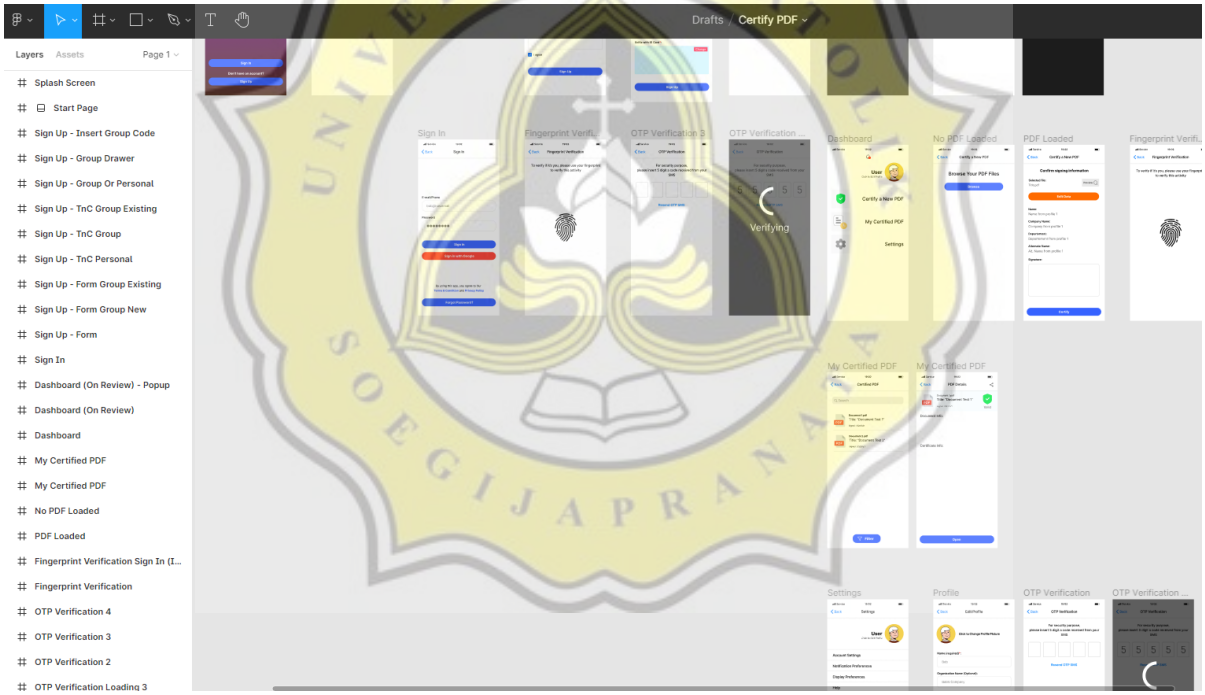
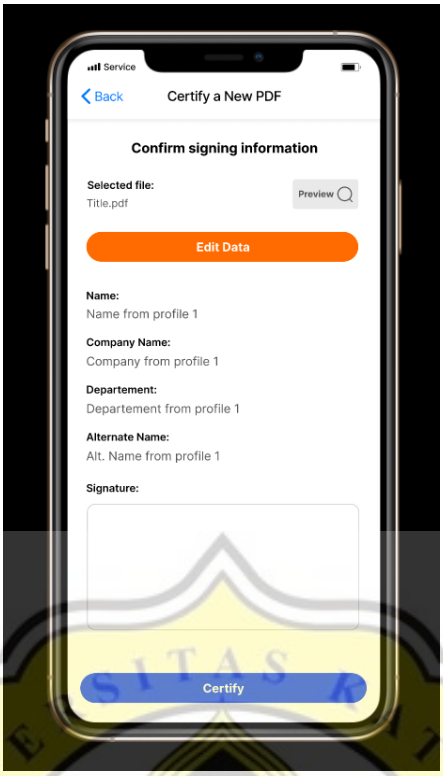
LAMPIRAN

A. Perancangan User Interface

Perancangan desain *Graphical User Interface* (GUI) dilakukan untuk membantu saat pengerjaan aplikasi yang akan dilakukan agar dapat mengerti bagaimana alur dari aplikasi baik itu *front-end* dan aplikasi *back-end* yang harus dibuat untuk memenuhi alur dari aplikasi yang dikehendaki. Figma dipilih sebagai *tools* yang berguna dalam perancangan desain dan alur sederhana dari aplikasi *front-end*. Walau pada akhirnya dalam eksekusi pembuatan aplikasi mungkin hasil tidak sesuai secara sepenuhnya dengan desain yang telah dibuat, setidaknya dengan adanya desain yang dibuat terlebih dahulu dapat membantu mengerti dalam menentukan inti alur yang ada di dalam aplikasi dan desain atau gaya tampilan yang dikehendaki untuk dibuat dan diterapkan ke dalam aplikasi. Contoh tampilan user interface figma dapat dilihat pada gambar A.1 dan gambar A.2.



Gambar A.1: Desain kasar melalui Figma.

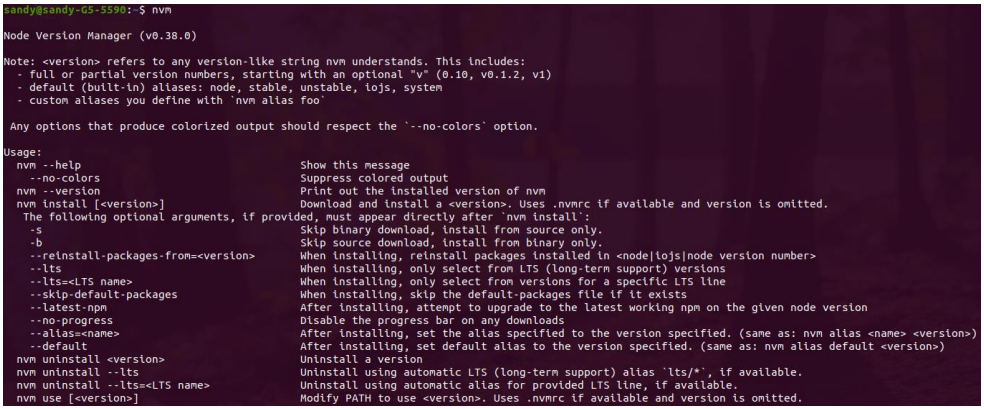


Gambar A.2: Preview design Figma secara interaktif.

B. Persiapan Pengembangan Sistem (Backend)

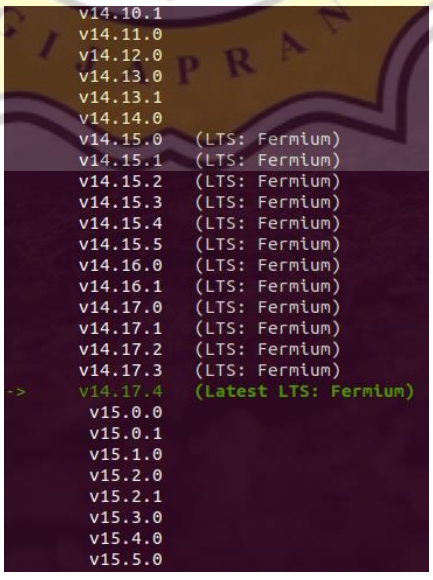
B.1 Node.Js

Node.Js dapat diunduh melalui situs resmi Node.Js secara langsung sesuai dengan sistem operasi yang digunakan, atau bisa juga menggunakan "Node Version Manager". Node Version Manager mempermudah dalam mengelola Node.Js yang akan digunakan dan bahkan dimungkinkan untuk menjalankan dua atau lebih versi Node.Js yang berbeda dalam satu komputer. Secara lengkapnya dapat dilihat melalui repository nvm pada https://github.com/nvm-sh/nvm. Berikut pada gambar B.2 adalah contoh mengakses NVM melalui *terminal*.

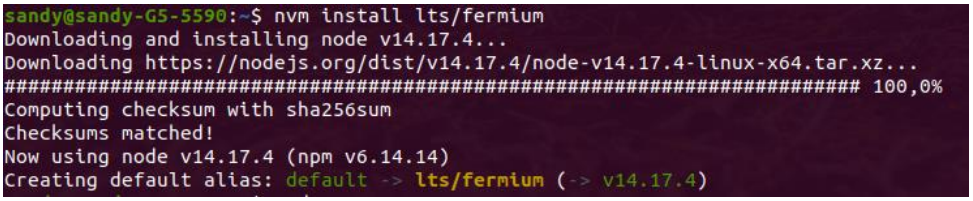


Gambar B.1: Tampilan nvm pada terminal

Setelah NVM terpasang, penginstalan Node.Js dapat dilakukan sesuai dengan versi yang diinginkan dengan cara memasukan perintah pada terminal "nvm install versi" di mana versi adalah versi dari Node.Js yang diinginkan. Apabila hendak mengunduh versi rilis terbaru dapat cukup dengan mengetikan "nvm install node" atau apabila hendak menginstall versi LTS (*Long Term Support*) "nvm install --lts". Untuk melihat semua *installer* yang tersedia dilakukan dengan cara mengetikan "nvm ls-remote" (seperti pada gambar B.2) dan untuk melihat daftar Node.Js yang telah terinstall pada sistem dapat mengetikan "nvm ls". Untuk memilih versi dari Node.Js yang hendak digunakan dilakukan dengan cara mengetikan "nvm use versi" di mana versi adalah versi yang hendak diaktifkan. Tetapi pada dasarnya, apabila menginstall Node.Js dengan menggunakan nvm pertama kali, maka Node.Js yang diinstall tersebut akan terpilih secara *default* sebagai versi yang digunakan.



Gambar B.2: Hasil perintah "nvm ls-remote"



Gambar B.3: Mengunduh dan menginstall Node versi lts/fermium

Apabila Node.Js sudah terpasang dan versi yang hendak digunakan telah aktif, pembuatan projek Node.Js dapat dimulai. Pertama buat *folder* baru dengan diberi nama sesuai dengan nama *project* yang hendak dibuat. Lalu masuk ke direktori *folder* tersebut melalui *terminal*. Pada *terminal*, ketikan "npm init" untuk membuat *file* "package.json" pada *project folder*. Kemudian terdapat beberapa informasi yang harus diisikan seperti nama *package* dari *project* yang hendak dibuat, deskripsi, versi, nama pembuat, dan sebagainya. Apabila *file* "package.json" telah sukses muncul pada direktori *project*, penginstalan *library* yang hendak digunakan dalam projek dapat dilakukan kemudian dapat dilanjutkan untuk menjalankan kode JavaScript sederhana.

Pada pengembangan aplikasi JavaScript dengan menggunakan Node.Js apabila hendak menginstal atau mengunduh sebuat *package* atau *library* terdapat dua *package manager library* yang biasanya cukup sering digunakan yaitu "npm" dan "yarn". Keduanya dapat dicoba mana yang lebih cocok selama melakukan pengembangan. Pada npm, untuk menginstal *package* atau *library* dapat dilakukan dengan cara mengetikan "npm install nama package" di mana "nama package" adalah nama dari *package* atau *library* yang hendak diunduh dan untuk menghapus package tersebut cukup dengan mengganti kata perintah "install" menjadi "uninstall". Apabila hendak menggunakan yarn penginstalan *library* dapat dilakukan dengan cara mengetikan "yarn add namapackage" seperti tadi bahwa "namapackage" adalah nama *package* atau *library* yang hendak diunduh. Dan untuk menghapus *package* atau *library* yang telah terinstall dapat dilakukan dengan mengganti kata perintah "add" menjadi "remove". Hanya saja untuk dapat menggunakan yarn harus diinstal terlebih dahulu secara *global* dengan mengetikan pada terminal "npm install -g yarn". Semua *package* atau *library* yang telah terinstall secara *local* pada *project folder* akan disimpan pada *folder* "node_modules" dan dapat dilihat pada *file* "package.json".

Beberapa *package* atau *library* yang digunakan dalam pengerjaan aplikasi *back-end* pada projek ini di antaranya adalah:

1. express

Digunakan sebagai *framework* dalam pembuatan sistem REST API dalam berkomunikasi dengan aplikasi *front-end*.

2. body-parser

   Middleware yang dapat digunakan pada express untuk mengakses *body* dari *request* yang diterima.

