass123"
                role:
                  type: string
                  enum: [Admin, User]
      responses:
        "200":
          description: Account updated successfully. The details of the
updated account are returned.
          content:
            application/json:
              schema:
                type: object
                properties:
                  id:
                    type: string
                    example: "5eb12e197e06a76ccdefc121"
                  title:
                    type: string
                    example: "Mr"
                  firstName:
                    type: string
                    example: "Jason"
                  lastName:
                    type: string
                    example: "Watmore"
                  email:
                    type: string
                    example: "jason@example.com"
                  role:
                    type: string
                    example: "Admin"
                  created:
                    type: string
                    example: "2020-05-05T09:12:57.848Z"
                  updated:
                    type: string
                    example: "2020-05-08T03:11:21.553Z"
        "404":
          $ref: "#/components/responses/NotFoundError"
        "401":
          $ref: "#/components/responses/UnauthorizedError"
    delete:
      summary: Delete an account
      description: Admin users can delete any account, regular users are
restricted to their own account.
      operationId: deleteAccount
      security:
        - bearerAuth: []
      responses:
        "200":
          description: Account deleted successfully
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: "Account deleted successfully"
        "404":
          $ref: "#/components/responses/NotFoundError"
        "401":
          $ref: "#/components/responses/UnauthorizedError"

components:
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
  responses:
    UnauthorizedError:
      description: Access token is missing or invalid, or the user does
not have access to perform the action
      content:
        application/json:
          schema:
            type: object
            properties:
              message:
                type: string
                example: "Unauthorized"
    NotFoundError:
      description: Not Found
      content:
        application/json:
          schema:
