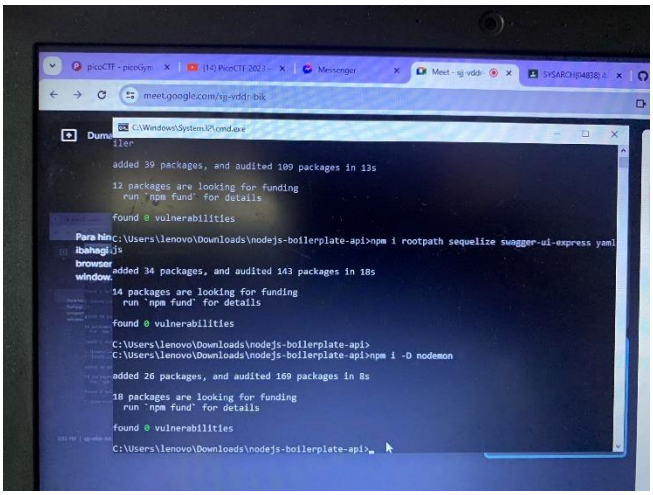
DUMALI, HESPHER T IT-INTPROG BSIT 3

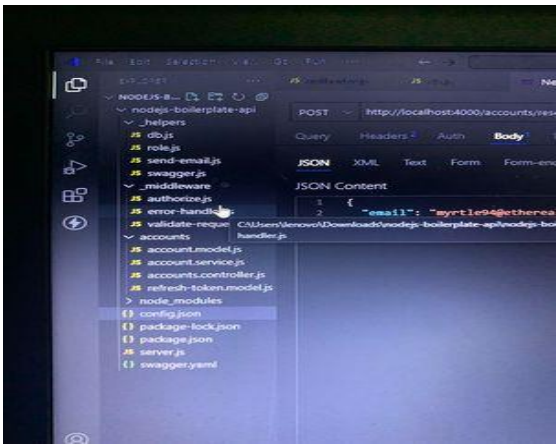
Node.JS + MySQL - Boilerplate API with Email Sign Up & Verification, Authentication & Forgot Password



Project Structure

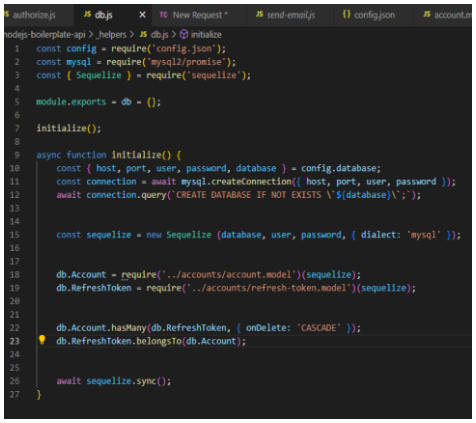
The Project is composed of multiple files and directories, each serving their own purpose.

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

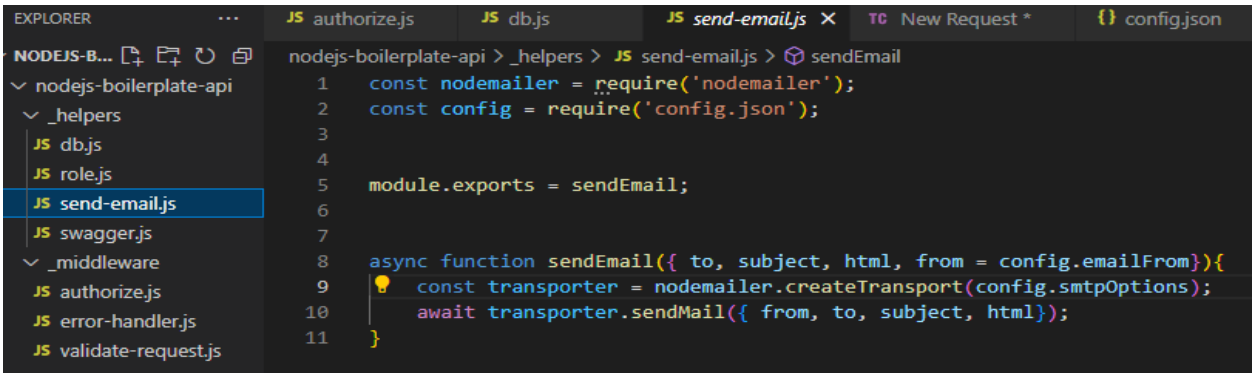


This is the Project Structure where the feature folders (accounts) and non-feature / shared component folders (_helpers, _middleware). Shared component folders contain code that can be used by multiple features and other parts of the application, and are prefixed with an underscore _ to group them together and make it easy to differentiate between feature folders and non-feature folders.