3. cors

   Middleware yang dapat digunakan pada express untuk dapat menggunakan CORS beserta pengaturannya.

4. dot-env

   Digunakan untuk mengakses *file* berisikan *environment data* dengan format ".env" seperti URL, *keys*, dan sebagainya. Digunakan agar *environment data* dapat disimpan secara terpisah dari *code* yang umumnya akan selalu diupdate ke sebuah *repository*. Data *environment* umumnya tidak disertakan ke dalam *repository* demi alasan keamanan yang mana biasanya kemudian akan dicatat pada *file* ".gitignore".

5. esm

   Untuk dapat menggunakan ECMAScript module pada Node.Js tanpa harus memisahkan *file* dengan ekstensi terpisah seperti ".mjs". *Library* digunakan agar saat menulis kode JavaScript dapat menggunakan perintah *import* JavaScript seperti biasa (require("")) dengan ES6 "import" dan "export" (import {} from "", dan export default modules). Cukup meningkatkan produktivitas agar seseorang dapat menulis code layaknya seperti saat sedang membuat aplikasi *front-end* menggunakan React yang juga menggunakan ES6 import.

6. node-fetch

   Membantu apabila hendak melakukan *fetch request* ke sebuah *server*.

7. morgan

   Membantu untuk menampilkan *logging* pada saat proses *development*.

8. node-forge

   Library ini menyediakan berbagai *utilities* yang dapat digunakan untuk melakukan aktivitas pembuatan atau pengelolaan yang berhubungan dengan *cryptography* seperti pembuatan sertifikat, konversi format sertifikat, pembuatan pasangan *private* dan *public key*, dan sebagainya.

9. grpc

Membantu untuk mengakses protokol gRPC seperti apabila hendak melakukan *request* ke gRPC API pada Hyperledger Iroha.

10. iroha-helpers

    Membantu dalam pembuatan *request* berupa "*commands*" dan "*queries*" ke Hyperledger Iroha.

11. mongoose

    Membantu dalam melakukan operasi dan komunikasi dengan *database* MongoDb.

12. @pdftron/pdfnet-node

    Membantu dalam pengolahan dan manipulasi PDF pada sistem *back-end*.

13. bcrypt

    Membantu dalam proses autentikasi dan pembuatan *password hash* yang lebih aman.

14. hasha

    Membantu proses *hashing* agar menjadi lebih mudah.

15. jsonwebtoken

    Membantu dalam proses pembuatan *token* dan otentikasi menggunakan JWT (JSON Web Token).

16. qrcode

    Membantu dalam pembuatan QR Code dengan format PNG atau SVG.

17. jimp

    Membantu dalam proses konversi gambar dengan berbagai format sesuai yang diinginkan. Dapat juga digunakan untuk melakukan manipulasi gambar yang lain seperti *crop*, *resize*, *blur*, dan sebagainya.

18. twilio

    Untuk menghubungkan dengan API Twilio yang akan digunakan untuk mengirimkan OTP SMS atau bisa juga digunakan untuk aktivitas lainnya apabila dibutuhkan di masa mendatang.

19. underscore

    Dependency dari package @pdftron/pdfnet-node.

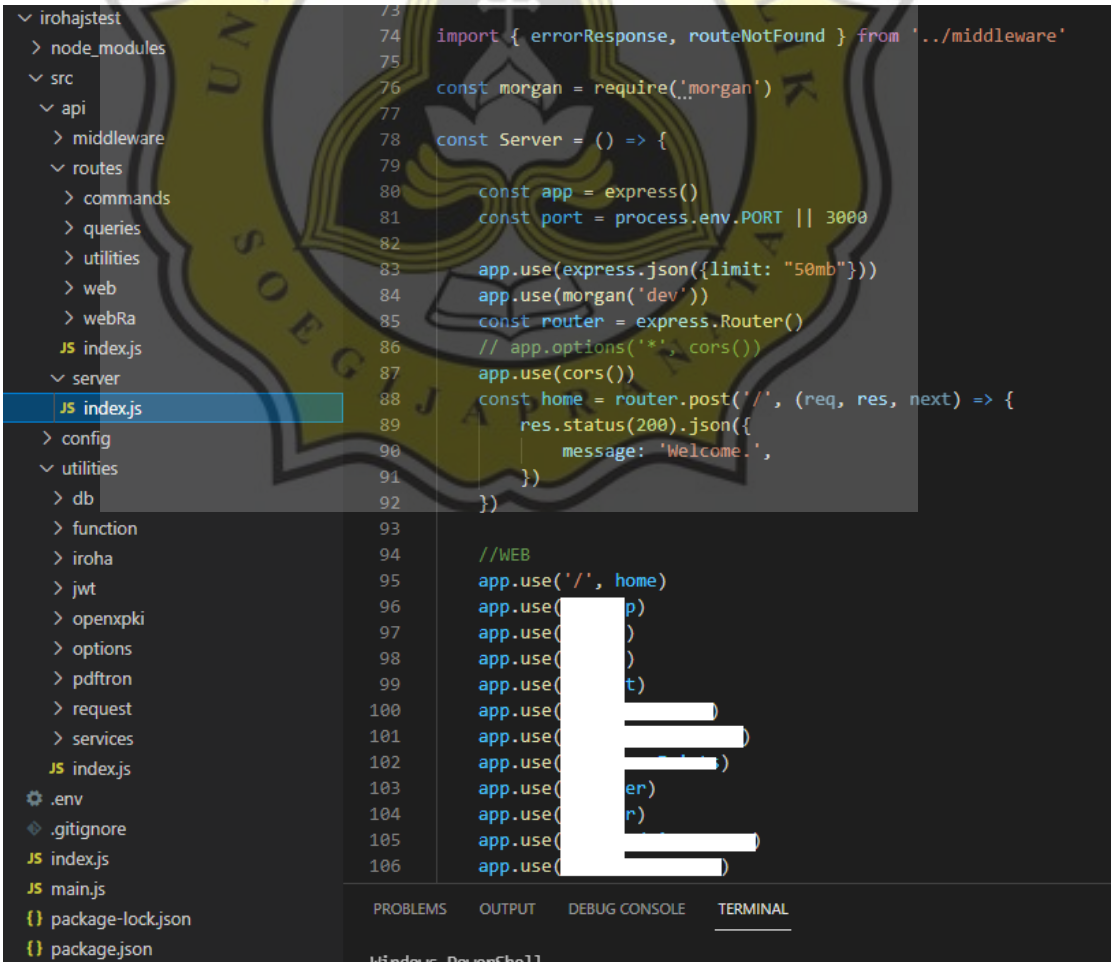
20. xhr2

    Dependency dari package @pdftron/pdfnet-node.

Untuk menjalankan server Node Js menggunakan bantuan *library* PM2 yang memberikan keunggulan dalam proses *logging* serta memastikan aplikasi agar dapat

selalu berjalan walaupun setelah komputer dimatikan atau tampilan *terminal* ditutup. Dan masih terdapat fitur lainnya yang dapat digunakan apabila dibutuhkan. Cara pemasangannya adalah dengan mengetikan “npm install -g pm2” untuk melakukan pemasangan *library* PM2 secara *global*. Kemudian apabila hendak menjalankan aplikasi Node.Js dapat dilakukan dengan cukup mengetikkan “pm2 start index.js --name namaaplikasi” di mana “index.js” adalah *file* yang hendak dijalankan dan “namaaplikasi” adalah nama proses PM2 yang ditentukan sendiri. Dan apabila hendak menjalankannya saat *development* dan menghendaki untuk langsung melihat perubahan dari code yang telah dibuat dapat dilakukan dengan cara mengetikan “pm2 start index.js --name namaaplikasi --watch --attach”. Untuk mengetahui cara membuat *startup script* pada perangkat komputer yang digunakan sesuai dengan sistem operasinya dapat dilihat secara detail pada dokumentasi PM2 di “https://pm2.keymetrics.io/docs/usage/startup/”.

Berikut pada gambar B.4 adalah contoh struktur folder projek aplikasi *back-end* dan file “server.js” yang menangani seluruh request yang masuk ke aplikasi *back-end*.



Gambar B.4: Contoh struktur folder project aplikasi backend.

Untuk membuat sistem REST API sederhana pastikan *library* express, body-parser, dan cors telah terinstal. Berikut adalah kode contoh route "getUser" yang juga memanfaatkan middleware authentication. Middleware authentication dibuat agar apabila diperlukan proses autentikasi saat mengakses route tertentu (protected route).

```
import express from 'express'
import mongoose from 'mongoose'
import { Db } from '../../../utilities/db/'
import { UserModel } from '../../../utilities/db/models/'
import { authentication } from '../../middleware/'

const router = express.Router()

const getUser = router.post('/getuser', authentication, async
(req, res) => {

    const username = req.body.username
    const id = req.body.userid

    let newToken = req.newToken

    if (!username && !id)
        return res.status(400).json({
            message: "Bad request.",
            info: {}
        })

    if (!newToken)
        newToken = ""

    const getData = async () => {
        if (username)
            return await UserModel.findOne({ username }).select([
                "-password",
                "-keypair",
                "-csr_pkcs10",
                "-cert_chain",
                "-refreshToken",
            ])

        if (id) {
```

```
            const idObj = mongoose.Types.ObjectId(id)
            return await UserModel.findOne({ _id: idObj
}).select([
                "-password",
                "-keypair",
                "-csr_pkcs10",
                "-cert_chain",
                "-refreshToken",
                "-idCard"
            ])
        }
    }

    Promise.all([
        await Db(),
        await getData()
    ])
        .then(a => {
            res.status(200).json({
                message: "Successfully getting user data.",
                info: a[1],
                newToken
            })
        })
        .catch(err => {
            res.status(500).json({
                message: "An error occured.",
                info: err.toString()
            })
        })
})

export default getUser
```

Kode B.1 Contoh kode pada protected route.

Proses autentikasi dilakukan dengan menggunakan middleware “authentication” yang dapat melakukan autentikasi terhadap token JWT yang dikirmkan oleh pengguna.

```
import jwt from 'jsonwebtoken'
import { JWT_SECRET, JWT_SECRET_2 } from '../../../config/'
import { generateTokenWRefreshToken } from
```

```
'../../../utilities/jwt/'

const authentication = (req, res, next) => {

    const throwExpired = () => {
        return res.status(403).json({
            message: "Your session is expired. Please login
again.",
            info: []
        })
    }

    const authHeader = req.headers['authorization']
    if (authHeader === "" || authHeader === undefined ||
authHeader === null)
        return res.status(400).json({
            message: "Unauthorized.",
            info: {}
        })

    const token = authHeader.split(" ")[1]
    jwt.verify(token, JWT_SECRET, async (err, tokenDecoded) => {
        if (err) {
            const refreshToken = req.body.refreshToken
            if (!refreshToken) return throwExpired()
            let id = ""
            jwt.verify(refreshToken, JWT_SECRET_2, (err,
tokenDecoded) => {
                if(err) return null
                id = tokenDecoded.id
            })
            if(!id) return throwExpired()
            const newToken = await generateTokenWRefreshToken(id,
refreshToken)
            if (newToken === "400") {
                return res.status(400).json({
                    message: "Session ended",
                    info: {}
                })
            }
            if (newToken === "403") {
                return res.status(403).json({
```

```
                    message: "Unauthorized",
                    info: {}
                })
            }
            req.newToken = newToken
            next()
        } else {
            next()
        }
    })
}

export default authentication
```

Kode B.2 Middleware "authentication".

Berikut kode B.3 adalah contoh file yang membungkus seluruh proses dan API service pada backend yang dibangun menggunakan "express".

```
import http from 'http'
import express from 'express'
import cors from 'cors'
import {
    //Web
    signIn,
    signUp,
    signOut,
    getUser
} from '../routes/'
import { errorResponse, routeNotFound } from '../middleware'


const morgan = require('morgan')

const Server = () => {
    const app = express()
    const port = process.env.PORT || 3000

    app.use(express.json({limit: "50mb"}))
    app.use(morgan('dev'))
    const router = express.Router()
    app.use(cors())
    const home = router.post('/', (req, res, next) => {
```

```
        res.status(200).json({
            message: 'Welcome.',
        })
    })

    //WEB
    app.use('/', home)
    app.use(signIn)
    app.use(signUp)
    app.use(signOut)
    app.use(getUser)

    app.use(routeNotFound)

    app.use(errorResponse)

    const server = http.createServer(app)
    server.listen(port)
}

export default Server
```

Kode B.3 REST API menggunakan express.