            type: object
            properties:
              message:
                type: string
                example: "Not Found"
```

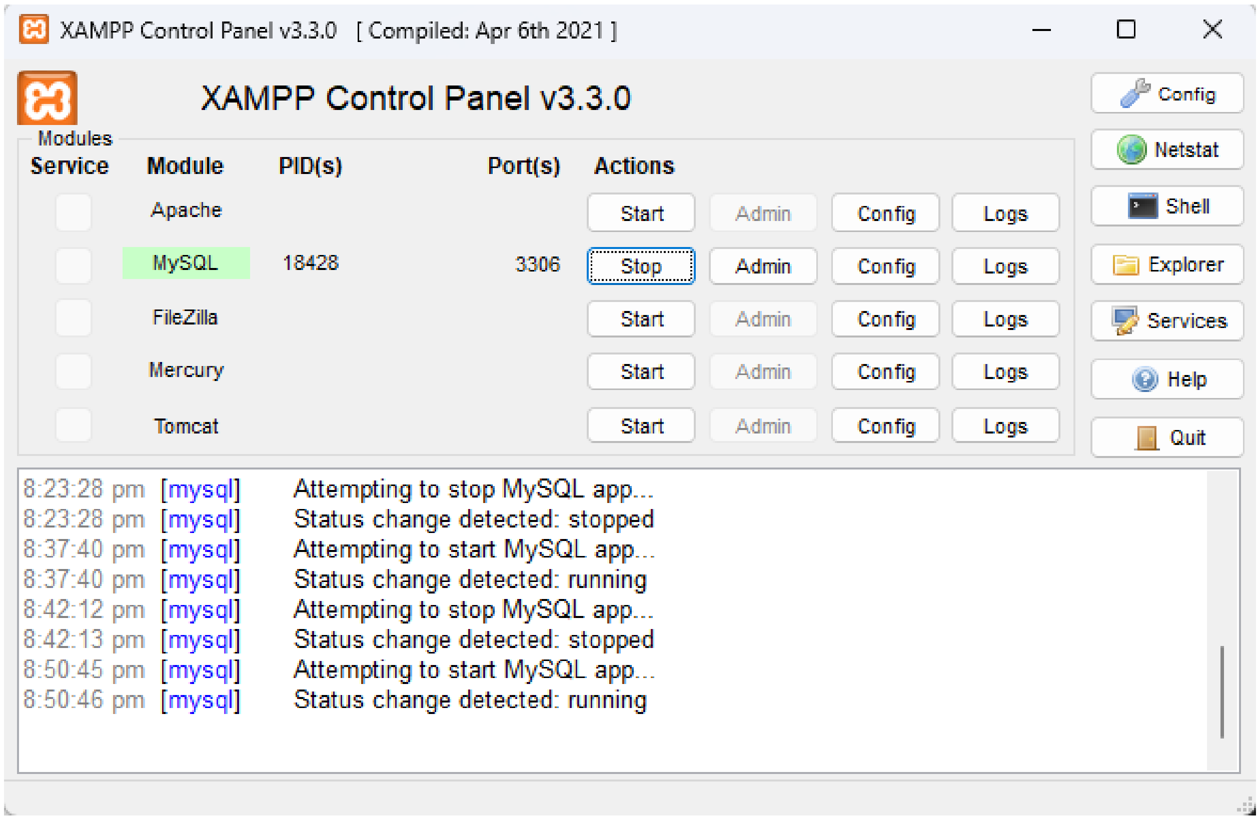
Preparations

Before the test, Ensure that you have followed the following guidelines:

- Installed NodeJS from their official website Download Node.js
- Installed MySQL from their website Download MySQL Community Server. Or use the XAMPP installer Apache Friends to run MySQL(MariaDB) on the XAMPP Control Panel.
- Prepared the API project source code.
- Installed the required npm packages by running the command npm i or npm install in the command-line within the root folder of the project.
- Configured the SMTP setting for email within the smtpOptions property in the /src/config.json file. We use Ethereal Email for testing.
- Updated the secret property in the config.json file as it is used for signing and verifying JWT tokens for authentication. We use GUID Generator to join a couple of GUIDs together and make a long random string.
- And finally start the API by running npm start (or npm run start:dev to start with nodemon) from the command line in the project root folder, you should see the message Server listening on port 4000, and you can view the Swagger API documentation at http://localhost:4000/api-docs.

npTesting the API Locally using NodeJS

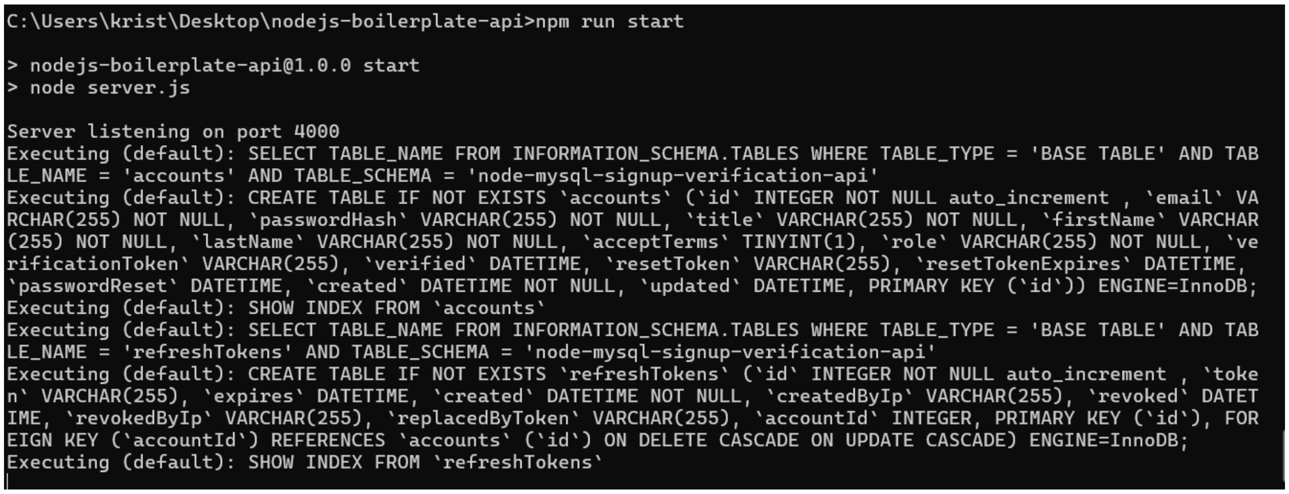
First thing to do is run an instance of your MySQL Server


XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]
XAMPP Control Panel v3.3.0
Modules
Config
Netstat
Shell
Explorer
Services
Help
Quit

Service	Module	PID(s)	Port(s)	Actions			
	Apache			Start	Admin	Config	Logs
	MySQL	18428	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

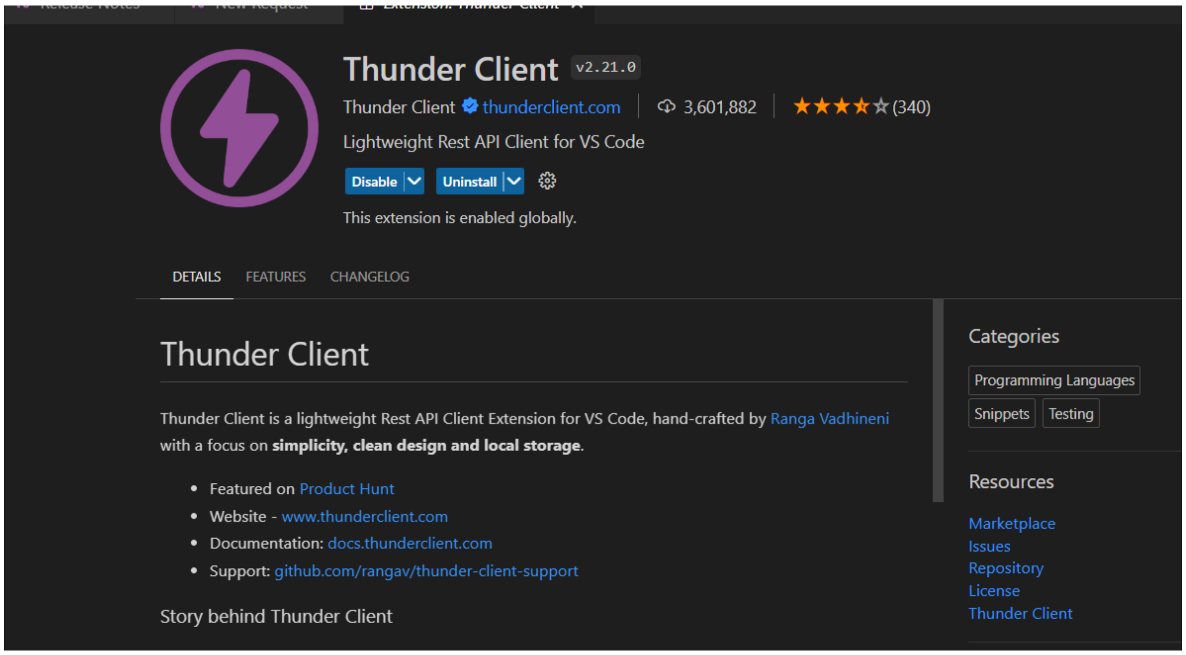
```
8:23:28 pm  [mysql]     Attempting to stop MySQL app...
8:23:28 pm  [mysql]     Status change detected: stopped
8:37:40 pm  [mysql]     Attempting to start MySQL app...
8:37:40 pm  [mysql]     Status change detected: running
8:42:12 pm  [mysql]     Attempting to stop MySQL app...
8:42:13 pm  [mysql]     Status change detected: stopped
8:50:45 pm  [mysql]     Attempting to start MySQL app...
8:50:46 pm  [mysql]     Status change detected: running
```

Then type 'npm start' or 'npm run start:dev' in the terminal within your project folder's root directory to start the server. It should show the Sequelize taking action.