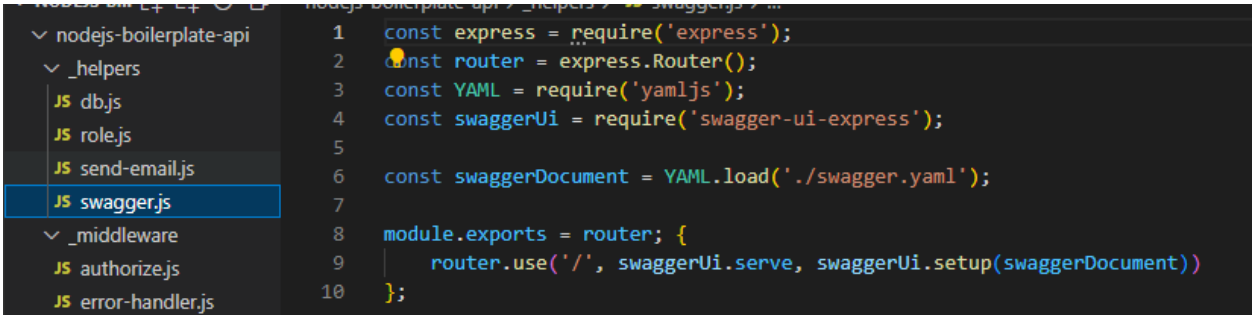
The boilerplate example only contains a single (accounts) feature at the moment, but could be easily extended with other features by copying the accounts folder and following the same pattern.



```
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();
}
```



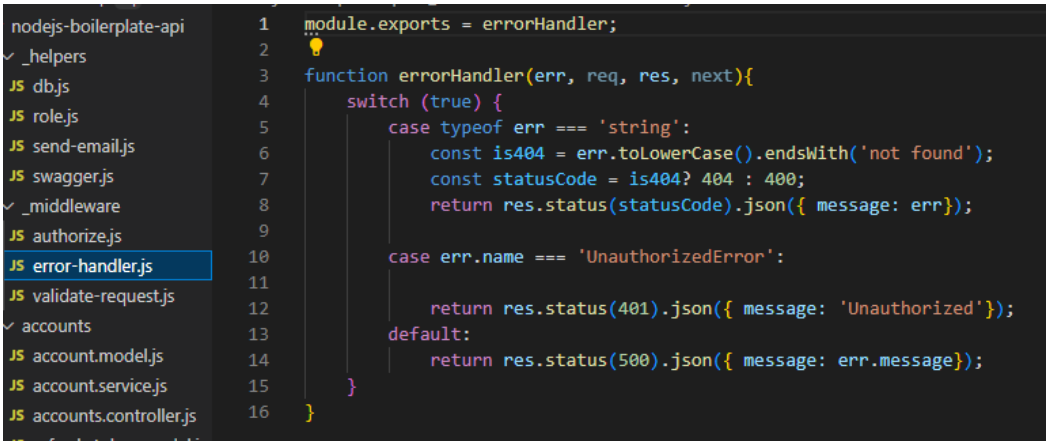
```
EXPLORER
NODEJS-B...
nodejs-boilerplate-api
  _helpers
    db.js
    role.js
    send-email.js
    swagger.js
  _middleware
    authorize.js
    error-handler.js
    validate-request.js

authorize.js | db.js | send-email.js | New Request * | config.json
nodejs-boilerplate-api > _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});
}
```



```
nodejs-boilerplate-api
  _helpers
    db.js
    role.js
    send-email.js
    swagger.js
  _middleware
    authorize.js
    error-handler.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)

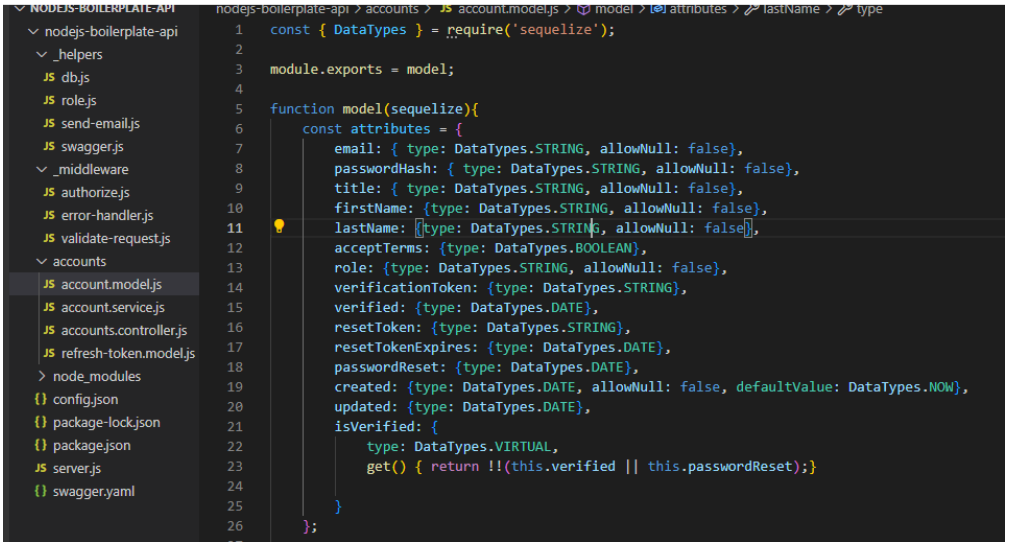
```
nodejs-boilerplate-api > _middleware > JS authorize.js > ...
const jwt = require('express-jwt');
const { secret } = require('config.json');
const db = require('_helpers/db');