Untuk mempermudah dalam menangkap dan memberikan respon apabila terdapat respon yang tidak diinginkan (di atas 299) dapat juga diatasi dengan membuat *middleware* yang dapat menangkap *error* melalui fungsi "next" pada express. Contohnya dapat dilihat pada kode B.4.

```
const errorResponse = (err, req, res, next) => {

    let error = err

    console.error(error)

    if(!(err instanceof Error)) {
        error = new Error(error.message)
        error.status = err.status || 500
    }

    const status = error.status || 500
    const message = error.message.toString()
```

```
    return res.status(status).json({
        status,
        message: message || "An error occured."
    })
}

const badRequest = (message) => {
    return {
        status: 400,
        message: message || "Bad request."
    }
}

const unauthorized = (message) => {
    return {
        status: 401,
        message: message || "Unauthorized."
    }
}

const paymentRequired = (message) => {
    return {
        status: 402,
        message: message || "Insufficient funds."
    }
}

const forbidden = (message) => {
    return {
        status: 403,
        message: message || "Forbidden."
    }
}

const notFound = (message) => {
    return {
        status: 404,
        message: message || "Not found."
    }
}
```

```
const timeout = () => {
    return {
        status: 408,
        message: "Request timeout. Is destination URL online?"
    }
}

const conflicted = (message) => {
    return {
        status: 409,
        message: message || "Already exists."
    }
}

const fatalError = (error) => {
    return {
        status: 500,
        message: error.toString() || "Fatal error."
    }
}
```

Kode B.4 Middleware error handler.

B.2 MongoDb

Untuk menginstall MongoDb dapat dilihat pada link berikut ini https://docs.mongodb.com/manual/administration/install-community/ dengan cara-caranya disesuaikan terhadap sistem operasi yang digunakan. Pada sistem operasi Linux terdapat beberapa pilihan untuk menginstal MongoDb yaitu melalui *installer* (.deb), melalui *shell script* (.sh) yang dikemas dalam *tarball archive* (.tgz), atau manual melalui *apt package manager*.

Apabila menggunakan versi tarball Ubuntu terdapat *folder* dengan nama “bin” yang di dalamnya terdapat *script* untuk menginstal MongoDb Compass di mana itu adalah *tools* yang akan digunakan untuk menampilkan dan mengelola *database* MongoDb dengan bantuan *Graphical User Interface* (GUI) agar lebih mudah. *File* tersebut bernama “install_compass” dan dapat dijalankan apabila telah menginstall python pada sistem Linux yang digunakan. Langkahnya dapat dilihat pada kode B.5 dan B.6. Cara alternatif adalah dengan menginstall Compass melalui halaman

download pada website resmi MongoDb di https://www.mongodb.com/try/download/compass.

```
sudo mkdir -p /var/lib/mongo
sudo mkdir -p /var/log/mongodb
sudo cp <direktori di mana folder instalasi mongodb
berada>/bin/* /usr/local/bin/
sudo chown 'whoami' /var/lib/mongo
sudo chown 'whoami' /var/log/mongodb

# untuk menjalankan mongodb saat pertama kali
mongod --dbpath /var/lib/mongo --logpath
/var/log/mongodb/mongod.log --fork

# masuk ke CLI MongoDb
mongo
```

Kode B.5 Melakukan instalasi MongoDb pada sistem Ubuntu menggunakan package tarball (tgz).

```
sudo install_compass
```

Kode B.6 Menginstall MongoDb Compass (GUI MongoDb).

Apabila hendak mengatur hak akses pada MongoDb, ada beberapa langkah yang harus dilakukan. Pastikan *process* "mongod" telah berjalan dengan menjalankan perintah terakhir di atas setelah melakukan penginstalan. Lalu yang harus dilakukan adalah membuat *root user* baru dalam *database* "admin". Langkahnya dapat dilihat pada kode B.7.

```
#masuk ke mongodb cli
mongo

#gunakan database admin
use admin

#buat user dengan role sebagai root
db.createUser({
user: "namauser",
pwd: "password"
roles: [ "root" ]
})

#keluar dari cli
exit
```

Kode B.7 Command untuk membuat root user pada MongoDb.

Kemudian, apabila semua proses berjalan dengan lancar, saatnya melakukan *restart* terhadap sistem operasi yang sedang digunakan. Setelah melakukan *restart*, saatnya menjalankan MongoDb dengan pengaturan *port* yang akan digunakan, dan juga mengaktifkan autentikasi. Caranya dapat dilihat pada kode B.8.

```
mongod --auth --port 27017 --dbpath /var/lib/mongo --
logpath /var/log/mongodb/mongod.log --fork
```

Kode B.8 Mengaktifkan MongoDb dengan fitur autentikasi pada port yang dipilih.

Apabila berhasil, maka saat masuk ke MongoDb CLI tanpa melakukan autentikasi, seharusnya MongoDb CLI tidak akan menampilkan apa-apa kecuali apabila sebelumnya pengguna melakukan autentikasi terlebih dahulu. Ada beberapa cara untuk melakukan autentikasi melalui CLI yaitu saat hendak membuka CLI seperti pada kode B.9, atau saat sudah masuk ke dalam CLI MongoDb seperti pada kode B.10.

```
mongo --port 27017 --authenticationDatabase "admin" -u
"namauser" -p "password"
```

Kode B.9 Masuk ke MongoDb CLI sekaligus melakukan autentikasi pengguna.

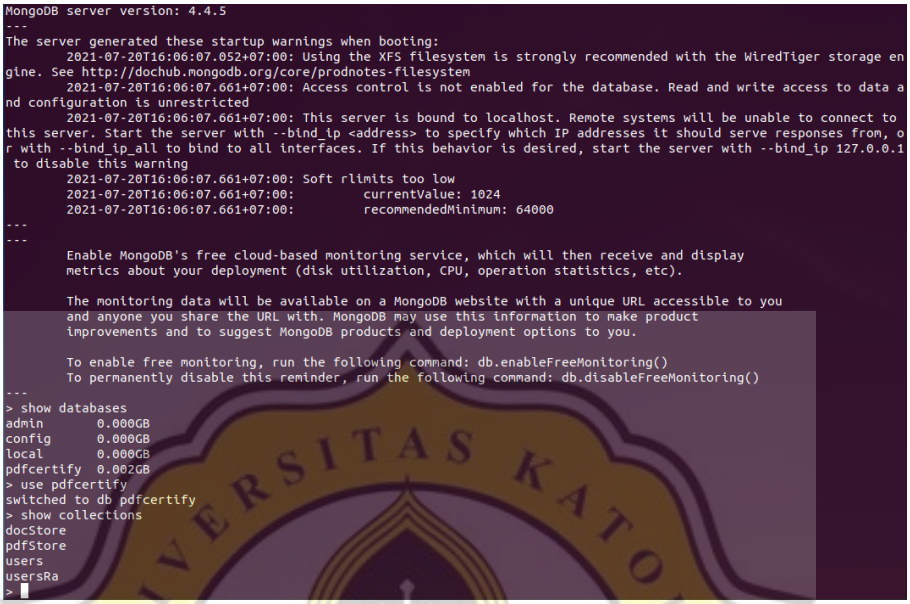
```
# masuk ke MongoDb CLI
mongo

# gunakan database admin
use admin

# fungsi untuk melakukan autentikasi
db.auth("namauser", "password")
```

Kode B.10 Command pada terminal untuk melakukan autentikasi di dalam MongoDb CLI.

Apabila hendak menambahkan *user* baru, dapat juga menggunakan cara yang sama dengan melihat *role* apa yang hendak ditambahkan ke *user* yang hendak dibuat. *Role* tersebut dapat dilihat pada dokumentasi MongoDb di link berikut https://docs.mongodb.com/manual/reference/built-in-roles/ dan selain itu dimungkinkan juga untuk menerapkan *role* yang berbeda terhadap masing-masing *database*. Dan perlu diingat bahwa apabila hendak menambahkan pengguna baru tidak harus menambahkan pengguna tersebut pada *database* admin, melainkan dapat juga dilakukan pada *database* lain. Hanya saja demi kemudahan dalam memanajemen pengguna, mungkin pilihan untuk menggunakan *database* admin sebagai *authentication database* adalah pilihan yang lebih baik seperti bahasan yang dijelaskan

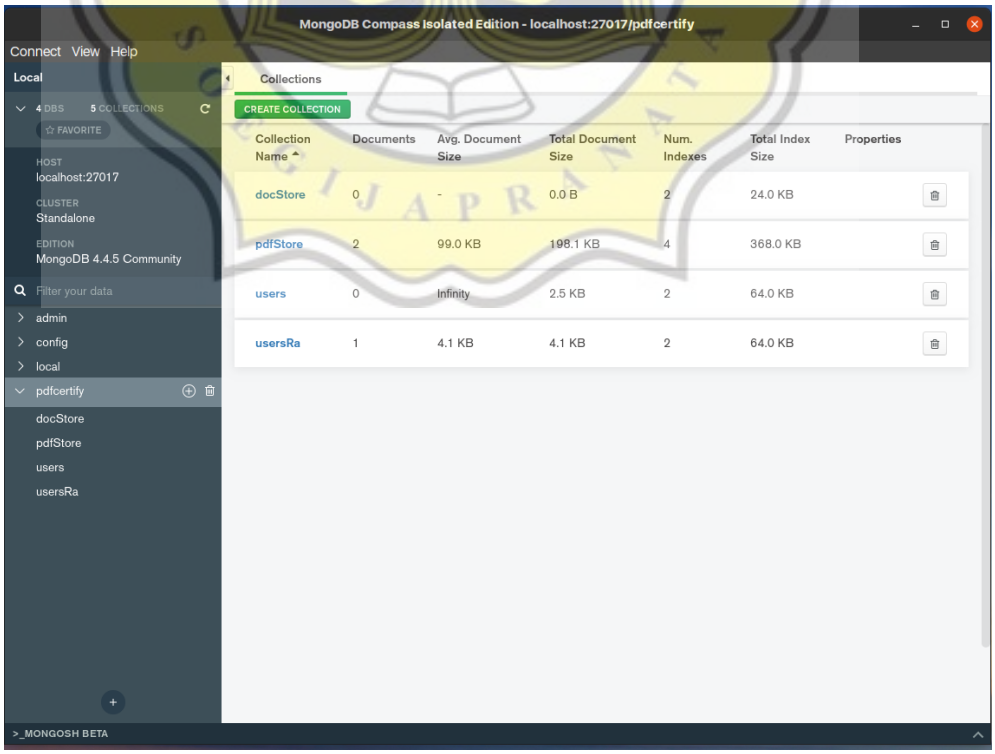
dalam artikel pada link berikut ini "https://medium.com/idomongodb/mongodb-which-authentication-database-should-i-use-when-creating-users-56876565dfa2".

Berikut pada gambar B.5 adalah contoh mengakses MongoDb melalui CLI MongoDb yang diakses menggunakan *terminal*.