```
C:\Users\krist\Desktop\nodejs-boilerplate-api>npm run start

> nodejs-boilerplate-api@1.0.0 start
> node server.js

Server listening on port 4000
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TAB
LE_NAME = 'accounts' AND TABLE_SCHEMA = 'node-mysql-signup-verification-api'
Executing (default): CREATE TABLE IF NOT EXISTS `accounts` (`id` INTEGER NOT NULL auto_increment , `email` VA
RCHAR(255) NOT NULL, `passwordHash` VARCHAR(255) NOT NULL, `title` VARCHAR(255) NOT NULL, `firstName` VARCHAR
(255) NOT NULL, `lastName` VARCHAR(255) NOT NULL, `acceptTerms` TINYINT(1), `role` VARCHAR(255) NOT NULL, `ve
rificationToken` VARCHAR(255), `verified` DATETIME, `resetToken` VARCHAR(255), `resetTokenExpires` DATETIME,
`passwordReset` DATETIME, `created` DATETIME NOT NULL, `updated` DATETIME, PRIMARY KEY (`id`)) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `accounts`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TAB
LE_NAME = 'refreshTokens' AND TABLE_SCHEMA = 'node-mysql-signup-verification-api'
Executing (default): CREATE TABLE IF NOT EXISTS `refreshTokens` (`id` INTEGER NOT NULL auto_increment , `toke
n` VARCHAR(255), `expires` DATETIME, `created` DATETIME NOT NULL, `createdByIp` VARCHAR(255), `revoked` DATET
IME, `revokedByIp` VARCHAR(255), `replacedByToken` VARCHAR(255), `accountId` INTEGER, PRIMARY KEY (`id`), FOR
EIGN KEY (`accountId`) REFERENCES `accounts` (`id`) ON DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `refreshTokens`
```

You can test the API directly with a tool such as Postman or VSCode extension ThunderClient. This time around we use Thunder Client.


Thunder Client v2.21.0

Thunder Client thunderclient.com | 3,601,882 | ★★★★☆ (340)

Lightweight Rest API Client for VS Code

This extension is enabled globally.

Thunder Client

Thunder Client is a lightweight Rest API Client Extension for VS Code, hand-crafted by Ranga Vadhineni with a focus on **simplicity, clean design and local storage**.

- Featured on Product Hunt
- Website - www.thunderclient.com
- Documentation: docs.thunderclient.com
- Support: github.com/rangav/thunder-client-support

Story behind Thunder Client

Categories

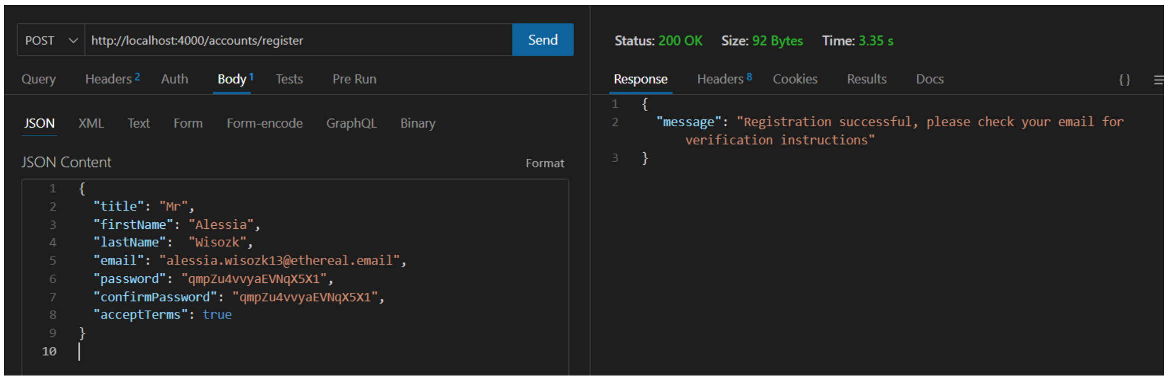
Programming Languages | Snippets | Testing

Resources

Marketplace
Issues
Repository
License
Thunder Client

To register a new account with the Node.js boilerplate api follow these steps:

- Open a new request tab by clicking the plus (+) button at the end of the tabs.
- Change the http request method to "POST" with the dropdown selector on the left of the URL input field.
- In the URL field enter the address to the register route of your local API - http://localhost:4000/accounts/register
- Select the "Body" tab below the URL field, change the body type radio button to "raw", and change the format dropdown selector to "JSON".
- Enter a JSON object containing the required user properties in the "Body" textarea, e.g:
- Click the "Send" button, you should receive a "200 OK" response with a "registration successful" message in the response body.



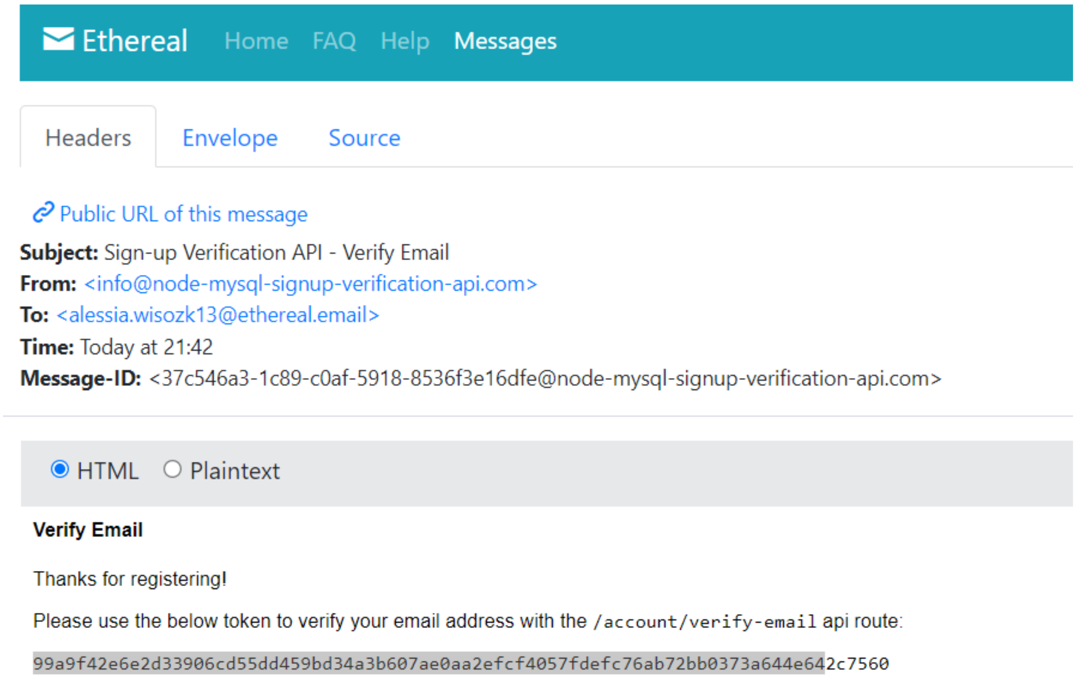
http://localhost:4000/accounts/register

```
{
  "title": "Mr",
  "firstName": "Alessia",
  "lastName":  "Wisozk",
  "email": "alessia.wisozk13@ethereal.email",
  "password": "qmpZu4vvyaEVNqX5X1",
  "confirmPassword": "qmpZu4vvyaEVNqX5X1",
  "acceptTerms": true
}
```

Status: 200 OK Size: 92 Bytes Time: 3.35 s

```
{
  "message": "Registration successful, please check your email for
     verification instructions"
}
```

Received a verification email with the token to verify the newly created account.



Subject: Sign-up Verification API - Verify Email
From: <info@node-mysql-signup-verification-api.com>
To: <alessia.wisozk13@ethereal.email>
Time: Today at 21:42
Message-ID: <37c546a3-1c89-c0af-5918-8536f3e16dfe@node-mysql-signup-verification-api.com>

Verify Email

Thanks for registering!

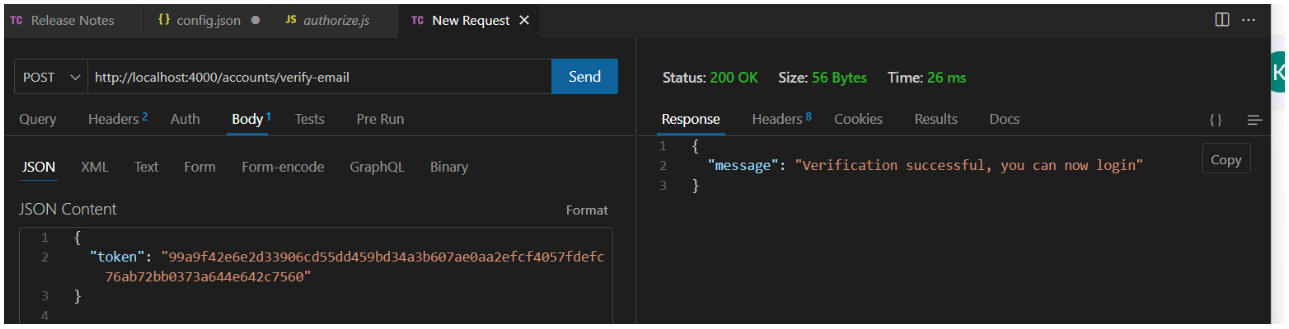
Please use the below token to verify your email address with the `/account/verify-email` api route:

`99a9f42e6e2d33906cd55dd459bd34a3b607ae0aa2efcf4057fdefc76ab72bb0373a644e642c7560`

To verify an account with the Node api follow these steps:

- Open a new request tab by clicking the plus (+) button at the end of the tabs.
- Change the http request method to "POST" with the dropdown selector on the left of the URL input field.
- In the URL field enter the address to the authenticate route of your local API - http://localhost:4000/accounts/verify-email

- Select the "Body" tab below the URL field, change the body type radio button to "raw", and change the format dropdown selector to "JSON".
- Enter a JSON object containing the token received in the verification email (in the previous step) in the "Body" textarea, e.g:
- Click the "Send" button, you should receive a "200 OK" response with a "verification successful" message in the response body.

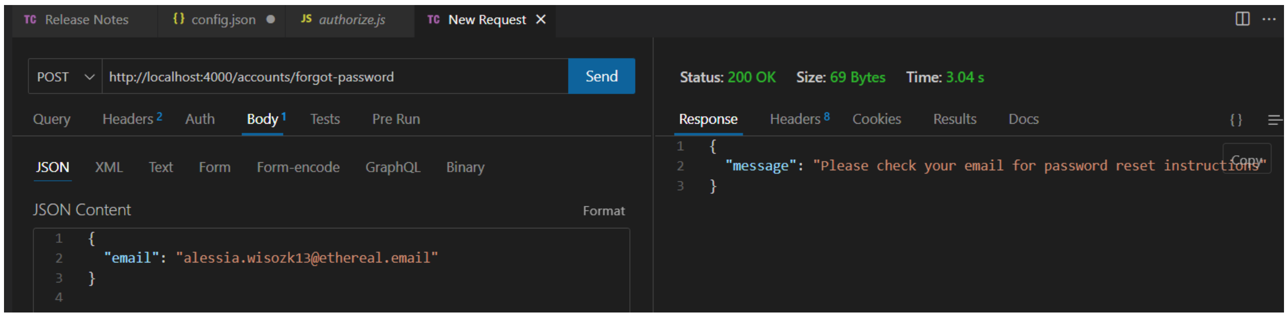


