User management API

TypeScript, Fastify, TypeORM, MariaDB

COMP.SEC.300 Secure Programming / Exercise work

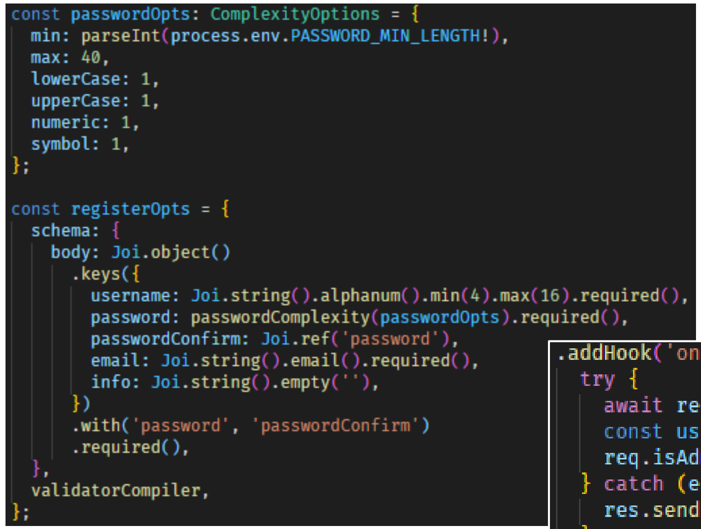
Valtteri Valtanen

Aim of the program / other goals

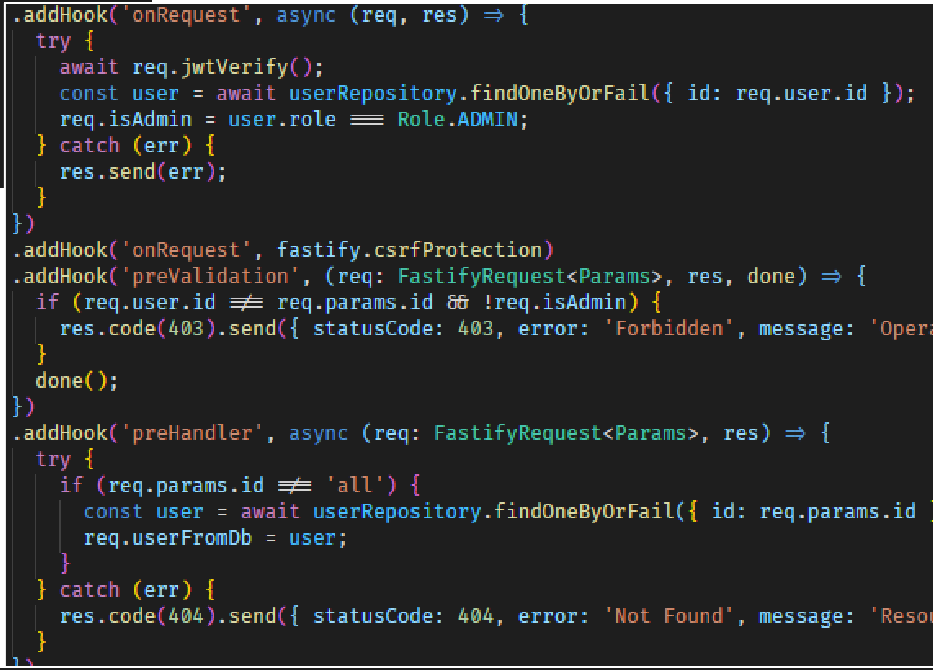
- Provide a simple user CRUD API with authentication
  - Register
  - Login
  - Roles (currently admin and user)
  - Managing users based on role
- Check out Fastify
- Implement API using security best practices