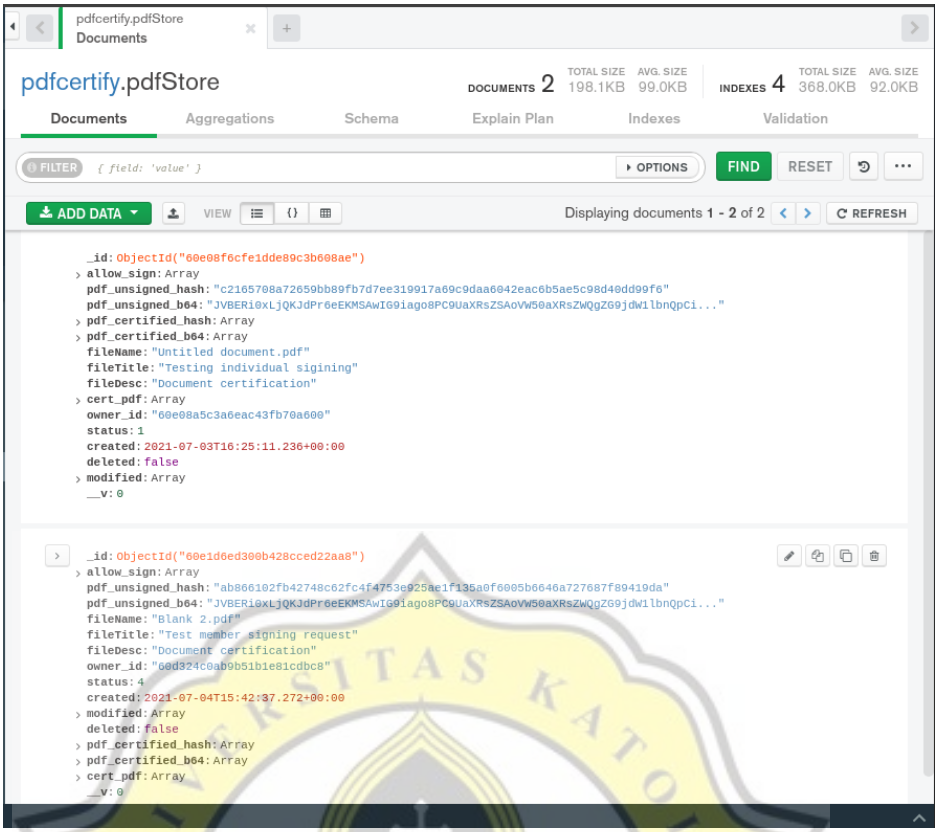


Gambar B.5: Tampilan mengakses database melalui MongoDb shell

Dan berikut pada gambar B.6 dan gambar B.7 adalah contoh mengakses MongoDb melalui MongoDb Compass.



Gambar B.6: Mengakses database melalui MongoDb Compass



Gambar B.7: Contoh tampilan data pada collections

Untuk dapat mengakses *data* pada MongoDb melalui Node.Js, *library* "mongoose" dapat dimanfaatkan dalam melakukan pengontrolan dan komunikasi dengan MongoDb. *Library* tersebut dapat diinstall menggunakan *package manager* baik itu npm atau yarn.

Umumnya yang harus dilakukan pertama kali adalah membuat skema dari *collection* (mirip seperti tabel apabila pada *database* berbasis SQL) yang hendak dibuat apabila hendak menentukan struktur data pada tabel beserta aturan pengisian datanya. Struktur ini dapat diubah apabila diperlukan tanpa menghapus data sebelumnya. Berikut pada kode B.11 adalah contoh membuat skema *collection* "users".

```
import mongoose from 'mongoose'

const UserSchema = new mongoose.Schema({
     username: { type: String, required: true, unique: true },
   password: { type: String, required: true },
   fullName: { type: String, required: true },
   refreshToken: [{ type: String }],
   created: { type: Date, required: true },
   modified: [{
```

```
        time: { type: Date, required: true },
        modifier_id: { type: String, required: true }
    }],
    deleted: { type: Boolean, required: true }
}, { collection: "users" })

const UserModel = mongoose.model("UserSchema", UserRaSchema)

export default UserModel
```

Kode B.11 Contoh membuat model atau schema collection "users"

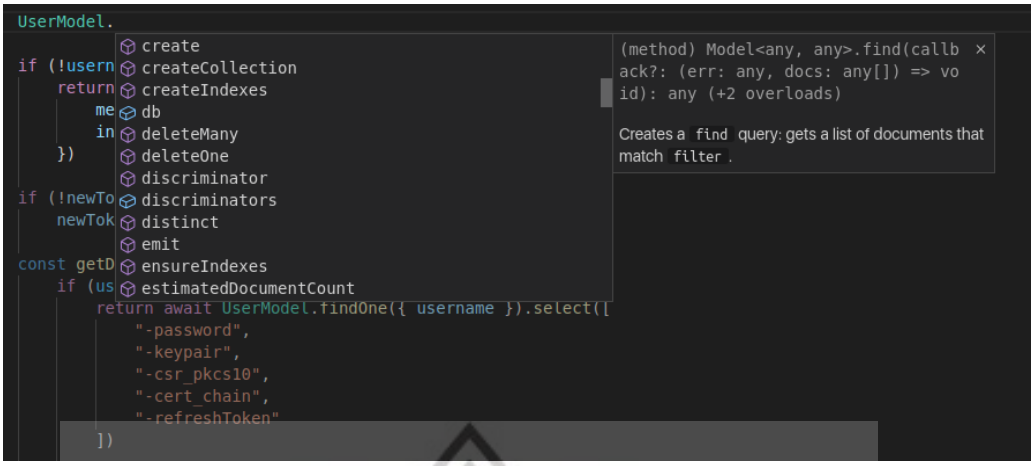
Sebelum dapat mengakses MongoDb, koneksi ke MongoDb harus dibuat terlebih dahulu dengan cara seperti pada kode B.12. Variabel "DB_NAME" adalah *database* yang akan digunakan, dan juga "DB_URL" adalah *connection string* yang akan digunakan untuk melakukan koneksi ke MongoDb sekaligus melakukan autentikasi dengan menyertakan *username*, *password*, dan *database* autentikasi (contoh: mongodb://namauser:password@localhost:27017/?authSource=admin).

```
import mongoose from 'mongoose'
import { DB_URL, DB_NAME } from '../../config/'

const options = {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    useCreateIndex: true,
    dbName: DB_NAME
}

const Db = async () => { return await mongoose.connect(DB_URL,
options) }

export { Db }
```

Kode B.12 Membuat koneksi ke MongoDb menggunakan package "mongoose".

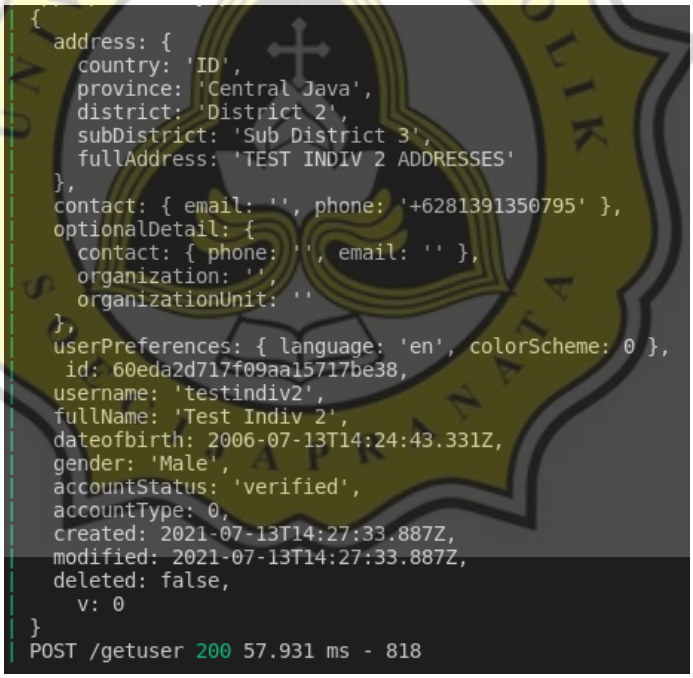
Kemudian apabila langkah di atas telah dilakukan, menjadi dimungkinkan untuk dapat melakukan operasi berdasarkan *collection* yang dipilih baik itu apabila hendak membuat *data* baru, merubah *data*, atau melakukan *update* pada salah satu *data* tertentu. Contohnya seperti pada gambar B.8 dapat dilihat bahwa opsi dan suggestion

pada Visual Studio Code telah muncul apabila mengetikan skema/model yang telah dibuat. Suggestion hanya muncul apabila *library* "mongoose" telah terinstall.



Gambar B.8: Contoh operasi yang dapat dilakukan dengan model yang telah dibuat.

Dan berikut pada gambar B.9 adalah contoh hasil mendapatkan data menggunakan skema *collection* yang telah dibuat sebelumnya.



Gambar B.9: Mengambil data pada model yang telah dibuat menggunakan package "mongoose".

B.3 Hyperledger Iroha

Perlu menjadi catatan bahwa pada saat pembuatan *project*, sistem operasi yang digunakan adalah Ubuntu 20.04 LTS atau sistem operasi berbasis Linux. Apabila hendak menggunakan sistem operasi Windows, sekarang dapat menggunakan WSL (Windows Subsystem for Linux) dan Ubuntu for Windows untuk dapat menjalankan

beberapa perintah yang hanya dapat dilakukan pada sistem Linux termasuk apabila hendak menjalankan *shell script* atau sh tanpa harus melakukan penginstalan sistem operasi terpisah pada perangkat berbeda atau dengan perangkat sama menggunakan *dual boot*. Pastikan juga Docker telah terinstal pada sistem agar dapat menjalankan *container* Hyperledger Iroha di dalam sistem yang sedang digunakan. Apabila menggunakan Windows, setelah membandingkan disarankan untuk menggunakan Docker versi Ubuntu yang diinstall menggunakan WSL Ubuntu. Karena pada versi Ubuntu, *container* dapat diakses dengan mudah melalui *host* (Windows). Sedangkan apabila menggunakan Docker for Windows, terdapat langkah-langkah lagi yang harus dilakukan agar dapat mengakses *container* melalui sistem *host* (Windows) yang digunakan.

Untuk mengatur versi yang hendak dipakai (seperti pada contoh pada gambar B.10), versi dapat diset pada *file shell script* "./entrypoint-build.sh", "./bin/genesis.sh" dan juga "./bin/build.sh".

```
TAG=${TAG:-1.2.1}
VERSION=${VERSION:-1.2.1}
```

Gambar B.10: Versi image Hyperledger Iroha yang hendak digunakan.

Genesis block yang nantinya akan digunakan dapat dikonfigurasikan dengan mengedit *file* pada *path* "./data-genesis/genesis.block" seperti pada gambar B.11 dan gambar B.12. Umumnya yang dilakukan adalah pembuatan *private* dan *public key* untuk *peer* Hyperledger Iroha, kemudian membuat juga *user* dan *domain* yang dapat digunakan beserta *role*-nya juga, ataupun *asset* yang akan digunakan serta berapa saldonya.

```
{
   "addPeer":{
      "peer":{
         "address":"n3.pdfcertify.local:10001",
         "peerKey":"$N3"
      }
   }
},
{
   "addPeer":{
      "peer":{
         "address":"n4.pdfcertify.local:10001",
         "peerKey":"$N4"
      }
   }
},
{
   "addPeer":{
      "peer":{
         "address":"n5.pdfcertify.local:10001",
         "peerKey":"$N5"
      }
   }
},
{
   "addPeer":{
      "peer":{
         "address":"n6.pdfcertify.local:10001",
         "peerKey":"$N6"
      }
   }
},
{
   "addPeer":{
      "peer":{
         "address":"n7.pdfcertify.local:10001",
         "peerKey":"$N7"
      }
   }
},
{
   "createRole":{
      "roleName":"admin",
      "permissions":[
         "root"
      ]
   }
```

Gambar B.11: Commands yang dituliskan pada file "genesis.block".

Gambar B.12: Lanjutan commands yang dituliskan pada file "./genesis.block".