```
{
  "token": "99a9f42e6e2d33906cd55dd459bd34a3b607ae0aa2efcf4057fdefc
    76ab72bb0373a644e642c7560"
}
```

```
{
  "message": "Verification successful, you can now login"
}
```

Follow these steps in ThunderClient if you forgot the password for an account:

- Open a new request tab by clicking the plus (+) button at the end of the tabs.
- Change the http request method to "POST" with the dropdown selector on the left of the URL input field.
- In the URL field enter the address to the authenticate route of your local API - http://localhost:4000/accounts/forgot-password
- Select the "Body" tab below the URL field, change the body type radio button to "raw", and change the format dropdown selector to "JSON".
- Enter a JSON object containing the email of the account with the forgotten password in the "Body" textarea, e.g:
- Click the "Send" button, you should receive a "200 OK" response with the message "Please check your email for password reset instructions" in the response body.



POST http://localhost:4000/accounts/forgot-password Send

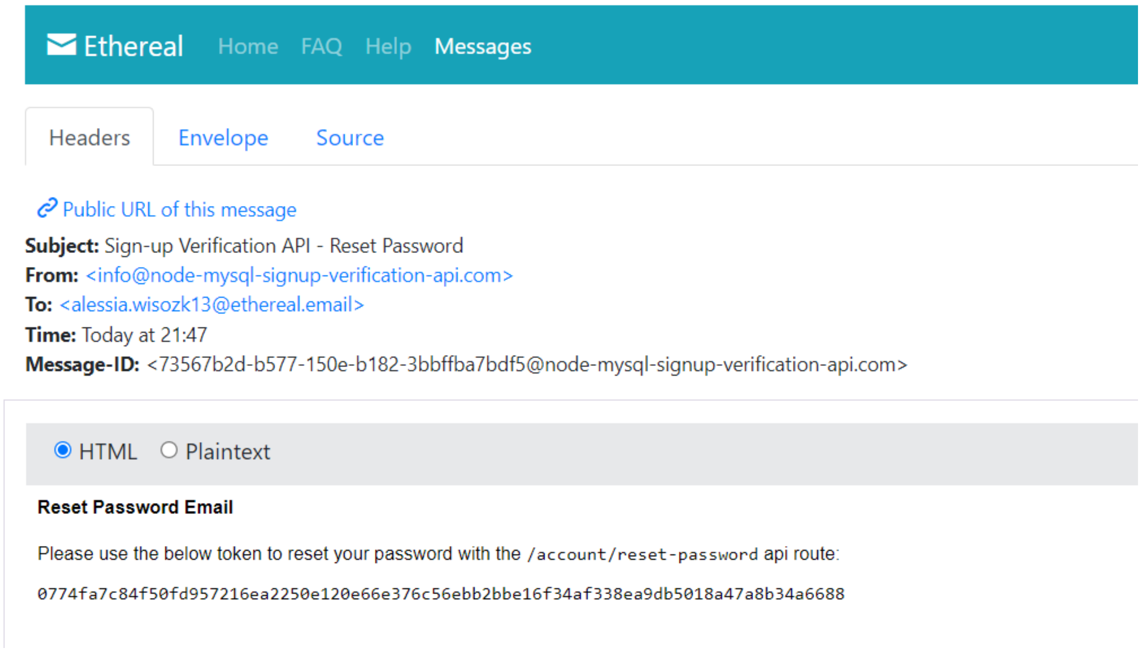
JSON Content

```
{
  "email": "alessia.wisozk13@ethereal.email"
}
```

Status: 200 OK Size: 69 Bytes Time: 3.04 s

```
{
  "message": "Please check your email for password reset instructions"
}
```

Received a verification email with the token to reset the password of the account.



Subject: Sign-up Verification API - Reset Password
From: <info@node-mysql-signup-verification-api.com>
To: <alessia.wisozk13@ethereal.email>
Time: Today at 21:47
Message-ID: <73567b2d-b577-150e-b182-3bbffba7bdf5@node-mysql-signup-verification-api.com>

Reset Password Email

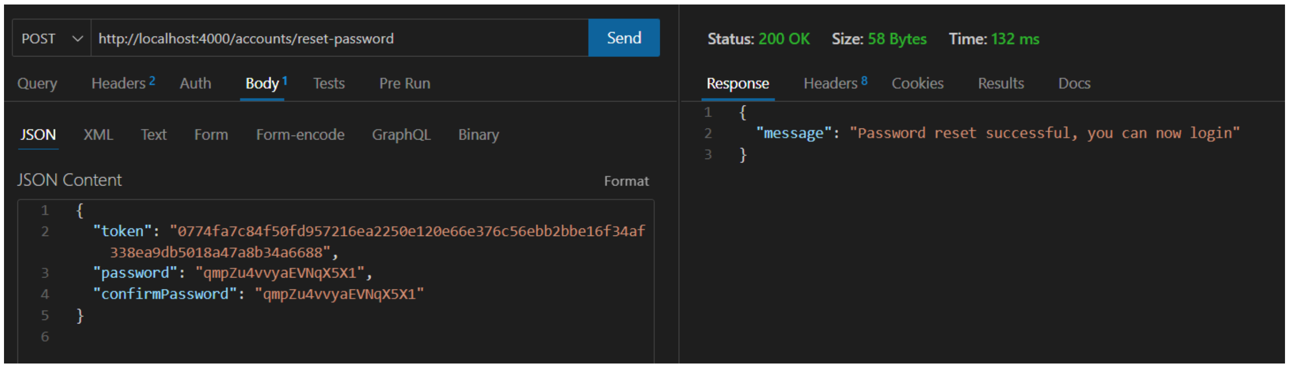
Please use the below token to reset your password with the `/account/reset-password` api route:

`0774fa7c84f50fd957216ea2250e120e66e376c56ebb2bbe16f34af338ea9db5018a47a8b34a6688`

To reset the password of an account with the api follow these steps:

- Open a new request tab by clicking the plus (+) button at the end of the tabs.
- Change the http request method to "POST" with the dropdown selector on the left of the URL input field.
- In the URL field enter the address to the authenticate route of your local API - http://localhost:4000/accounts/reset-password
- Select the "Body" tab below the URL field, change the body type radio button to "raw", and change the format dropdown selector to "JSON".
- Enter a JSON object containing the password reset token received in the email from the forgot password step, along with a new password and matching confirmPassword, into the "Body" textarea, e.g:

- Click the "Send" button, you should receive a "200 OK" response with the message "Please check your email for password reset instructions" in the response body.

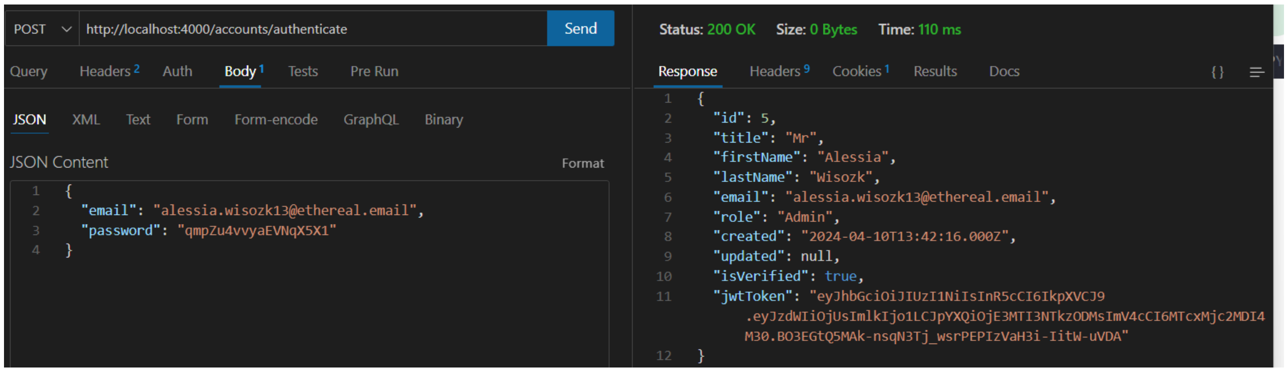

POST
http://localhost:4000/accounts/reset-password
Send
Query Headers 2 Auth Body 1 Tests Pre Run
JSON XML Text Form Form-encode GraphQL Binary
JSON Content
Format
Status: 200 OK Size: 58 Bytes Time: 132 ms
Response Headers 8 Cookies Results Docs