Most notable security features

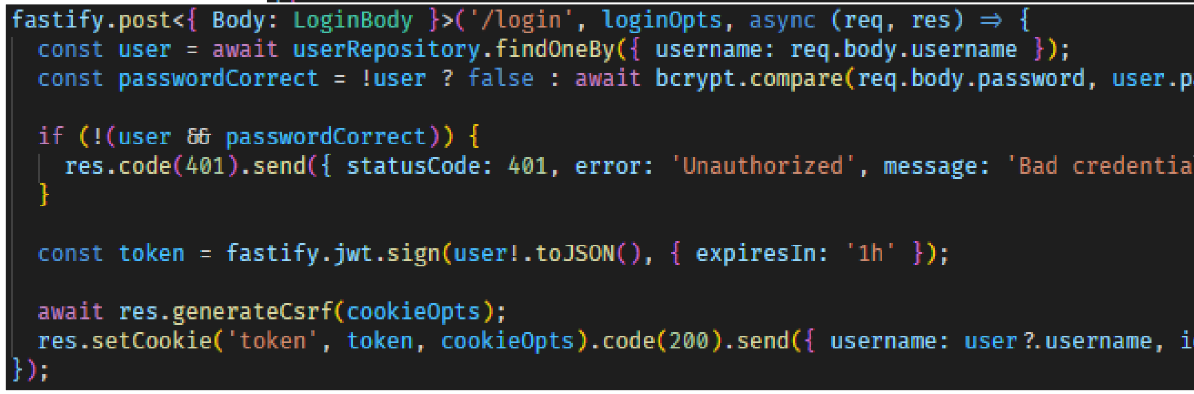
- Validate request bodies
- Signed httpOnly cookies
- Require strong passwords
- Bcrypt password hashing
- CORS configuration
- CSRF token
- Helmet for security headers
- Rate limiting to 50 requests in 1 min



```
const passwordOpts: ComplexityOptions = {
  min: parseInt(process.env.PASSWORD_MIN_LENGTH!),
  max: 40,
  lowerCase: 1,
  upperCase: 1,
  numeric: 1,
  symbol: 1,
};

const registerOpts = {
  schema: {
    body: Joi.object()
      .keys({
        username: Joi.string().alphanum().min(4).max(16).required(),
        password: passwordComplexity(passwordOpts).required(),
        passwordConfirm: Joi.ref('password'),
        email: Joi.string().email().required(),
        info: Joi.string().empty(''),
      })
      .with('password', 'passwordConfirm')
      .required(),
  },
  validatorCompiler,
};
```



```
.addHook('onRequest', async (req, res) => {
  try {
    await req.jwtVerify();
    const user = await userRepository.findOneByOrFail({ id: req.user.id });
    req.isAdmin = user.role === Role.ADMIN;
  } catch (err) {
    res.send(err);
  }
})
.addHook('onRequest', fastify.csrfProtection)
.addHook('preValidation', (req: FastifyRequest<Params>, res, done) => {
  if (req.user.id !== req.params.id && !req.isAdmin) {
    res.code(403).send({ statusCode: 403, error: 'Forbidden', message: 'Opera
  }
  done();
})
.addHook('preHandler', async (req: FastifyRequest<Params>, res) => {
  try {
    if (req.params.id !== 'all') {
      const user = await userRepository.findOneByOrFail({ id: req.params.id
      req.userFromDb = user;
    }
  } catch (err) {
    res.code(404).send({ statusCode: 404, error: 'Not Found', message: 'Reso
  }
```

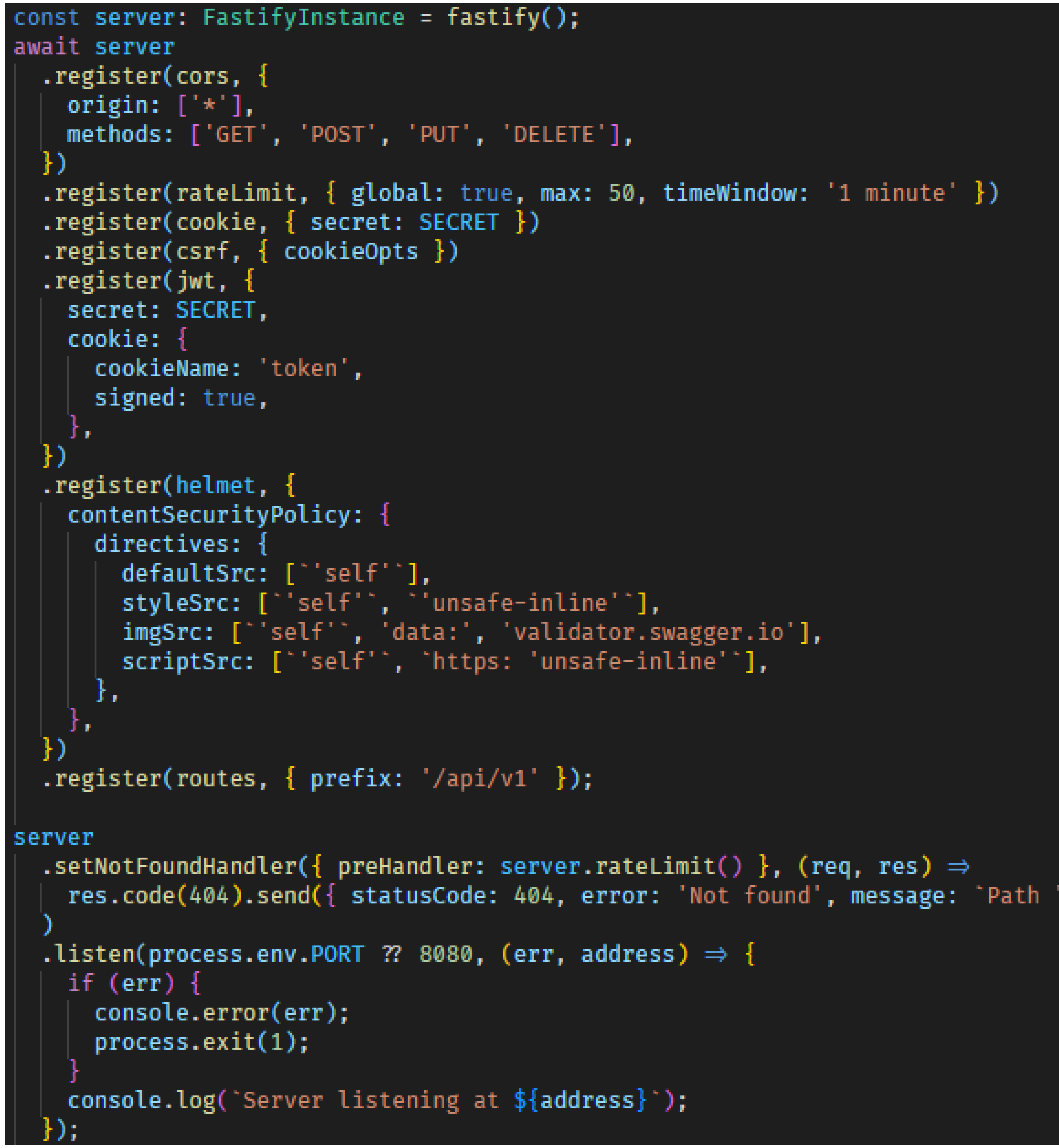


```
fastify.post<{ Body: LoginBody }>('/login', loginOpts, async (req, res) => {
  const user = await userRepository.findOneBy({ username: req.body.username });
  const passwordCorrect = !user ? false : await bcrypt.compare(req.body.password, user.p

  if (!(user && passwordCorrect)) {
    res.code(401).send({ statusCode: 401, error: 'Unauthorized', message: 'Bad credentia
  }

  const token = fastify.jwt.sign(user!.toJSON(), { expiresIn: '1h' });

  await res.generateCsrf(cookieOpts);
  res.setCookie('token', token, cookieOpts).code(200).send({ username: user?.username, i
});
```