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();
        }
    ];
}
```

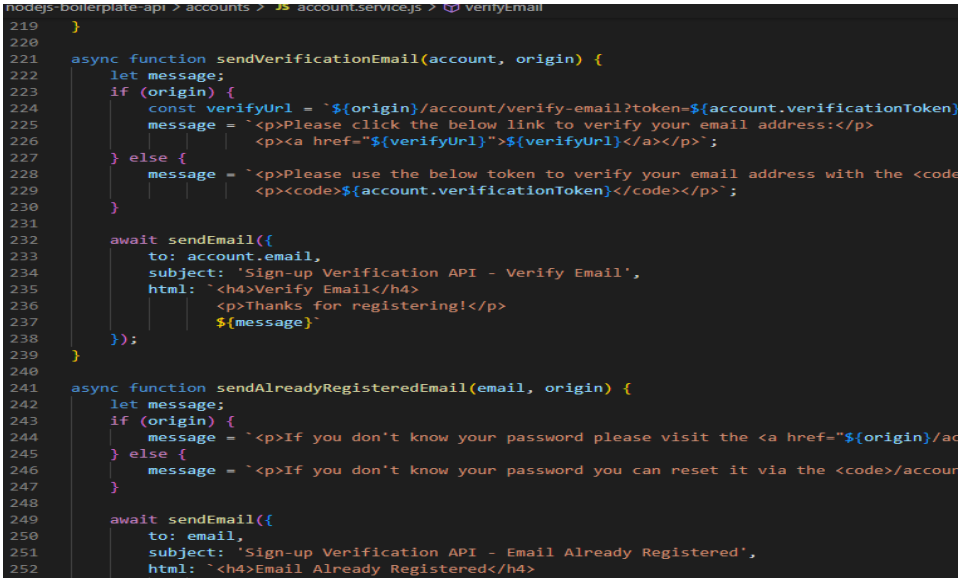


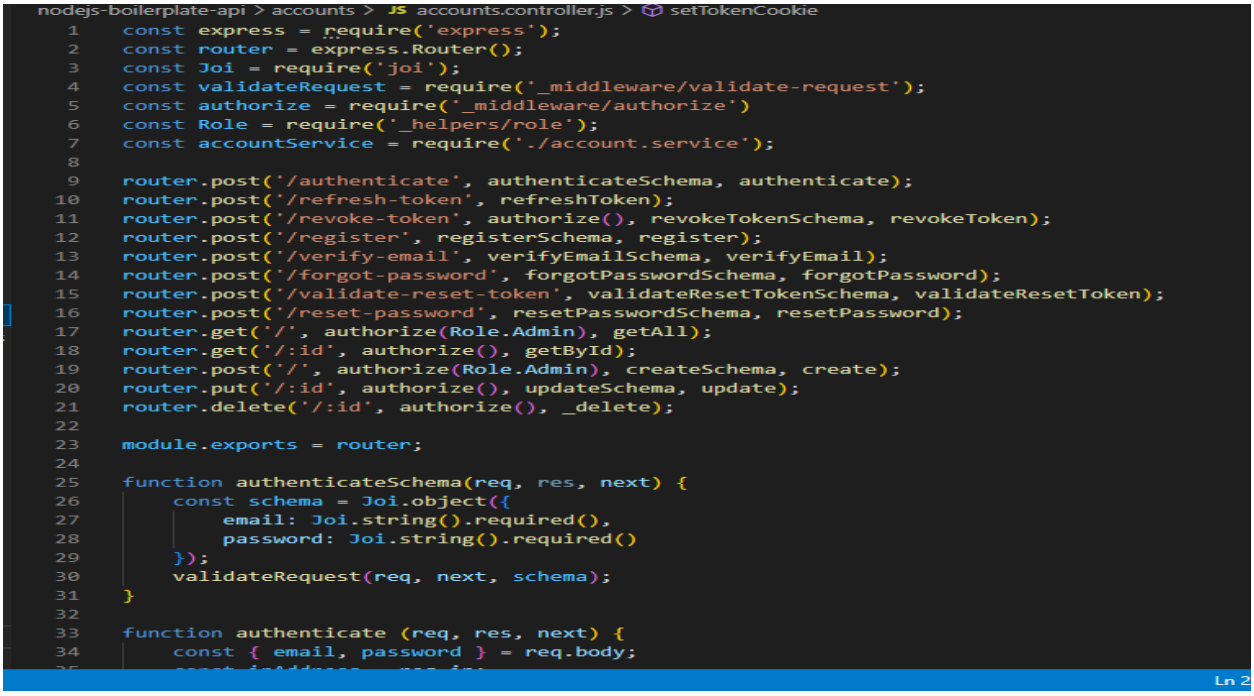
```
nodejs-boilerplate-api
_helpers
  db.js
  role.js
  send-email.js
  swagger.js
_middleware
  authorize.js
  error-handler.js
  validate-request.js
accounts
  account.model.js
  account.service.js
  accounts.controller.js
```