```
{
  "token": "0774fa7c84f50fd957216ea2250e120e66e376c56ebb2bbe16f34af
    338ea9db5018a47a8b34a6688",
  "password": "qmpZu4vvyaEVNqX5X1",
  "confirmPassword": "qmpZu4vvyaEVNqX5X1"
}
```

```
{
  "message": "Password reset successful, you can now login"
}
```

To authenticate an account with the api and get a JWT token follow these steps:

- Open a new request tab by clicking the plus (+) button at the end of the tabs.
- Change the http request method to "POST" with the dropdown selector on the left of the URL input field.
- In the URL field enter the address to the authenticate route of your local API - http://localhost:4000/accounts/authenticate
- Select the "Body" tab below the URL field, change the body type radio button to "raw", and change the format dropdown selector to "JSON".
- Enter a JSON object containing the account email and password in the "Body" textarea:
- Click the "Send" button, you should receive a "200 OK" response with a "password reset successful" message in the response body.
- Copy the JWT token value because we'll be using it in the next steps to make authenticated requests.



POST
http://localhost:4000/accounts/authenticate
Send
Query Headers 2 Auth Body 1 Tests Pre Run
JSON XML Text Form Form-encode GraphQL Binary
JSON Content
Format

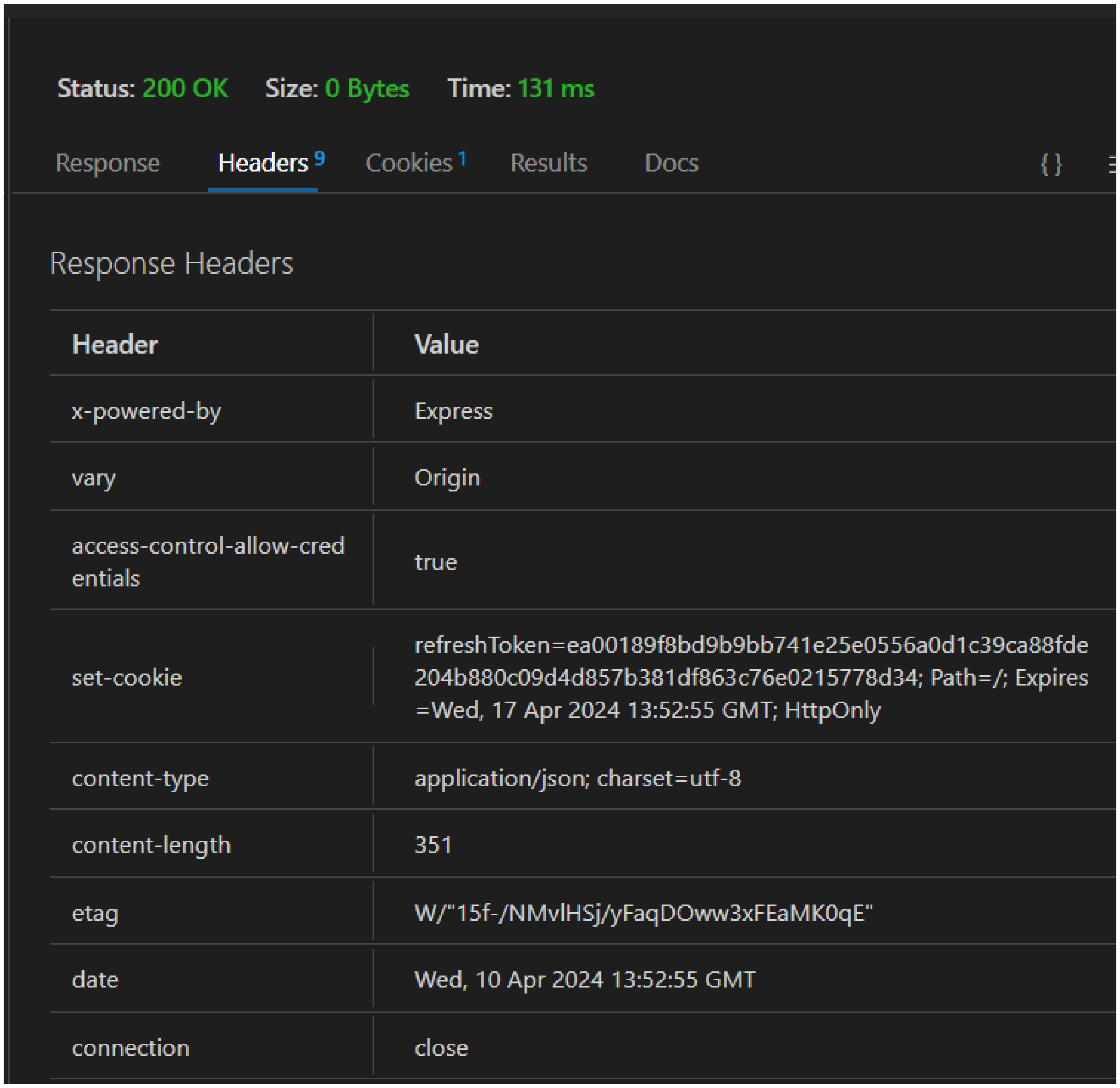
```
{
  "email": "alessia.wisozk13@ethereal.email",
  "password": "qmpZu4vvyaEVNqX5X1"
}
```

Status: 200 OK Size: 0 Bytes Time: 110 ms

Response Headers 9 Cookies 1 Results Docs