Pada *file* yang berada pada *path* "./entrypoint-genesis.sh" terdapat *script* yang dilakukan untuk membuat akun beserta *private* dan *public* keynya pada *container* "iroha-genesis" seperti pada gambar B.13. *Container* ini dibuat hanya dengan tujuan untuk membuat *private* dan *public key* dari *akun* yang telah disebutkan pada *file* "./genesis.block". Dan variabel yang telah dikonfigurasikan pada *genesis block* dapat diisikan valuenya untuk kali ini konteksnya adalah value dari *public key*. Kemudian *script* ini akan membuat *file genesis block* yang baru yang kemudian akan digunakan saat pembuatan *node* (*peer*) Hyperledger Iroha yang sesungguhnya.

```
iroha-cli -new_account -account_name admin@pdfcertify.local
iroha-cli -new_account -account_name n1
iroha-cli -new_account -account_name n2
iroha-cli -new_account -account_name n3
iroha-cli -new_account -account_name n4
iroha-cli -new_account -account_name n5
iroha-cli -new_account -account_name n6
iroha-cli -new_account -account_name n7
iroha-cli -new_account -account_name genesis-node

ADMIN=$(<admin@pdfcertify.local.pub)
N1=$(<n1.pub)
N2=$(<n2.pub)
N3=$(<n3.pub)
N4=$(<n4.pub)
N5=$(<n5.pub)
N6=$(<n6.pub)
N7=$(<n7.pub)

# replace the key values in the genesis.block setup
sed -i 's!\$ADMIN!'"${ADMIN}"'!g' genesis.block
sed -i 's!\$N1!'"${N1}"'!g' genesis.block
sed -i 's!\$N2!'"${N2}"'!g' genesis.block
sed -i 's!\$N3!'"${N3}"'!g' genesis.block
sed -i 's!\$N4!'"${N4}"'!g' genesis.block
sed -i 's!\$N5!'"${N5}"'!g' genesis.block
sed -i 's!\$N6!'"${N6}"'!g' genesis.block
sed -i 's!\$N7!'"${N7}"'!g' genesis.block

# create the genesis block
irohad \
  --drop_state \
  --overwrite_ledger \
  --genesis_block genesis.block \
  --config config-genesis.json \
  --keypair_name genesis-node
```

Gambar B.13: Script "entrypoint-genesis.sh" dalam pembuatan keypair beserta genesis block.

Pada "./bin/genesis.sh" (gambar B.14) terdapat aktivitas yang dilakukan yang paling utama untuk diperhatikan pada saat terdapat perintah untuk melakukan penyalinan masing-masing *private key* dan *public key* yang telah dibuat melalui *container* "iroha_genesis" ke direktori "./data" pada project.

```
# copy keys
docker cp iroha_genesis:/opt/iroha/data/admin@pdfcertify.local.pub data/
docker cp iroha_genesis:/opt/iroha/data/admin@pdfcertify.local.priv data/
docker cp iroha_genesis:/opt/iroha/data/n1.pub data/
docker cp iroha_genesis:/opt/iroha/data/n1.priv data/
docker cp iroha_genesis:/opt/iroha/data/n2.pub data/
docker cp iroha_genesis:/opt/iroha/data/n2.priv data/
docker cp iroha_genesis:/opt/iroha/data/n3.pub data/
docker cp iroha_genesis:/opt/iroha/data/n3.priv data/
docker cp iroha_genesis:/opt/iroha/data/n4.pub data/
docker cp iroha_genesis:/opt/iroha/data/n4.priv data/
docker cp iroha_genesis:/opt/iroha/data/n5.pub data/
docker cp iroha_genesis:/opt/iroha/data/n5.priv data/
docker cp iroha_genesis:/opt/iroha/data/n6.pub data/
docker cp iroha_genesis:/opt/iroha/data/n6.priv data/
docker cp iroha_genesis:/opt/iroha/data/n7.pub data/
docker cp iroha_genesis:/opt/iroha/data/n7.priv data/
```

Gambar B.14: Pengkopian file key yang telah dibuat melalui container iroha_genesis.

Apabila persiapan untuk membuat *genesis block* telah selesai dilakukan, script "genesis.sh" dapat dijalankan dengan cara seperti pada kode B.13.

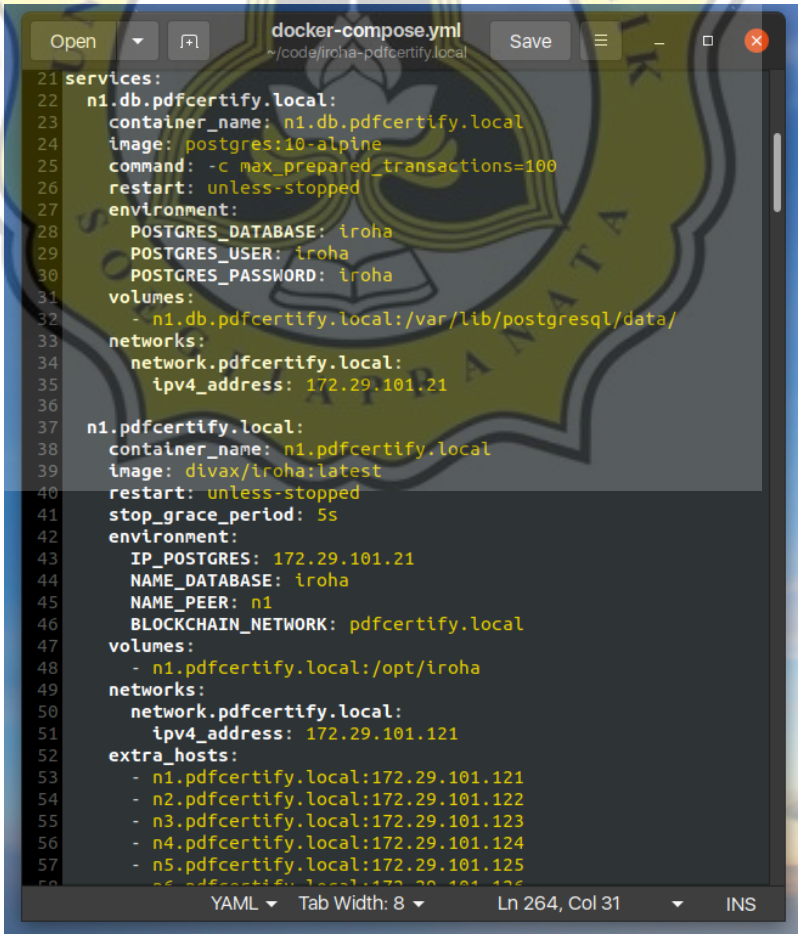
```
sudo sh genesis.sh
```

Kode B.13 Command untuk menjalankan script pembuatan genesis block serta pembuatan key.

File yang telah disalin melalui *script* "./bin/genesis.sh" dapat ditemukan dalam direktori "./data" pada folder project Hyperledger Iroha seperti pada gambar B.15.

Gambar B.15: Hasil dari key yang telah berhasil di copy ke directory local.

Kemudian saatnya membuat konfigurasi dari *container* Hyperledger Iroha yang akan dibuat dengan mengedit *file* "./docker-compose.yml" sesuai dengan konfigurasi yang telah ditentukan sebelumnya seperti pada gambar B.16.



Gambar B.16: Membuat container yang diperlukan melalui file "docker-compose.yml".

Apabila persiapan di atas telah dilakukan, saatnya menjalankan *script* pembuatan *container*, *network*, dan *volume* yang sebelumnya telah dikonfigurasikan pada *file* "docker-compose.yml" dengan bantuan *library* "docker-compose" dengan cara seperti pada kode B.14.

```
sudo docker-compose up -d
```

Kode B.14 Command untuk menjalankan container pada docker.

Apabila sudah, menjadi dimungkinkan untuk dapat masuk ke dalam CLI (bash) dari *container* yang telah dibuat tersebut dengan mengetikan perintah "docker exec -it n1.pdfcertify.local /bin/bash" di mana "n1.pdfcertify.local" adalah nama *container* yang hendak diakses (seperti pada kode B.15).

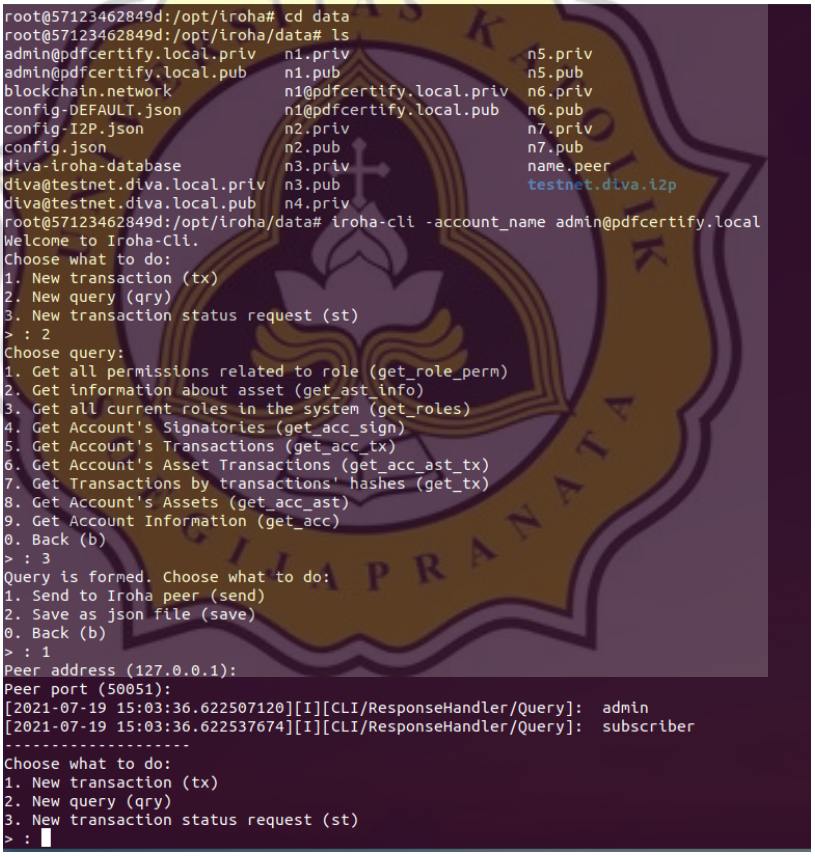
```
sudo docker exec -it n1.pdfcertify.local /bin/bash
```

Kode B.15 Command untuk masuk ke dalam direktori container peer Hyperledger Iroha.

Untuk dapat melihat *data* dari *block* yang ada pada *peer*, dapat dilakukan dengan masuk ke direktori "./blockstore" seperti pada gambar B.17. Dan apabila telah sukses menampilkan *genesis block* yang telah dibuat sebelumnya dan isinya sesuai dengan konfigurasi yang diberikan sebelumnya, maka langkah-langkah yang telah dilakukan telah berjalan dengan sukses.

Gambar B.17: Masuk ke directory salah satu peer dan melihat konten dari blockstore

Untuk dapat masuk ke “iroha-cli”, yang harus dilakukan adalah masuk ke direktori “./data” di mana pada directory tersebut terdapat *private key file* yang dapat digunakan untuk mengakses iroha-cli. Untuk dapat masuk ke iroha-cli dilakukan dengan cara mengetikan “iroha-cli -account_name admin@pdfcertify.local” di mana “admin@pdfcertify.local” adalah akun yang telah dibuat sebelumnya dan juga di mana “admin” adalah nama dari akun yang dibuat dan “pdfcertify.local” adalah nama dari *domain* yang telah ditentukan di mana akun tersebut akan berada. Setelah dapat masuk ke iroha-cli sebagai akun “admin” maka pengguna akun “admin” tersebut dapat melakukan aktivitas transaksi melalui “*command*” (“*New transaction*” apabila pada iroha-cli) dan juga mengambil atau meminta data melalui “*query*” (“*New Query*” pada iroha-cli). Contoh melakukan aktivitas menggunakan “iroha-cli” dapat dilihat pada gambar B.18.