```
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});
    }
}
```

```
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();
    }
}
```


NODEJS-BOILERPLATE-API
nodejs-boilerplate-api > accounts > account.model.js > model > attributes > lastName > type
nodejs-boilerplate-api
_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
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);}
}
};


nodejs-boilerplate-api > accounts > account.service.js > verifyEmail
}
async function sendVerificationEmail(account, origin) {
let message;
if (origin) {
const verifyUrl = `${origin}/account/verify-email?token=${account.verificationToken}`;
message = ` Please click the below link to verify your email address:
${verifyUrl} `;
} else {
message = ` Please use the below token to verify your email address with the <code
<code>${account.verificationToken}</code> `;
}
await sendEmail({
to: account.email,
subject: 'Sign-up Verification API - Verify Email',
html: `<h4>Verify Email</h4>
Thanks for registering!
${message}`
});
}
async function sendAlreadyRegisteredEmail(email, origin) {
let message;
if (origin) {
message = ` If you don't know your password please visit the <a href="${origin}/ac
} else {
message = ` If you don't know your password you can reset it via the <code>/accour
}
await sendEmail({
to: email,
subject: 'Sign-up Verification API - Email Already Registered',
html: `<h4>Email Already Registered</h4>`


nodejs-boilerplate-api > accounts > accounts.controller.js > setTokenCookie
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;
Ln 2


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);
}

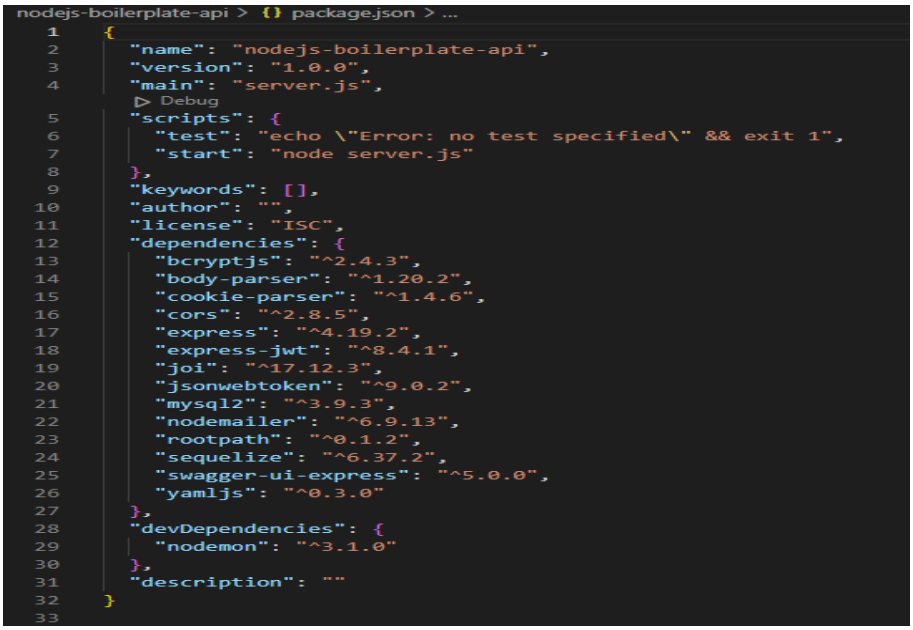
API Config (Path:/config.json)

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.

Package-lock.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.

Package.json

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