```
{
  "id": 5,
  "title": "Mr",
  "firstName": "Alessia",
  "lastName": "Wisozk",
  "email": "alessia.wisozk13@ethereal.email",
  "role": "Admin",
  "created": "2024-04-10T13:42:16.000Z",
  "updated": null,
  "isVerified": true,
  "jwtToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
     .eyJzdWIiOjUsImlkIjo1LCJpYXQiOjE3MTI3NTkzODMsImV4cCI6MTcxMjc2MDI4
     M30.BO3EGtQ5MAk-nsqN3Tj_wsrPEPIzVaH3i-IitW-uVDA"
}
```

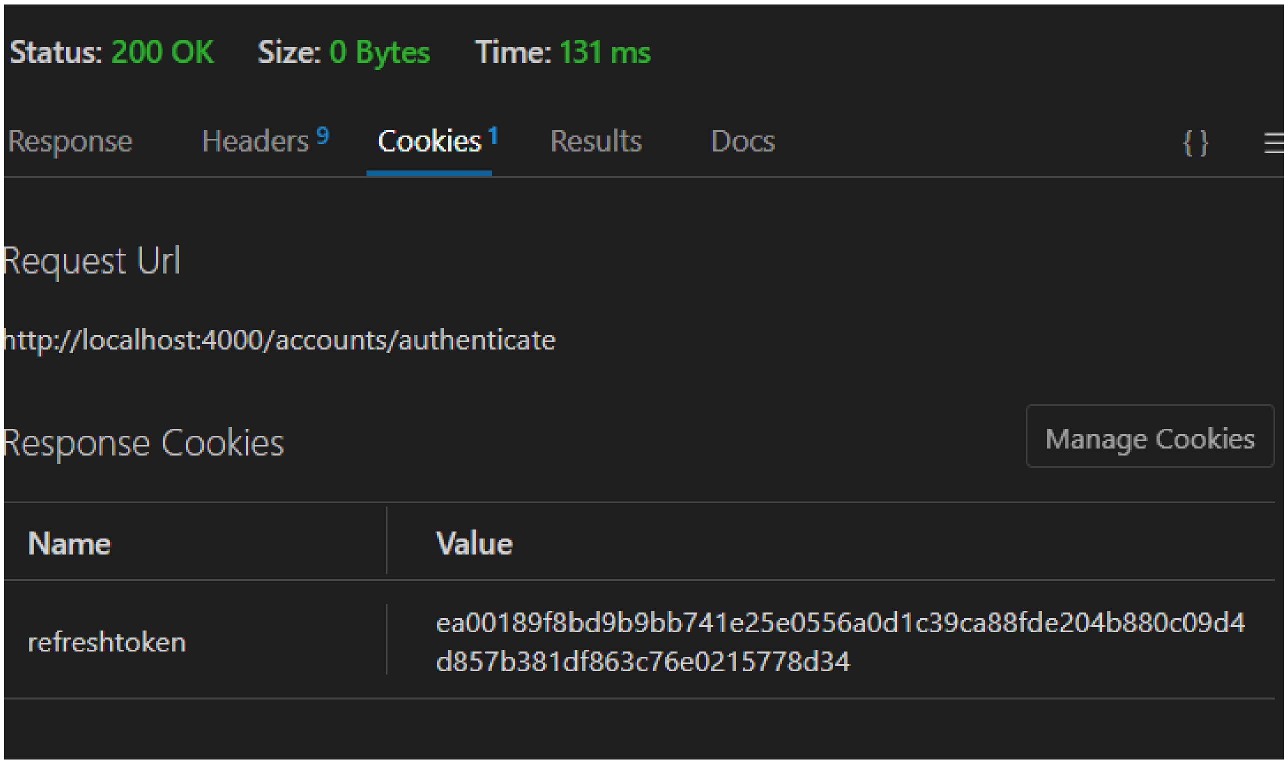
Response from the Headers and Cookies tab with the refresh token



Status: 200 OK Size: 0 Bytes Time: 131 ms

Response Headers

Header	Value
x-powered-by	Express
vary	Origin
access-control-allow-credentials	true
set-cookie	refreshToken=ea00189f8bd9b9bb741e25e0556a0d1c39ca88fde204b880c09d4d857b381df863c76e0215778d34; Path=/; Expires=Wed, 17 Apr 2024 13:52:55 GMT; HttpOnly
content-type	application/json; charset=utf-8
content-length	351
etag	W/"15f-/NMvlHSj/yFaqDOww3xFEaMK0qE"
date	Wed, 10 Apr 2024 13:52:55 GMT
connection	close


Status: 200 OK Size: 0 Bytes Time: 131 ms

Request Url

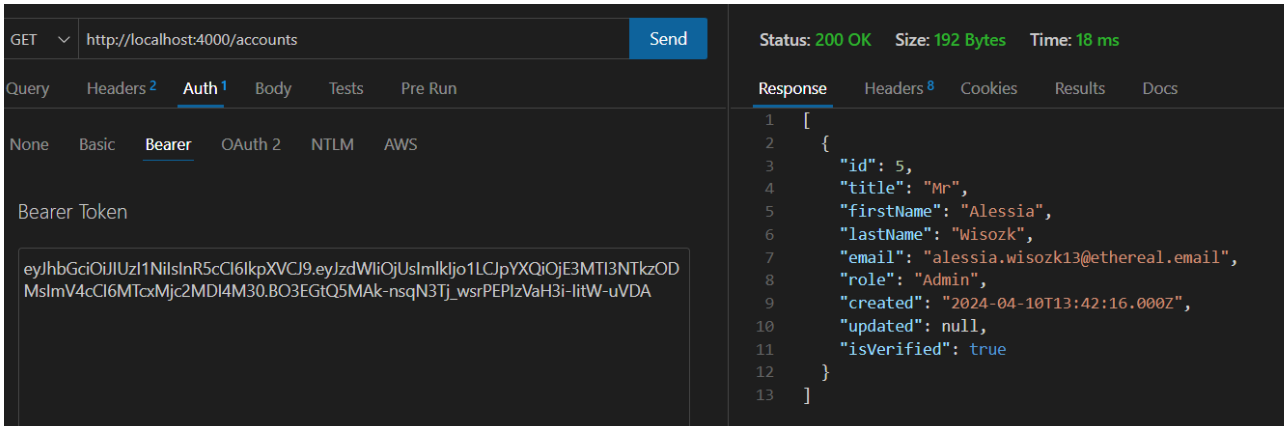
http://localhost:4000/accounts/authenticate

Response Cookies

Name	Value
refreshtoken	ea00189f8bd9b9bb741e25e0556a0d1c39ca88fde204b880c09d4d857b381df863c76e0215778d34

To get a list of all accounts from the Node boilerplate api follow these steps:

- Open a new request tab by clicking the plus (+) button at the end of the tabs.
- Change the http request method to "GET" with the dropdown selector on the left of the URL input field.
- In the URL field enter the address to the users route of your local API - http://localhost:4000/accounts
- Select the "Authorization" tab below the URL field, change the type to "Bearer Token" in the type dropdown selector, and paste the JWT token from the previous authenticate step into the "Token" field.
- Click the "Send" button, you should receive a "200 OK" response containing a JSON array with all of the account records in the system.

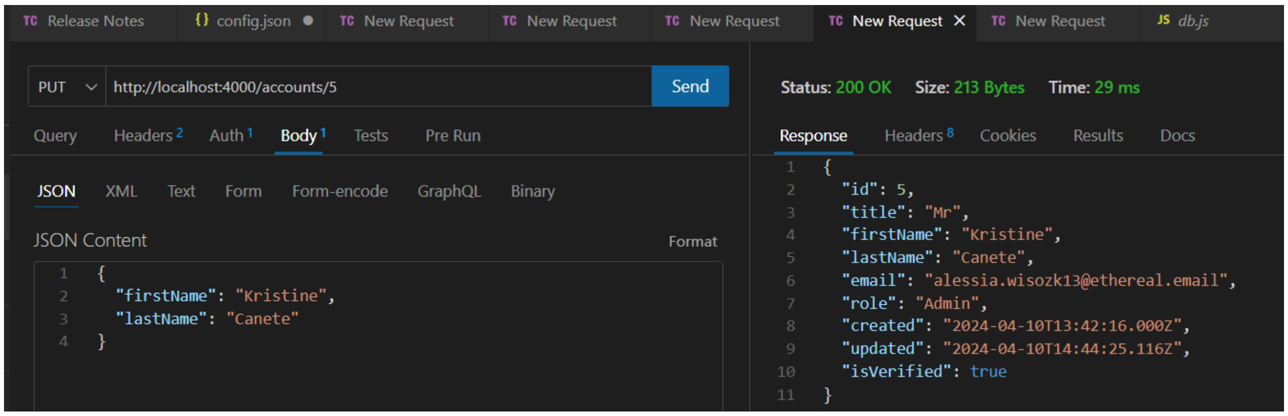


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOjUsImlkIjo1LCJpYXQiOjE3MTI3NTkzOD
MsImV4cCI6MTcxMjc2MDI4M30.BO3EGtQ5MAk-nsqN3Tj_wsrPEPIzVaH3i-litW-uVDA
```

```
[
  {
    "id": 5,
    "title": "Mr",
    "firstName": "Alessia",
    "lastName": "Wisozk",
    "email": "alessia.wisozk13@ethereal.email",
    "role": "Admin",
    "created": "2024-04-10T13:42:16.000Z",
    "updated": null,
    "isVerified": true
  }
]
```