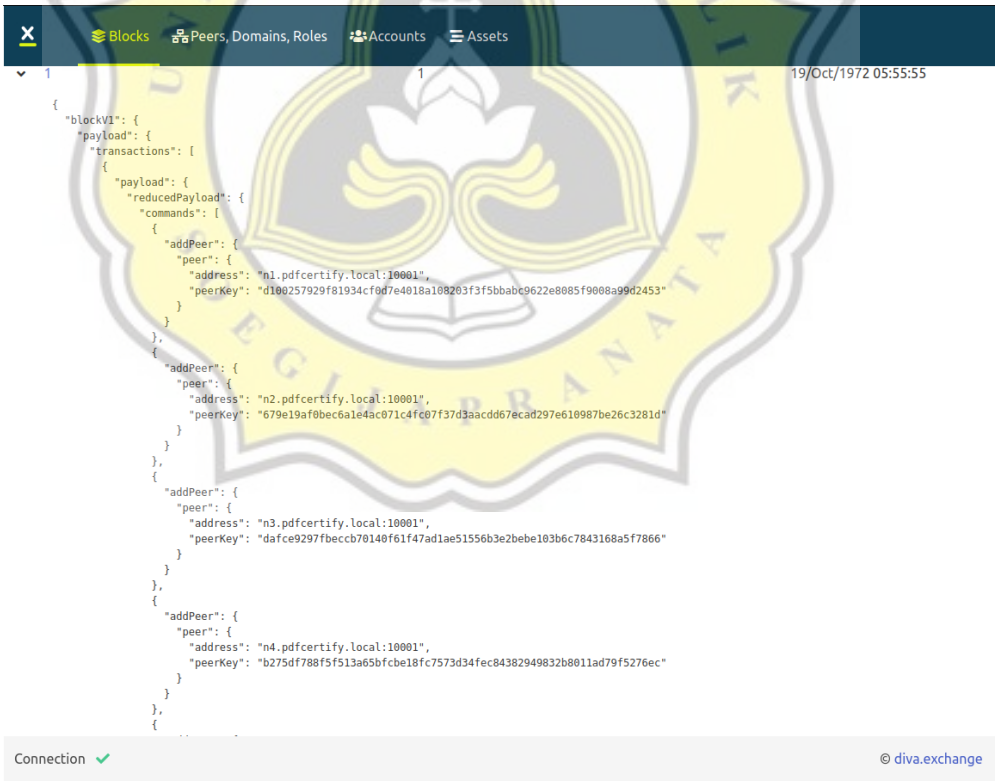


Gambar B.18: Masuk ke iroha-cli dan juga melakukan query menggunakan iroha-cli.

Apabila hendak menghapus *container peer* yang telah dibuat sebelumnya tanpa menghapus docker volume yang telah dibuat, dapat dilakukan dengan cara mengetikan “docker-compose down” pada direktori projek Hyperledger Iroha, atau apabila ingin menghapus seluruh *container* beserta volumenya secara benar-benar bersih dapat

dilakukan dengan cara mengetikan “docker-compose down --volumes”. Tidak perlu khawatir untuk mengulangi proses dari awal karena perintah tersebut tidak menghapus *docker image* yang telah diunduh sebelumnya untuk dapat melakukan instalasi atau pembuatan docker container kecuali apabila diberikan perintah untuk menghapus *docker image* tersebut.

Secara opsional untuk dapat mengakses “Iroha explorer” yang dikembangkan oleh “Diva.exchange” yang berjalan dalam bentuk *container* apabila perintah “docker-compose up” telah dilakukan, dapat dilakukan dengan mengakses alamat IP beserta *port website* tersebut yang telah ditentukan sebelumnya pada *file* konfigurasi docker-compose melalui *web browser*. Peer atau *blockstore* yang digunakan untuk menampilkan data *block* pada *web explorer* juga dapat diatur melalui konfigurasi pada *file* “./docker-compose.yml”. Secara *default* apabila belum melakukan perubahan apapun pada konfigurasi *docker compose file*, *website* tersebut dapat diakses melalui alamat “http://172.29.101.3:3920/”. Contoh tampilannya dapat dilihat pada gambar B.19.



Gambar B.19: Hyperledger Iroha Explorer yang dikembangkan oleh diva.exchange.

Kemudian karena pada *project* ini hendak menggunakan Node.Js untuk mengoperasikan Hyperledger Iroha, langkah pertama yang harus dilakukan adalah mengujinya dengan mengirimkan *request* berupa “*commands*” atau “*query*” ke *peer*

Hyperledger Iroha yang telah dibuat. Salah satu caranya adalah mencobanya dengan membuat *endpoint* pada Node.Js yang menerima *request* dari *client*.

Untuk berkomunikasi dengan *peer* pada Hyperledger Iroha diperlukan *package (library)* "grpc" karena sistem API yang telah disediakan oleh Hyperledger Iroha menggunakan grpc dan merupakan *dependency* dari package "iroha-helpers", kemudian juga diperlukan *library* "iroha-helpers" untuk dapat membuat *request* ke *peer* Hyperledger Iroha baik itu berupa *commands* maupun *query*. *Library* tersebut dapat diinstal menggunakan *package manager* seperti npm atau yarn pada project Node.Js yang telah disiapkan.

Untuk berkomunikasi dengan *peer* yang ada pada Hyperledger Iroha kita harus menyiapkan masing-masing *service* yang digunakan untuk melakukan komunikasi dengan *peer* Hyperledger Iroha melalui gRPC apabila hendak melakukan "*command request*" atau "*query request*". Secara umum masing-masing *service* tersebut biasa dinamai dengan "commandService" dan "queryService" seperti pada kode B.16. Secara umum kita perlu menyiapkan alamat dari *peer* Hyperledger Iroha (IP *address* dan *port* dari *peer* yang digunakan untuk menerima *request* dari gRPC.) yang hendak dihubungi, dan juga membuat *credential* gRPC. Apabila pengembangan program sedang dalam *development environment* dan tidak menggunakan *server* yang mempunyai SSL *digital certificate*, pembuatan *credential* dapat sementara dilakukan dengan fungsi "createInsecure". Atau apabila hendak menggunakan SSL *digital certificate*, dapat dilakukan dengan menggunakan fungsi "createSsl".

```
import grpc from 'grpc'
import {
   QueryService_v1Client,
   CommandService_v1Client
} from 'iroha-helpers/lib/proto/endpoint_grpc_pb'
import { IROHA_ADDRESS } from '../../config/'

const commandService = new CommandService_v1Client(
   IROHA_ADDRESS,
   grpc.credentials.createInsecure()
)

const queryService = new QueryService_v1Client(
   IROHA_ADDRESS,
   grpc.credentials.createInsecure()
```

```
)

export {
    commandService,
    queryService
}
```

Kode B.16 Command service dan query service.

Lalu untuk membuat sebuah transaksi dapat dilakukan dengan cara mengimport objek “commands” atau “queries” yang ada pada *library* “iroha-helpers”. Kemudian apabila hendak melakukan satu transaksi atau *request* saja, dapat dilakukan seperti pada contoh kode B.17.

```
return await commands.createAccount(commandOptions, {
    accountName,
    domainId,
    publicKey
})
```

Kode B.17 Melakukan pembuatan akun baru pada sistem blockchain Hyperledger Iroha.

Pada setiap kali mengirimkan perintah, tidak lupa juga harus menyertakan *private key* dari akun yang digunakan untuk melakukan transaksi sebagai *credential* atau identitas. Tidak lupa juga menyertakan *private key* pada objek “options” yang dimasukan pada *parameter* pertama saat hendak melakukan transaksi baik itu “*commands*” maupun “*queries*” (seperti pada kode B.18 dan kode B.19).

```
const commandOptions = {
 privateKeys: [PRIVATE_KEY], // Array of private keys in hex
format
 creatorAccountId: ACCOUNT_ID, // account@domain
 quorum: 1,
 commandService,
 timeoutLimit: 5000
}

const queryOptions = {
 privateKey: PRIVATE_KEY, // Private key in hex format
 creatorAccountId: ACCOUNT_ID, // account@domain
 queryService,
 timeoutLimit: 5000
```

```
}
```

Kode B.18 Command options dan query options.

```
return await queries.getAccount(queryOptions, { accountId })
        .then(info => {
            return {
                status: 200,
                message: "Success getting account info of '" +
accountId + "'.",
                info,
                data: JSON.parse(info.jsonData)
            }
        })
        .catch(e => { return fatalError(e) })
}
```

Kode B.19 Contoh melakukan query dari sebuah akun.

Pendekatan lain apabila hendak melakukan transaksi *commands* dapat dilakukan dengan cara dengan pendekatan "objek transaksi", yaitu dengan melihat transaksi sebagai objek yang dapat diolah kembali untuk mendapatkan sesuatu yang diinginkan seperti apabila hendak mendapatkan *hash* dari transaksi tersebut, atau untuk mendapatkan informasi seperti waktu transaksi, *payload* dari transaksi, dan sebagainya. Contohnya dapat dilihat seperti pada kode B.20.

```
import * as chain from 'iroha-helpers/lib/chain'

const TxBuilder = chain.TxBuilder

const createAccount = async ({ accountName, domainId, publicKey
}) => {

   if (!accountName || !domainId || !publicKey) return
badRequest()

   const ADMIN_ID = IROHA_ADMIN + "@" + IROHA_DOMAIN
   const accountId = accountName + "@" + domainId

   const tx = new TxBuilder()
       .createAccount({
           accountName,
```

```
            domainId,
            publicKey
        })
        .addMeta(ADMIN_ID, 1)
        .sign([PRIVATE_KEY])

    const txHash = hashTx(tx.tx)

    return await tx.send(commandService, 5000)
        .then(() => {
            return {
                status: 200,
                message: "Account '" + accountId + "' created
successfully.",
                info: txHash
            }
        })
        .catch(e => { return fatalError(e) })
}
```

Kode B.20 Contoh melakukan command "createAccount" dengan pendekatan "objek transaksi".

Pembuatan *hash* dari objek transaksi dapat dilakukan dengan mudah karena *library* "iroha-helpers" menyediakan fungsi *hash* yang dapat digunakan untuk membuat *hash* dari objek transaksi. Fungsi *hash* dapat dibuat seperti pada kode B.21.

```
import { txHelper } from 'iroha-helpers'

const hashTx = (tx) => {
    const hash = txHelper.hash
    return hash(tx).toString("hex")
}
```

Kode: B.21 Fungsi untuk membuat hash dari suatu transaksi Hyperledger Iroha.

Selain itu juga dimungkinkan untuk menjalankan perintah secara "batch" apabila hendak melakukan transaksi berupa *command* lebih dari satu transaksi dengan cara seperti contoh di bawah ini (kode B.22).

```
import * as chain from 'iroha-helpers/lib/chain'

const TxBuilder = chain.TxBuilder
const BatchBuilder = chain.BatchBuilder
```

```
const setKtp = new TxBuilder()
.setAccountDetail({
    accountId,
    key: "ktp",
    value: idChar
})
.addMeta(ACCOUNT_ID, 1)
.tx

const setFullName = new TxBuilder()
.setAccountDetail({
    accountId,
    key: "fullName",
    value: fullName
})
.addMeta(ACCOUNT_ID, 1)
.tx

const batch = async () => {

    return new BatchBuilder([
        setKtp,
        setFullName
    ])
    .setBatchMeta(0)
    .sign([PRIVATE_KEY], 0)
    .sign([PRIVATE_KEY], 1)
    .send(commandService)
    .then(info => {
        return res.status(200).json({
        message: "Account detail has been set successfully.",
        info
      })
    })
    .catch(next)
}

return await batch()
```

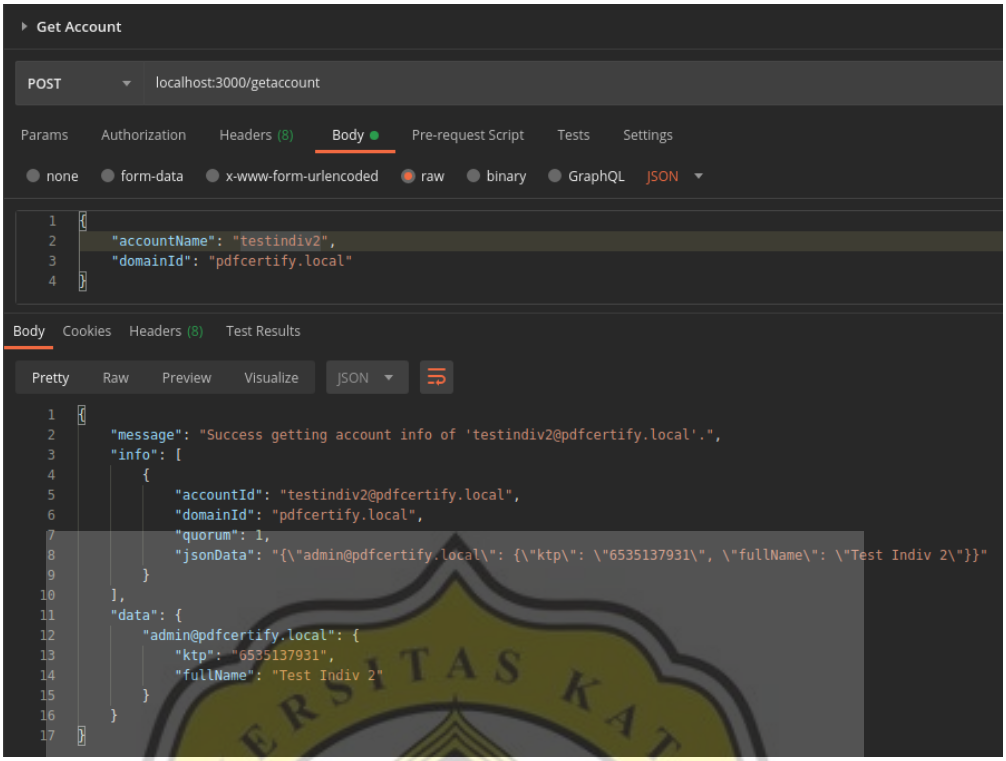
Kode B.22 Membuat transaksi batch.

