IL TECHNICAL DOCUMENT:

Demystifying the IL

Purpose: To help developers quickly get to understand the IL codebase

Intended Audience: Developers

Background:
The IL is composed of 2 parts, the backend and the frontend.

Summary of Technology Stack

Technology / Framework / Library	Description	Links
Backend		
Hapijs	Hapi is a nodejs framework for developing secure REST API.	https://hapijs.com/
Sequelize	Sequelize is a promise-based ORM for Node.js v4 and up. It supports the dialects PostgreSQL, MySQL, SQLite and MSSQL and features solid transaction support, relations, read replication and more.	http://docs.sequelizejs.com/
Bunyan	Bunyan is a simple and fast JSON logging library for node.js services	https://www.npmjs.com/package/bunyan
Glue	A server composer for hapi.js. Glue provides configuration based composition of hapi's Server object	https://github.com/hapijs/glue
Lout	lout is a documentation generator for hapi	https://github.com/hapijs/lout

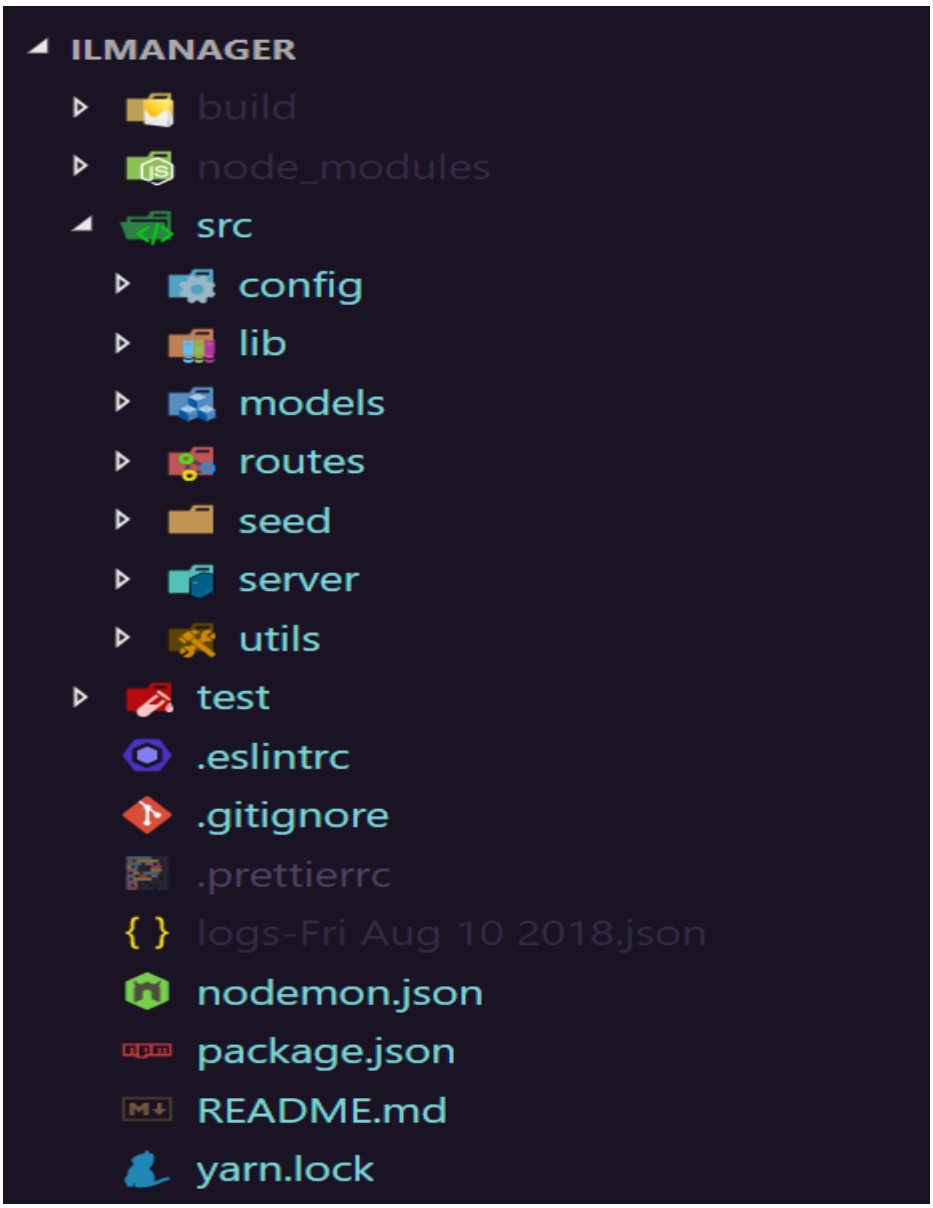
	servers, providing a human-readable guide for every endpoint using the route configuration. Under the hood it uses swagger	
Ping	a ping wrapper for nodejs	https://www.npmjs.com/package/ping
Dateformat	A nodejs package for human-readable dates	https://www.npmjs.com/package/dateformat
Boom	HTTP-friendly error objects	https://github.com/hapijs/boom
isomorphic-fetch	Used to consume REST endpoints	https://www.npmjs.com/package/isomorphic-fetch
pm2	Provides advanced production process management for nodejs apps	http://pm2.keymetrics.io/
Frontend		
React	React is a lightweight library for developing user interfaces	https://reactjs.org/
axios	Promise based HTTP client for the browser and node.js	https://www.npmjs.com/package/axios
semantic-ui-react	Semantic UI React is the official React integration for Semantic UI, which is a css framework for designing User Interfaces	https://react.semantic-ui.com/