```
nodejs-boilerplate-api > {} package.json > ...
{
  "name": "nodejs-boilerplate-api",
  "version": "1.0.0",
  "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"
  },
  "description": ""
}
```

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

```
nodejs-boilerplate-api > JS 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', require('./accounts/accounts.controller'));

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 (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.

```yaml
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=51872eca5efedcf424db4cf5afd16a9d00ad25b743a034c9c221afc85d18dcd5e4ad
6e3f08607550; 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.eyJzdWIiOiI1ZWIxMmUxOTdlMDZhNzZjY2RlZmMxMjE
iLCJpZCI6IjVlYjEyZTE5N2UwNmE3NmNjZGVmYzEyMSIsImlhdCI6MTU4ODc1ODE1N30.xR9H0STbFOpS
kuGA9jHNZOJ6eS7umHHqKRhI807YT1Y"
        "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:
51872eca5efedcf424db4cf5afd16a9d00ad25b743a034c9c221afc85d18dcd5e4ad6e3f08607550
      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=51872eca5efedcf424db4cf5afd16a9d00ad25b743a034c9c221afc85d18dcd5e4ad
6e3f08607550; 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.eyJzdWIiOiI1ZWIxMmUxOTdlMDZhNzZjY2RlZmMxMjE
iLCJpZCI6IjVlYjEyZTE5N2UwNmE3NmNjZGVmYzEyMSIsImlhdCI6MTU4ODc1ODE1N30.xR9H0STbFOpS
kuGA9jHNZOJ6eS7umHHqKRhI807YT1Y"
        "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:
51872eca5efedcf424db4cf5afd16a9d00ad25b743a034c9c221afc85d18dcd5e4ad6e3f08607550
      requestBody:
        content:
          application/json:
            schema:
              type: object
              properties:
                token:
                  type: string
                  example:
"51872eca5efedcf424db4cf5afd16a9d00ad25b743a034c9c221afc85d18dcd5e4ad6e3f08607550
"
      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:
```

```yaml
        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:
"3c7f8d9c4cb348ff95a0b74a1452aa24fc9611bb76768bb9eafeeb826ddae2935f1880bc7713318f
"
              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:
"3c7f8d9c4cb348ff95a0b74a1452aa24fc9611bb76768bb9eafeeb826ddae2935f1880bc7713318f
"
```

```
              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:
"3c7f8d9c4cb348ff95a0b74a1452aa24fc9611bb76768bb9eafeeb826ddae2935f1880bc7713318f
"
                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:
      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
```

```yaml
          properties:
            message:
              type: string
              example: "Unauthorized"
  NotFoundError:
    description: Not Found
    content:
      application/json:
        schema:
          type: object
          properties:
            message:
              type: string
              example: "Not Found"
```