Untuk melakukan transaksi *batch*, pertama-tama yang harus dilakukan adalah menyiapkan masing-masing transaksi yang hendak dilakukan menjadi "objek transaksi" yang dibuat menggunakan *helper* "TxBuilder". Kemudian objek transaksi yang telah siap dikirimkan dibungkus ke dalam sebuah array, dan kemudian menjadikan array berisi transaksi tersebut menjadi *parameter* pertama pada *helper* "BatchBuilder". Masing-masing transaksi harus ditandatangani dengan *private key* dari pelaku transaksi secara berurutan (dihitung dari 0 sebagai indeks pertama), barulah transaksi tersebut dapat dikirim menggunakan "commandService" yang sebelumnya telah dibuat. Dan untuk mendapatkan *hash* dapat dilakukan dengan cara melakukan mapping dari objek "txs" yang dapat ditemukan pada objek transaksi *batch* (transaksi *batch* yang dibuat menggunakan *helper* "BatchBuilder"), kemudian memanggil fungsi *hash* yang telah dibuat sebelumnya dengan menyertakan objek transaksi yang diberikan saat melakukan *array mapping* menjadi *parameter* pertama (seperti pada kode B.23).

```
const txList = batch.txs
const txHash = txList.map(item => { return hashTx(item) })
```

Kode B.23 Contoh mendapatkan hash dari masing-masing transaksi pada batch.

Maka apabila masing-masing transaksi telah sukses dilakukan, apabila dicoba mengakses *endpoint* yang telah dibuat tersebut dengan *tools* REST API testing seperti "Postman" seharusnya akan mendapat respon dengan kode status 200 dan mendapatkan hasil atau efek dari transaksi yang sebelumnya telah dilakukan apabila tidak terdapat *error* (dapat dilihat pada gambar B.20).



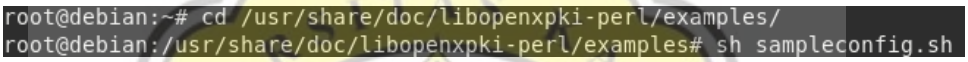
Gambar B.20: Hasil percobaan sukses memanggil endpoint "./getaccount".

Apabila semua langkah di atas dapat dilakukan dengan baik, pengembangan aplikasi lebih lanjut dapat dilakukan.

B.4 OpenxPKI

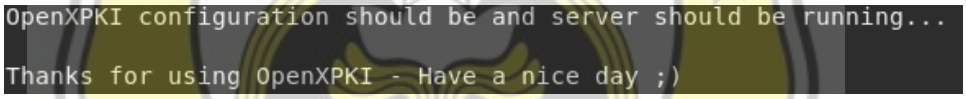
Untuk sistem yang digunakan dalam menjalankan *server* OpenXPKI disarankan untuk menggunakan Debian Buster. Terdapat beberapa cara yang dapat digunakan untuk menginstal Debian Buster selain melakukan instalasi pada perangkat lain atau *dual boot* apabila belum mempunyai sistem Debian Buster. Yang pertama adalah menggunakan software *Hypervisor* seperti Oracle Virtual Box, atau apabila menggunakan Windows, dapat juga memanfaatkan WSL (Windows Subsystem for Linux). OpenXPKI yang digunakan saat pembuatan *project* ini adalah versi 3.1.2. Dan demi kemudahan dalam melakukan penelitian serta pengembangan aplikasi, instalasi OpenXPKI dilakukan dengan menggunakan konfigurasi sampel yang telah disediakan oleh OpenXPKI. Cara penyiapan OpenXPKI pada sistem operasi Debian dapat dilihat dalam dokumentasi resmi OpenXPKI yaitu pada URL "https://openxpki.readthedocs.io/en/stable/quickstart.html" pada bagian "Debian Builds". Secara singkatnya yang harus dilakukan adalah mengunduh segala *requirement* yang dibutuhkan oleh OpenXPKI terlebih dahulu. Seperti yang dijelaskan

bahwa pada sistem yang akan digunakan harus dipastikan bahwa gnupg telah terinstal, dan "wget" telah terinstal juga agar dapat mengunduh *file* pada dokumentasi OpenXPKI. Kemudian setelah persiapan *requirement* telah dilakukan, saatnya menjalankan file *shell script* sample konfigurasi yang dapat ditemukan pada direktori "./usr/share/doc/libopenxpki-perl/examples/sampleconfig.sh" dalam sistem operasi Debian Buster dengan cara mengetikkan "sh sampleconfig.sh" pada *terminal* (seperti pada gambar B.21). Pada saat pertama kali mengunduh OpenxPKI *file* konfigurasi sampel biasanya masih terbungkus dalam sebuah *gunzip archive* yang mana harus di extract terlebih dahulu sebelum dapat menjalankan script tersebut. Apabila *script* konfigurasi sample tersebut telah berhasil dijalankan, maka proses penyiapan akan berjalan sampai *server* OpenXPKI dapat berjalan lengkap beserta dengan *web* nya juga dapat diakses melalui *web browser*.



Gambar B.21: Menjalankan script konfigurasi sampel pada terminal.

Berikut pada gambar B.22 adalah hasil pada *terminal* apabila proses konfigurasi OpenXPKI menggunakan file *sample* konfigurasi telah berhasil dilakukan.

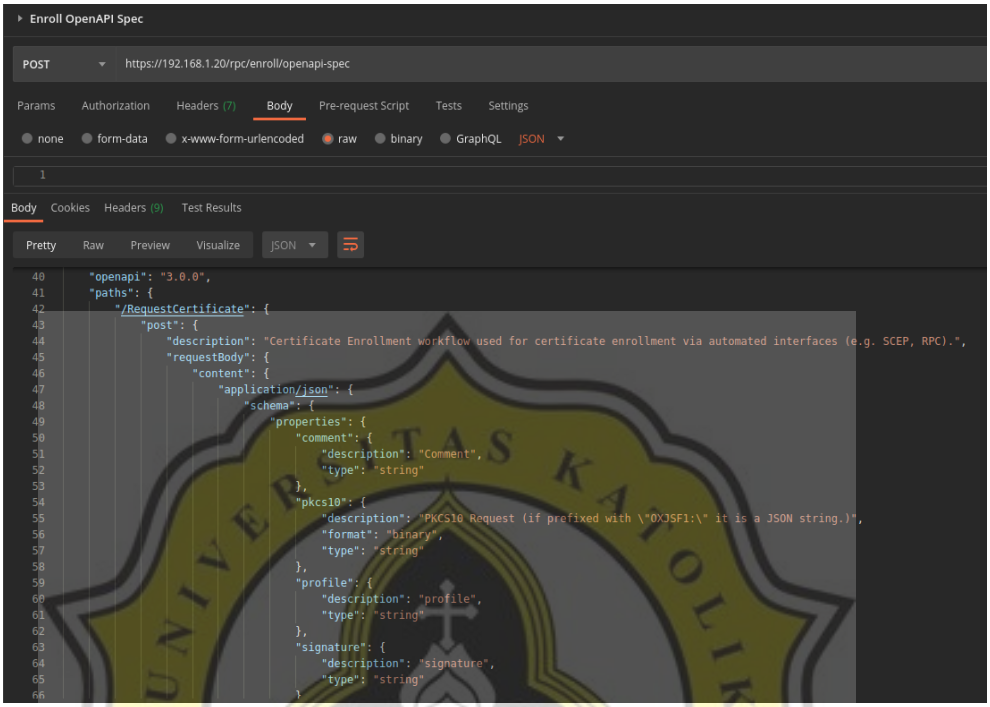


Gambar B.22: Tampilan log bila OpenXPKI sukses terinstall dengan script sampel konfigurasi.

Apabila hendak menjalankan OpenXPKI lagi setelah sistem berjalan kembali, pastikan bahwa *service* "mysql" dan "apache2" telah berjalan pada Debian Buster. Cara yang terbaik adalah memanfaatkan "systemd" apabila anda menggunakan Debian Buster secara sistem operasi terpisah. Tetapi apabila menggunakan *Windows Subsystem for Linux* (WSL), mungkin terdapat beberapa cara untuk dapat menjalankan perintah untuk menjalankan *service* saat *startup* di sistem operasi Windows seperti pada contoh yang dijelaskan pada artikel berikut ini "https://www.how2shout.com/linux/how-to-start-wsl-services-automatically-on-ubuntu-with-windows-10-startup/".

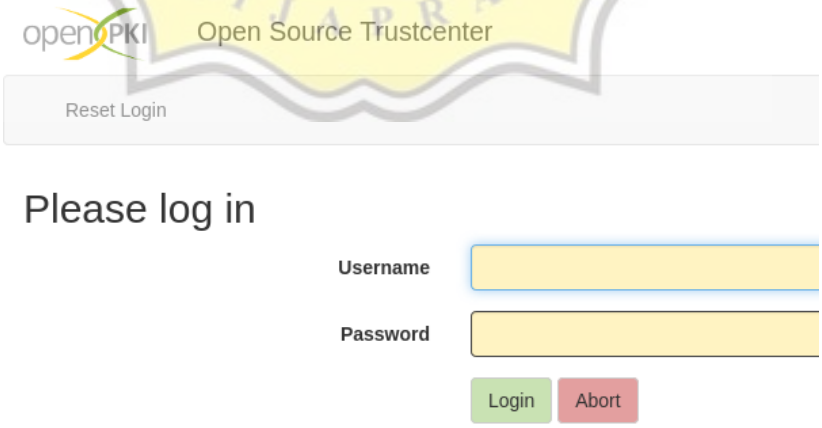
Salah satu hal yang cukup membantu dalam menggunakan OpenXPKI, OpenXPKI juga sudah dilengkapi dengan sistem RPC Server API yang detail spesifikasinya dapat dilihat dengan memanggil *endpoint* "./rpc/openapi-spec" (seperti pada gambar B.23), sehingga dapat dimungkinkan untuk melihat serta mengetahui *endpoint* apa saja yang dapat digunakan beserta *requirement body* dengan format

seperti apa yang harus dikirimkan. OpenXPKI juga memperbolehkan kustomisasi terhadap format sertifikat yang hendak dihasilkan, kustomisasi terhadap *web* OpenXPKI, konfigurasi API, dan sebagainya apabila hendak membuat konfigurasi sesuai dengan kebutuhan bisnis yang hendak digunakan pada *production*.



Gambar B.23: Tampilan respon sukses endpoint "/rpc/enroll/openapi-spec".

Berikut pada gambar B.24 adalah contoh tampilan *log-in* pada *website* OpenXPKI yang dapat diakses melalui *web browser* setelah selesai melakukan konfigurasi menggunakan *sample configuration*.



Gambar B.24: Website yang dihasilkan oleh script konfigurasi sampel diakses melalui web browser.