react-drag-and-drop	Used to implement the drag and drop on the subscription page	https://www.npmjs.com/package/react-drag-and-drop
react-json-view	Used to display json messages in a tree-like grid, allowing users to browse through the json	https://www.npmjs.com/package/react-json-view
react-ace	Used to display xml messages in an interactive tree-like grid, allowing users to browse through the xml.	https://www.npmjs.com/package/react-ace
victory	Data visualization for react	https://www.npmjs.com/package/victory

The Backend: The backend is a nodejs application called ***ilmanager***. It is built on top of the hapijs framework.

- It interacts with a relational database, called the **interopdb.**

Directory Structure:



- The **src** folder contains all the source code for the project.

  Server

  This folder contains the **index.js** file which is the **entry point** to the project. The code in this file does the following:

  - Creates a hapijs server. The server is composed using glue
  - Creates a cluster of the following stateless services:

- DAD (Data Acquisition and Dispersal)
- VL (Viral Load)
- Message Dispatcher

Note
Nodejs by default runs on a single thread (single CPU core), we spawn new processes for each CPU core. We can only spawn multiple, concurrent processes for stateless services.
We use the nodejs internal clustering library to ensure a higher throughput for all the endpoints.

Config

- This folder contains:
  - ***Config.json*****:** Contains configuration details for the database in use during development and testing.However, in production we use the environment variables for storing these values.
  - ***Constants.js:*** contains *constants* used throughout the project
  - ***Manifest.json:*** used for composing a hapijs server

Lib

- Contains business logic for queueing, cleaning up and statistics.
- More information in the module section

Models

- Contains the entities, and their relationships described using sequelize.
- The *index.js* file checks which environment we are in (test/dev/prod), and applies the right configuration to this environment

Routes

- Contains both the HTTP and TCP endpoints

Seed

- This contains the seeding files. Data like message types, participating systems, lab codes, settings, known subscriptions etc are seeded on fresh installations.

Test

- Contains the unit tests for the various components.

Eslintrc

- This is the linting configuration file. The codebase follows the standard coding style. This keeps the code consistent, and easier for any new devs to take up.

The rest of the files are standard files for a nodejs application.

