Server (@app/server)

This is a back-end server which provides data access and business logic to the front-end applications.

Structure

Directory	Description
config	Configuration
db	Database utilities
express	Express application
graphql	Apollo GraphQL
logger	Logging utility
models	Database models
modules	Server functionality
resolvers	GraphQL resolvers
types	Custom types and interfaces

Config

Configuration of any sort is provided via constants and environment variables that are located in the `~/config` directory.

- Check `.env.config` to see how environment variables are/should be set
- Check `~/config/env` to see how environment variables are configured

Database

Database utilites are located in the `~/db` directory.

Utility	Description
tasks	Database initialization tasks
connect	Database connection
error	Custom error implementation
initialize	Database initializer
repository	Entity manager (CRUD)

- Database connection URI is set via `DB_URI` environment variable
- Database initialization tasks can be configured via `DB_INIT` environment variable:
    - Tasks: `reset`, `seed`, `mock`, `migrate`

- Initializer looks for the tasks to be executed
- See `~/db/initialize` for more details

Express

- Custom Express middlewares are/should be located at `~/express/middlewares`

Middleware	Description
auth	Custom authentication middleware

GraphQL

- Custom GraphQL scalars are/should be located at `~/graphql/scalars`
- Custom GraphQL middlewares are/should be located at `~/graphql/middlewares`
- Apollo Server can be configured via environment variables starting with `GQL_`
- Apollo Studio can be configured via environment variable `APOLLO_KEY`

Auth Checker

Queries and mutation can be authorized by user roles:

```
@Authorized() // can be executed by only logged in users
@Authorized([]) // can be executed by only logged in users
@Authorized([UserRole.ADMIN]) // can be executed only by admins
```

Auth checker can be enabled/disabled via environment variable `GQL_AUTH`:

Config	Description
false	Disabled
true (default)	Enabled
user roles as comma-separated values	Enabled only for the specified roles

Logger

Logger is a custom console API that provides conditional logging and the capability of storing logs in various environments which is located in the `~/logger` directory.

```
const logger = Logger.create({
  // Origin
  src: 'my-module',
  // Save log messages to the files (my-module_<date#yyyymmdd>.log)
  // if the log level of the message is error and below
  file: 'error',
  // Print log messages to the console
  // if the log level of the message is debug and below
```

```
  level: 'debug'
})
```

Below methods can be used for logging in addition to the standard console methods (debug, error, info, warn, trace):

- **fatal**: For critical errors
- **success**: For success messages
- **todo**: For todo messages
- **newline**: For printing new lines to the console

Levels

Level	Severity	Description
off	0	Silent mode
fatal	1	Critical errors
error	2	Errors
success	3	Success messages
info	4	Info messages
warn	5	Warnings
todo	6	To-do messages
debug	7	Debugging
trace	8	Tracing

Storage

- [ ] db: saves logs to the database
- [x] file: saves logs in the filesystem
- [ ] remote: sends logs to a remote server

Models

Database/business models (entities and embed documents) can be located in the `~/models` directory.

Directory	Description
embed	Embed models for the entities
entity	Database entities
entity/base	Base entity
entity/log	Log entity
entity/user	User entity

Network models:

Directory	Description
entity/antenna	Antenna
entity/parent	Base model for topological parents
entity/bsc	BSC (derived from entity/parent)
entity/rnc	RNC (derived from entity/parent)
entity/tac	TAC (derived from entity/parent)
entity/lac	LAC (derived from entity/parent)
entity/site	Base station
entity/cell	Base model for cells
entity/cell2g	2G cell (derived from entity/cell)
entity/cell3g	3G cell (derived from entity/cell)
entity/cell4g	4G cell (derived from entity/cell)

Network Topology:

- Parent: BSC/RNC/TAC
    - Site
        - Cell: Cell2G, Cell3G, Cell4G

Embed models

- should be located in their own directory under `~/models/embed`
- should use `EmbedModel` (`~/models/embed/options.ts`)

Entity models

- should be located in their own directory under `~/models/entity`
- should be derived from the base entity (`~/models/entity/base`)
- should use `EntityModel` (`~/models/entity/options.ts`)

Structure:

File	Description
const	Default values and other constants
enum	Enumerations
input	Input models for queries
type	The object model - represented in the database
types	Types and interfaces

File	Description
utility	Utilities related to the model (e.g. DTOs)

Modules

Modules which are located in the `~/modules` directory extend the functionality of the server.

Module	Description
auth	Authentication and authorization
fs	File system and uploads

Resolvers

GraphQL resolvers which are located in the `~/resolvers` directory, provide data to the front-end applications via GraphQL queries, mutations and subscriptions.

Built-in resolvers:

Resolver	Description
auth	Authentication API
common	API for non-model data