C. Persiapan Pengembangan Aplikasi Front-end

Aplikasi *front-end* akan dibuat menggunakan *framework* Expo. Untuk cara menginstalnya cukup mudah, sebelumnya harus memastikan bahwa Node.Js dan NPM telah terpasang dan dapat digunakan pada sistem yang akan digunakan. Untuk dapat membuat *Expo project* dan menginstal *library* yang terspesial untuk Expo atau juga menjalankan perintah "*Expo specific*" yang harus dilakukan adalah menginstal "expo-cli" dan juga "react-native-cli" secara global apabila hendak menjalankan *development* server serta melakukan *build* aplikasi react-native yang hendak dibuat. Caranya seperti pada kode B.24.

```
npm install -g expo-cli
npm install -g react-native-cli
```

Kode B.24 Command untuk menginstall expo-cli dan react-native-cli secara global

Apabila expo-cli telah terinstal, pembuatan *project* aplikasi expo dapat dilakukan. Caranya dengan cara mengetikan "expo init my-app" di mana "my-app" adalah nama project yang hendak dibuat. Kemudian pada proses setelah menjalankan perintah tersebut akan diminta untuk memilih tipe projek Expo yang akan dibuat apakah hendak menggunakan *bare workflow* atau *managed workflow*. Setelah memilih *workflow* yang hendak digunakan, proses pembuatan *project* aplikasi Expo akan berlanjut. Untuk pembuatan pertama kali tidak masalah apabila memilih "managed workflow" terlebih dahulu. Tetapi nanti diperlukan untuk melakukan "eject" agar dapat mengakses native code dari *project* aplikasi yang dibuat saat proses penyiapan PDFTron versi React Native.

Apabila proses pembuatan project Expo telah selesai, *folder* sesuai dengan nama *project* yang telah ditentukan dapat ditemukan dan *terminal* dimana perintah "expo init" berjalan akan menampilkan *list* perintah yang dapat dilakukan di dalam direktori Expo *project* yang telah dibuat untuk menjalankan project sesuai dengan *platform* yang dituju, kemudian akhirnya pengembangan aplikasi serta debugging secara bersamaan dengan perangkat yang berbeda dapat dilakukan.

Karena hendak menggunakan *library* PDFTron, terdapat beberapa langkah yang harus dilakukan. Langkah pertama adalah menginstall *library* PDFTron sesuai dengan *platform* yang dituju. Karena aplikasi inti akan dibuat pada *platform mobile* terlebih dahulu *library* yang diinstall adalah "react-native-pdftron" yang dapat diinstall. Apabila menggunakan package manager yarn cara menginstal adalah dengan

mengetikan “yarn add github:PDFTron/pdftron-react-native” atau apabila menggunakan npm dengan mengetikan “npm install github:PDFTron/pdftron-react-native” pada direktori projek yang sebelumnya telah dibuat. Kemudian langkah yang harus dilakukan adalah langkah-langkah penyiapan sesuai dengan *platform* yang hendak dituju (Android atau iOS) yang detailnya dapat dilihat pada dokumentasi GitHub PDFTron di “https://github.com/PDFTron/pdftron-react-native”. Lalu apabila hendak mengembangkan bersamaan juga dengan *platform web*, *library* PDFTron WebViewer harus diinstal dengan cara mengetikan “npm install @pdftron/webviewer” pada terminal, atau apabila hendak menggunakan yarn dapat mengetikan “yarn add @pdftron/webviewer” pada terminal di dalam direktori projek yang telah dibuat sebelumnya. Kemudian yang harus dilakukan adalah mengunduh *script* yang akan digunakan untuk mengcopy *asset web viewer* agar dapat selalu tersedia pada direktori “view” yang ada dalam folder “web” pada *project* Expo. *Script* JavaScript tersebut dapat diunduh pada “https://github.com/PDFTron/webviewer-react-sample/tree/master/tools” kemudian pada *file* “package.json” dapat ditambahkan juga perintah untuk menjalankan *script* tersebut pada saat *project* dijalankan seperti pada kode B.25.

```
"scripts": {
   //script lain yang sebelumnya sudah ada,
   "postinstall": "node tools/copy-webviewer-files.js"
}
```

Kode B.25 Mendaftarkan kode copy-webviewer-files yang akan berjalan begitu project dijalankan.

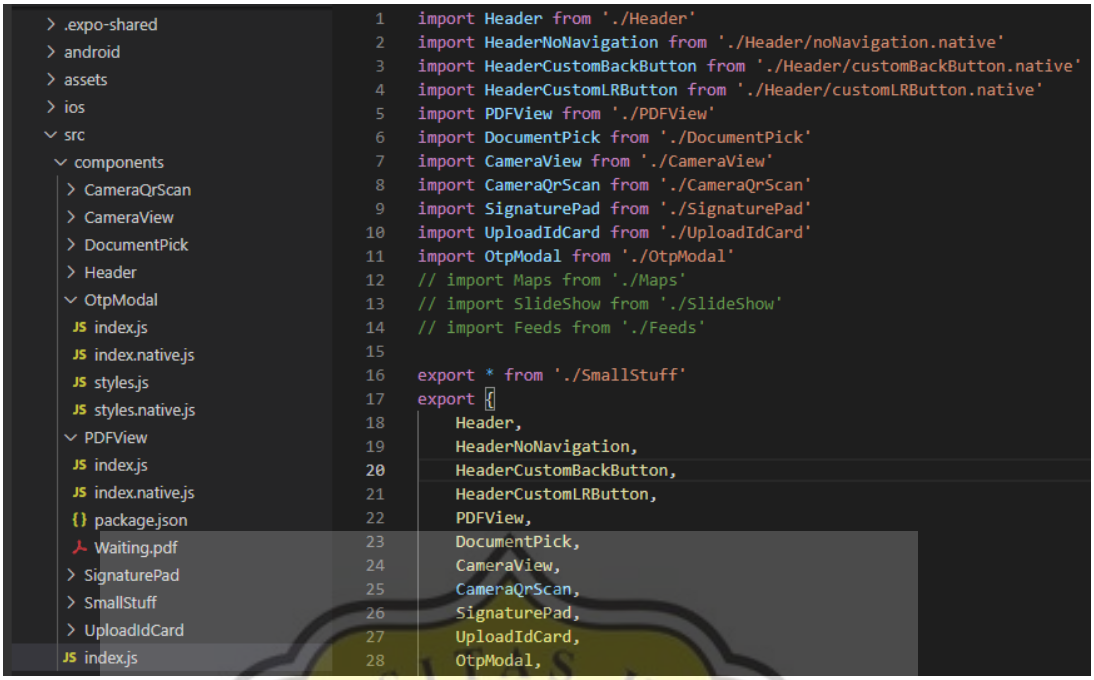
Lalu secara normal apabila sebelumnya membuat *project* Expo dengan *workflow* “*managed workflow*” direktori “web” tidak akan dapat ditemukan yang mana akan membuat *script* yang barusan ditambahkan tersebut tidak dapat berjalan dengan semestinya dan apabila menggunakan PDFTron *web viewer* pada *web* tidak akan tampil seperti yang diharapkan, dan konfigurasi pada masing-masing *platform* android atau ios apabila dibutuhkan tidak akan dapat dilakukan. Yang harus dilakukan apabila adalah melakukan *eject* pada *project* Expo dengan mengetikan “expo eject” pada *terminal* di dalam direktori project Expo yang telah dibuat sebelumnya, kemudian mengikuti setiap langkah yang ada, dan pada langkah tersebut akan diminta memilih apa saja *file* yang harus diekspos ke *project folder* secara terbuka seperti babel.config.js, metro.config.js dan sebagainya. Apabila proses telah selesai untuk mengekspos folder

“web” dapat dilakukan dengan mengetikan “expo customize:web” pada terminal dan seperti sebelumnya akan diminta untuk memilih *file* apa saja yang hendak diekspos, apabila tidak yakin tidak masalah untuk memilih semua. Apabila sudah, komponen PDFTron *platform* spesifik yang dapat dipanggil pada dua jenis aplikasi yaitu aplikasi *web* dan *mobile* (ios dan android) dapat mulai dibuat.

Perlu diingat bahwa apabila hendak melakukan aktivitas spesifik untuk *platform* Android seperti melakukan gradle *update* dan *build* aplikasi tanpa menggunakan Expo, terdapat langkah tambahan yang harus dilakukan. Yaitu menyiapkan SDK Android sesuai dengan sistem operasi yang digunakan. Cara yang paling mudah adalah dengan melakukan penginstalan Android Studio. Tetapi apabila dirasa tidak membutuhkan Android Studio secara keseluruhan atau karena Android Studio dirasa terlalu berat dan memakan cukup banyak *space* pada *disk*, dapat juga menggunakan “*Command Line Tools*” seperti yang dijelaskan pada artikel berikut apabila menggunakan sistem operasi berbasis Linux atau MacOS “https://proandroiddev.com/how-to-setup-android-sdk-without-android-studio-6d60d0f2812a”, dan apabila menggunakan Windows dapat dilihat pada artikel berikut “https://cloudreports.net/install-android-sdk-on-windows-10-without-android-studio/”. Kemudian yang harus dilakukan lagi adalah mengeset direktori SDK Android pada *file* “local.properties” yang dapat ditemukan pada *folder* “android” pada direktori project Expo apabila sebelumnya telah melakukan *eject* pada Expo project dengan *managed workflow*.

Apabila persiapan di atas telah dilakukan, aplikasi siap dikembangkan lebih lanjut sesuai dengan kebutuhan dan tujuan yang dituju. Apabila belum mempunyai dasar atau pengalaman membuat aplikasi berbasis React Native, terdapat banyak sekali contoh yang tersebar di internet seperti cara mengembangkan aplikasi menggunakan React Native, bagaimana cara menggunakan React Navigation, dan *library* pendukung lainnya sesuai dengan kebutuhan dari aplikasi yang hendak dibangun agar tujuan dari pembuatan aplikasi dapat tercapai dan aplikasi dapat dikerjakan dengan baik.

Berikut adalah contoh struktur folder React Component pada project Expo dapat dilihat pada gambar C.1.



Gambar C.1: Contoh struktur folder React Component.

Umumnya saat mengembangkan aplikasi berbasis React Native, masing-masing komponen dipisahkan dan dikemas menjadi satu folder misalnya untuk menampilkan PDF dalam folder tersebut terdapat dua *file* "index.js" yang akan diakses pada perangkat *web* (atau pada *mobile* juga apabila tidak terdapat *file* dengan nama ....native.js) atau "index.native.js" yang akan diakses oleh perangkat *mobile* (dapat dilihat juga pada gambar C.1).

Apabila hendak menyiapkan komponen yang menggunakan PDFTron, pada folder komponen tersebut dibuat *file* "package.json", di mana seperti contoh kode di bawah (kode B.26) bahwa nilai "name" diisikan dengan sesuai nama komponen yang akan dibuat.

```
{
  "name": "PDFView",
  "version": "0.0.0",
  "private": true,
  "main": "./"
}
```

Kode B.26 File "package.json" yang berada pada direktori React component PDFTron.

Kemudian untuk mempelajari konfigurasi PDFTron secara lebih lengkap dan jelas, terdapat artikel blog PDFTron yang dapat diakses melalui URL "https://www.pdftron.com/blog/mobile/react-and-react-native-pdf-viewers/", dan

melalui video Youtube channel resmi PDFTron pada URL berikut “https://youtu.be/p6eVk34wVDI”. Atau dapat juga melihat contoh pada kode yang telah disematkan pada penelitian ini pada pembahasan tentang bagaimana proses penandatanganan berlangsung.

17.N1.0013.docx
Oct 19, 2021
12369 words / 81950 characters

17.N1.0013.docx

Sources Overview

2%
OVERALL SIMILARITY

1	docplayer.info INTERNET	<1%
2	123dok.com INTERNET	<1%
3	SDM Universitas Gadjah Mada on 2021-10-08 SUBMITTED WORKS	<1%
4	repository.unpar.ac.id INTERNET	<1%
5	Universitas Islam Indonesia on 2018-08-07 SUBMITTED WORKS	<1%
6	Universitas Negeri Jakarta on 2017-08-03 SUBMITTED WORKS	<1%
7	kmtg.ft.ugm.ac.id INTERNET	<1%
8	media.neliti.com INTERNET	<1%
9	sucirohaini.blogspot.com INTERNET	<1%
10	Universitas Brawijaya on 2018-07-15 SUBMITTED WORKS	<1%
11	databoks.katadata.co.id INTERNET	<1%
12	docplayer.net INTERNET	<1%
13	Universitas Brawijaya on 2017-12-30 SUBMITTED WORKS	<1%
14	commune.dreawer.com INTERNET	<1%
15	dosenit.com INTERNET	<1%
16	indradimarza.blogspot.com INTERNET	<1%
17	iwaandjrs.blogspot.com INTERNET	<1%
18	jurnal.untan.ac.id INTERNET	<1%