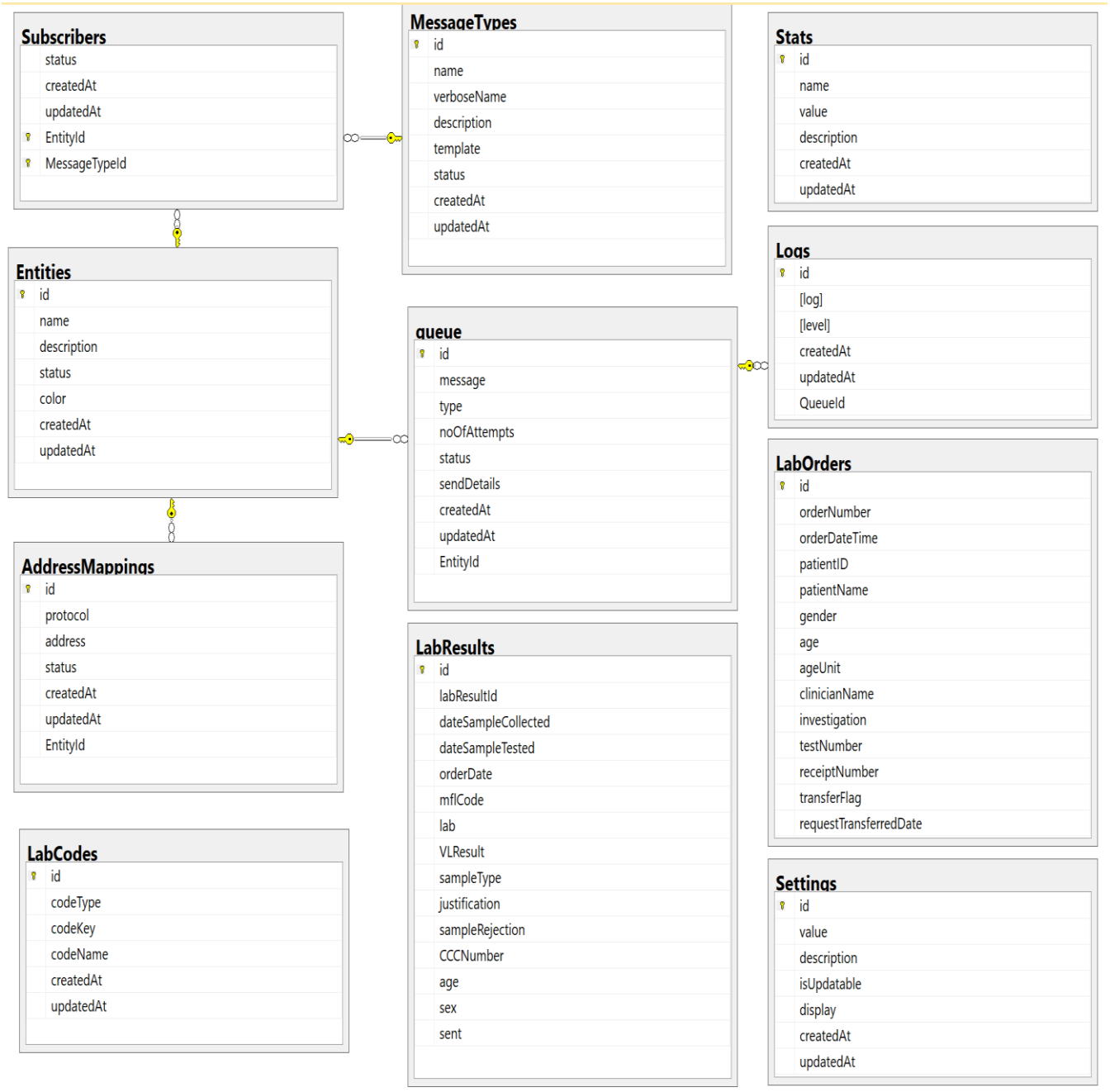
Modules:

- **DAD**
  - DAD stands for Data Acquisition and Dispersal
  - This is the module that receives all the incoming data from external systems.
  - It supports both TCP and HTTP protocols
  - The DAD module also implements endpoints for sending data to DHIS2
  - On receiving a new message, the DAD module validates the message, then sends the message to the queue.
- **Queue**
  - IL implements a database-based queueing system.
  - When new messages are received, they are saved in the queue.
  - A queue manager then runs through the queue, giving priority to those messages with the *least number of send attempts* and those that *have stayed in the queue the longest*, and then sends these out to message dispatcher module
- **Message Dispatcher**
  - The Message Dispatcher receives messages to be dispatched to the participating systems, gets the active participating systems with subscriptions on this message, then proceeds to send the message to the participating system.
- **Zookeeper**
  - The zookeeper module is a scheduled process that cleans after the IL.
  - It prunes old data from the logs and the queue tables.
  - Right now messages in the queue with a status of SENT and are 5 days old are regarded as old data, and are marked for pruning.
  - Moreover, Log messages with a level of INFO and are 5 days old are also regarded as old data, and are marked for pruning.
  - The zookeeper is scheduled to run every 2 hrs to check if there are any items that need to be pruned.
- **Notifications**
  - This is the logging module.
  - We use the bunyan library for logging
  - We also store log messages that may be helpful for end users when troubleshooting, or following up on a message.
- **Lab Component(s)**
  - This component is currently processing only Viral Load messages, but was meant to go beyond that.
  - Both lab orders and lab results can be processed right now, although lab orders has not be tested on a production environment yet (A discussion with Rufus is necessary here).
  - For the lab results, we have endpoints for both the SMS-based (MLAB) results and the NASCOP EID server results via REST API.
  - The SMS-based results are base64 encoded, and coded to ensure the entire lab result can fit within the 160 characters range for an SMS.
  - On receiving new viral loads, the message is decoded, and used to construct a valid Viral Load message
- **Stats Component**

- This module is responsible for updating statistics for the dashboard area, to give end users a quick snapshot of what's going on

- **Web UI**
  - This is just an endpoint for serving the frontend build. This is done so that we don't have a dependency on any web server. In this way, IL ends up with a very small footprint when deployed, and can run on any machine!
- **Data Seeding**
  - This module runs the seed files to populate the database during initial installs.

Interopdb:



Layers:

Endpoints:

- These provide the interface exposed by the IL for interaction with other systems.
- We have both HTTP (REST) endpoints and TCP endpoints.

HTTP endpoints:

- Hapijs (A nodejs library for developing REST endpoints) is used to create the endpoints.
- Isomorphic-fetch is used to consume REST endpoints (e.g. when sending data out)

TCP endpoints:

- Net is used to send and receive data via sockets

ORM (Object Relational Mapper)

- Sequelize (A simple nodejs library for modelling your entities and interacting with relational databases)
- Most of the logic in the IL is database logic that involves interactions between the entities. Sequelize is therefore widely used in such cases.