To update an account with the api follow these steps:

- Open a new request tab by clicking the plus (+) button at the end of the tabs.
- Change the http request method to "PUT" with the dropdown selector on the left of the URL input field.
- In the URL field enter the address to the /accounts/{id} route with the id of the account you want to update, e.g - http://localhost:4000/accounts/1
- Select the "Authorization" tab below the URL field, change the type to "Bearer Token" in the type dropdown selector, and paste the JWT token from the previous authenticate step into the "Token" field.
- Select the "Body" tab below the URL field, change the body type radio button to "raw", and change the format dropdown selector to "JSON".
- Enter a JSON object in the "Body" textarea containing the properties you want to update, for example to update the first and last names:
- Click the "Send" button, you should receive a "200 OK" response with the updated account details in the response body.



PUT http://localhost:4000/accounts/5

```
{
  "firstName": "Kristine",
  "lastName": "Canete"
}
```

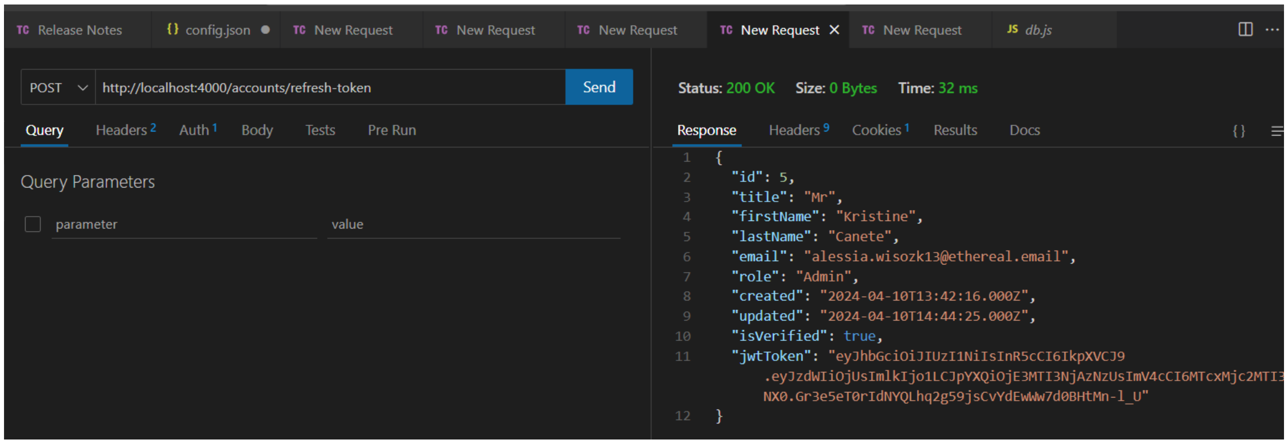
Status: 200 OK Size: 213 Bytes Time: 29 ms

```
{
  "id": 5,
  "title": "Mr",
  "firstName": "Kristine",
  "lastName": "Canete",
  "email": "alessia.wisozk13@ethereal.email",
  "role": "Admin",
  "created": "2024-04-10T13:42:16.000Z",
  "updated": "2024-04-10T14:44:25.116Z",
  "isVerified": true
}
```

To use a refresh token cookie to get a new JWT token and a new refresh token follow these steps:

- Open a new request tab by clicking the plus (+) button at the end of the tabs.
- Change the http request method to "POST" with the dropdown selector on the left of the URL input field.
- In the URL field enter the address to the refresh token route of your local API - http://localhost:4000/accounts/refresh-token
- Click the "Send" button, you should receive a "200 OK" response with the account details including a new JWT token in the response body and a new refresh token in the response cookies.
- Copy the JWT token value because we'll be using it in the next steps to make authenticated requests.

