Senior Node.js Back End Engineer Assignment

You are building a back-end for a wallet app in Nest.js and TypeScript that will be capable of taking care of large amounts of requests.

📝 Assignment description:

Wallet will have two microservices that will communicate between each other, one for calculations (lets call it `wallet-processor`) and another that partners will use for integration with our system (lets call it `wallet-api`). When requests comes from the partner, `wallet-api` will determine which transaction should pass and send those to the `wallet-processor` that will process the transactions. `wallet-processor` should never be throttled and that's the reason why `wallet-api` should arrange the transactions before sending them to the `wallet-processor`. Both microservices shouldn't require more than `1024MB` of memory to run smoothly.

🌐 `wallet-api` details:

`wallet-api` should separate the transactions into chunks, and all chunks should be sent to the `wallet-processor`, by priority. This is how one transaction will look like:

```
[
  { value: 110, latency: 600, customerId: "..." },
  { value: 70, latency: 250, customerId: "..." },
  { value: 200, latency: 850, customerId: "..." },
  { value: 120, latency: 1000, customerId: "..." },
  { value: 20, latency: 50, customerId: "..." },
  { value: 40, latency: 100, customerId: "..." },
]
```

You should develop a splitting algorithm that will separate transactions into chunks and those chunks will be sent to the `wallet-processor`. You should make sure to not have more than **1000ms** latency per chunk and to have the highest possible value per chunk. So a higher value chunk should be sent first to the `wallet-processor`. Here is what split transactions should look like (sorted by priority by which they will be sent).

```
chunk 1:
        transactions: [{"value":200,"latency":850},{"value":40,"latency":100},{"value":20,"latency":50}]
        total value: 260
        time left: 0
chunk 2:
        transactions: [{"value":130,"latency":600},{"value":70,"latency":250}]
        total value: 200
        time left: 150
chunk 3:
        transactions: [{"value":120,"latency":1000}]
        total value: 120
        time left: 0
```

`wallet-api` will have the following endpoints:

- POST `/transaction` - Endpoint protected by api key which will receive the transaction array (the one described above) separate them into chunks and send it to the `wallet-processor`. It should not wait for the processing all of them to return, but should store unsuccessful ones. (eg. customer doesn't have enough balance).
- GET `/customer/:id` - Will return current customer name and balance if called with api key and if not it should return only customers name.
- DELETE `/customer/:id` - Should soft delete customer.
- PATCH `/customer/:id` - Update customers name and balance.

`wallet-processor` details:

`wallet-processor` will be able to process only one chunk at a time and it will communicate with MongoDB and will be responsible for updating the customer balance in the DB.

Value received in transaction should be subtracted from customer balance. If customer doesn't have enough balance this transaction should be stored as unsuccessful.

Customer schema should have the following fields:

```
_id: ObjectId,
uid: String,
password: String,
first_name: String,
last_name: String,
username: String,
gender: String,
phone_number: String,
social_insurance_number: String,
avatar: String,
date_of_birth: Date,
email: String,
employment: {
  title: String,
  key_skill: String,
}
credit_card: {
  ballance: Double,
  cc_number: String,
}
address: {
  coordinates: {
    lat: Double,
    lng: Double,
  },
  country: String,
  state: String,
  street_address: String,
  street_name: String,
  zip_code: String,
};
```

```
  createdAt: Date,
  updatedAt: Date,
  __v: Integer,
  ...
```

you can also add new fields if you need some additional ones.

💾 `seeding` details:

You should write a method that will take the current DB snapshot from S3 and seed it in your DB. This should happen every time an application starts, and it check if seeding already happened and to run it if needed. Even if the application is processing the snapshot, it should not reject requests coming from the front end, but should be able to process them after seeding is finished. DB dump can be found in `nt-interview-files` S3 bucket under `data.json` in `eu-central-1` region.

💡 **HINT**: since this file is publicly available you don't need any AWS credentials. Using AWS CLI you can pass `--no-sign-reques` flag, but AWS SDK doesn't have support for unsigned request. Workaround for this is to pass custom `signer` to the `S3Client` like this:

```
new S3Client({
  signer: { sign: async request => request }
});
```

🎉 Bonus points! (not mandatory)

- Create a scheduler that will take care of unsuccessful transactions and retry them after 1 hour.
- Write e2e and unit tests.
- Add swagger.

🤝 Assignment handover:

- Store project into private GitHub repo, and add `@fr1sk` as well as `@madicnikola` as a collaborators. Also send the repo URL to the hiring manager.
- Provide a postman collection for your `wallet-api`.
- Write documentation in `README.md` how project should be setup and used.
- Please state that it's not necessary to have everything, most important is for us to see your way of thinking. For any questions, please, do not hesitate to contact me at marko.jovanovic@neotechsolutions.org.

Best of luck! 🤞🍀
Neotech Solutions