Database

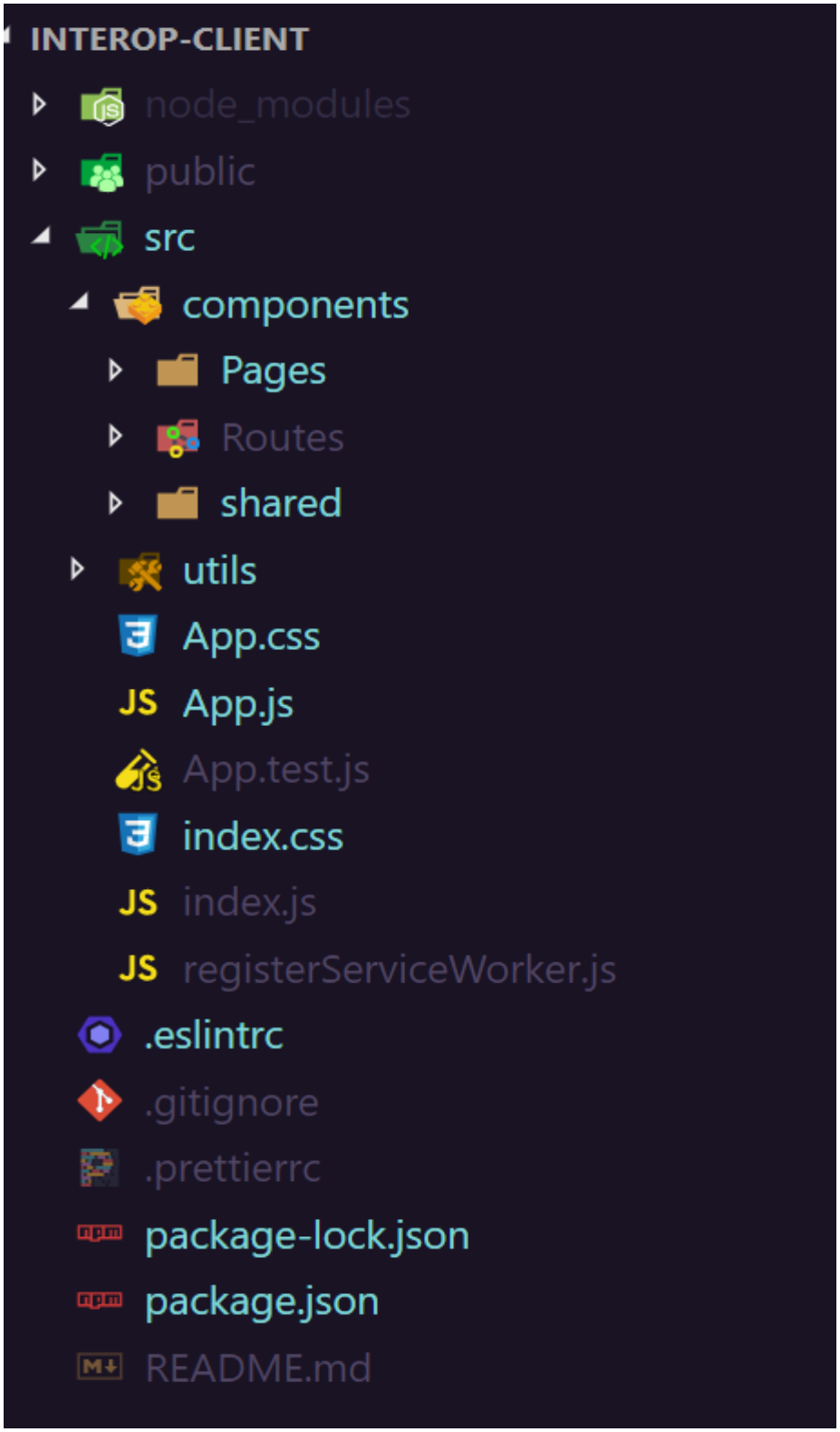
- Out of the box, the IL supports the following DBMS:
  - Mysql for linux/windows installations (this is the preferred DBMS for kenyaemr installations)
  - Microsoft SQL for windows installations (this is the preferred DBMS for IQCare installations)
  - Sqlite for unit tests
- To add support for any new DBMS (lets take **postgres** as an example), do the following:
    - Run the following at the root of the project

```
# Using NPM:
$ npm i -S pg pg-hstore
        --- OR ---
// Using Yarn
$ yarn add pg pg-hstore
```

Frontend:

- Built using react.
- The project is meant to provide details about the exchange, and to enable the initial setup of the exchange.

Directory Structure:



- The **src** folder contains all the source code for the project.

Components:

- This contains all the components on the project.

  Pages:

  - This contains the code for all the pages created, and any other dependency that's not shared between pages

  Routes:

  - This contains the mapping of the path (url) to the pages. It houses all the routes available in the project.

  Shared:

  - This contains all the components shared between different pages.

Utils:

- This contains all the interactions with the backend API endpoints.

Installers:

Linux

- We use shell script for this.
- PM2 is the process manager used to ensure IL runs as a service even on restarts

Windows

- We use advanced installer for this.
- This helps us add IL as a service on windows, and use environment variables to store database configuration details.

Pending Work:

Message Validation

- This is a critical module. The requirement is as follows:
    - For each new message type added, a validation schema is also uploaded to accompany it.
    - The validation schema would have details like required fields, max/min length of a given field, data types of each field.
    - Each time a new message is received by the IL, it is validated against this message type's validation schema.
    - The result of the validation is then sent back as a response to the sending system.
    - The validation schema is not hard-coded into the codebase, rather it is part of the configuration details uploaded whenever a new message type is created.
    - The approach is to use joi for the json object schema validation, and make it so that the uploaded schema file/data, can be ran against the message received by IL.

ACK Design and Implementation

- ACK is an acknowledgement message sent back to the sending system, to notify them of the status of the message.
-