Response after the request is sent and the token has been refreshed

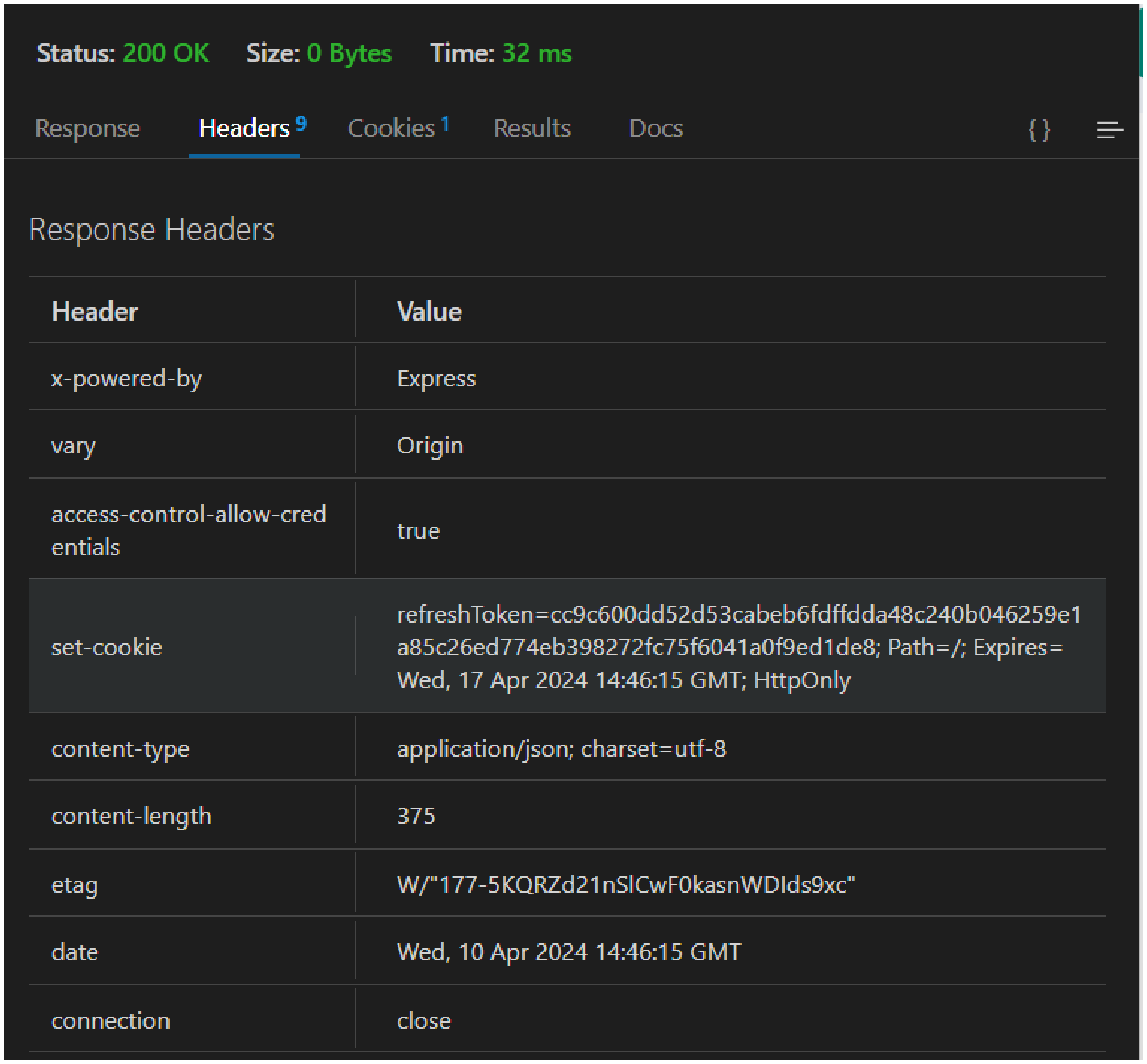


POST http://localhost:4000/accounts/refresh-token

Status: 200 OK Size: 0 Bytes Time: 32 ms

```
{
  "id": 5,
  "title": "Mr",
  "firstName": "Kristine",
  "lastName": "Canete",
  "email": "alessia.wisozk13@ethereal.email",
  "role": "Admin",
  "created": "2024-04-10T13:42:16.000Z",
  "updated": "2024-04-10T14:44:25.000Z",
  "isVerified": true,
  "jwtToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
      .eyJzdWIiOjUsImlkIjo1LCJpYXQiOjE3MTI3NjAzNzUsImV4cCI6MTcxMjc2MTI3
      NX0.Gr3e5eT0rIdNYQLhq2g59jsCvYdEwWw7d0BHtMn-l_U"
}
```

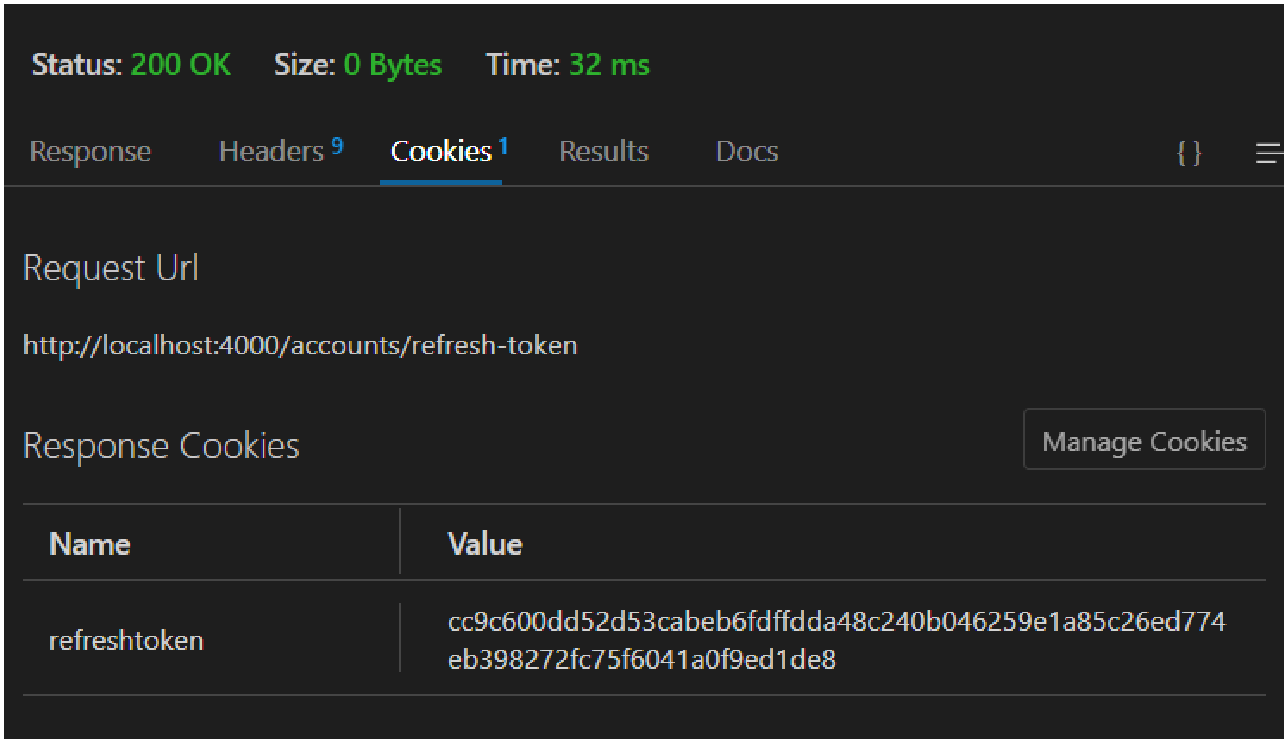
Response from the Headers and Cookies tab with the newly refreshed token



Status: 200 OK Size: 0 Bytes Time: 32 ms

Response Headers

Header	Value
x-powered-by	Express
vary	Origin
access-control-allow-credentials	true
set-cookie	refreshToken=cc9c600dd52d53cabeb6fdffdda48c240b046259e1a85c26ed774eb398272fc75f6041a0f9ed1de8; Path=/; Expires=Wed, 17 Apr 2024 14:46:15 GMT; HttpOnly
content-type	application/json; charset=utf-8
content-length	375
etag	W/"177-5KQRZd21nSlCwF0kasnWDIds9xc"
date	Wed, 10 Apr 2024 14:46:15 GMT
connection	close



Status: 200 OK Size: 0 Bytes Time: 32 ms

Request Url

http://localhost:4000/accounts/refresh-token

Response Cookies

Name	Value
refreshtoken	cc9c600dd52d53cabeb6fdffdda48c240b046259e1a85c26ed774eb398272fc75f6041a0f9ed1de8