The good

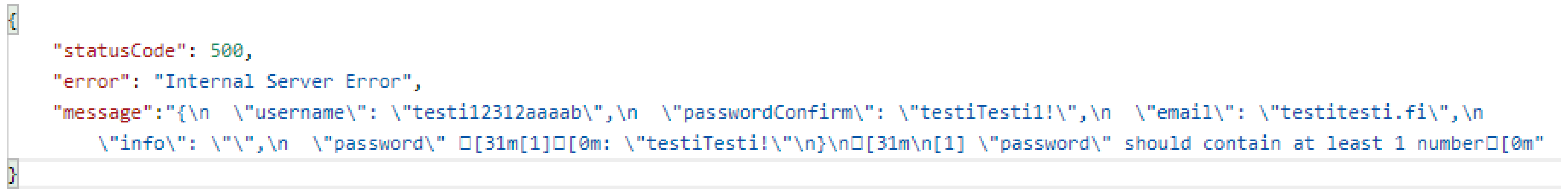
- Fastify is easy to use
  - Most of the commonly used, state-of-the-art stuff is already implemented, just plug-n-play
  - Because security features are easy to implement, there is no excuses for not doing it

- Implementing user auth manually is not very hard, at least on a rudimentary level



```
const server: FastifyInstance = fastify();
await server
  .register(cors, {
    origin: ['*'],
    methods: ['GET', 'POST', 'PUT', 'DELETE'],
  })
  .register(rateLimit, { global: true, max: 50, timeWindow: '1 minute' })
  .register(cookie, { secret: SECRET })
  .register(csrf, { cookieOpts })
  .register(jwt, {
    secret: SECRET,
    cookie: {
      cookieName: 'token',
      signed: true,
    },
  })
  .register(helmet, {
    contentSecurityPolicy: {
      directives: {
        defaultSrc: [`'self'`],
        styleSrc: [`'self'`, `'unsafe-inline'`],
        imgSrc: [`'self'`, 'data:', 'validator.swagger.io'],
        scriptSrc: [`'self'`, `https: 'unsafe-inline'`],
      },
    },
  })
  .register(routes, { prefix: '/api/v1' });

server
  .setNotFoundHandler({ preHandler: server.rateLimit() }, (req, res) =>
    res.code(404).send({ statusCode: 404, error: 'Not found', message: `Path
  )
  .listen(process.env.PORT ?? 8080, (err, address) => {
    if (err) {
      console.error(err);
      process.exit(1);
    }
    console.log(`Server listening at ${address}`);
  });
```

The bad



```
{
    "statusCode": 500,
    "error": "Internal Server Error",
    "message":"{\n  \"username\": \"testi12312aaaab\",\n  \"passwordConfirm\": \"testiTesti1!\",\n  \"email\": \"testitesti.fi\",\n
        \"info\": \"\",\n  \"password\" □[31m[1]□[0m: \"testiTesti!\"\n}\n□[31m\n[1] \"password\" should contain at least 1 number□[0m"
}
```

- Fastify and Joi didn’t integrate quite as seamlessly as hoped
  - Would require some *tunkkaus*, not in the scope of this project
- Although implementing user auth is doable, probably wouldn’t do it myself and use some IAM/SSO solution instead
- The application is architecture-wise not very well implemented, should be refactored if one would continue working on this
- HTTPS is a hassle to implement locally, so currently can’t test it
- No automated testing yet (automated postman api tests coming)

http://localhost:8080/api/v1/documentation