# IOTICS documentation

### Server-Side Application (Node.js) & Desktop Application (Electron.js & Vue.js)

**Version:** 1.0.0

**Date:** September 20, 2024

**Authors:**

* Bakhodir Abdurakhmonov – CEO / Product Manager
* Salokhbek Eshmirzayev – Main Frontend Developer
* Akmal Ahmadjonov – Backend Developer
* Jamshid Nasyrov – UI/UX Designer
* Azamjon Abdullayev – Frontend Developer

**Contact:**

* Email: [b.abduraxmanov.b@gmail.com](mailto:b.abduraxmanov.b@gmail.com)
* Website: [www.iotics.uz](https://www.iotics.uz)
* Phone: +998 94 675 11 00

Tashkent - 2024

**Project Description:**

The **IOTICS** is a comprehensive solution designed to streamline the monitoring and management of energy systems for businesses of all sizes. It consists of a server-side application developed with Node.js and a desktop application built using Electron.js and Vue.js. The project aims to provide real-time insights and control over various energy parameters, ensuring efficient and cost-effective energy utilization.

### **Key Features:**

* **Real-Time Data Monitoring:** The desktop application displays live data from multiple energy sources, including electricity, water, heat, and gas meters.
* **Data Analysis and Reporting:** Users can analyze historical data, generate detailed reports, and identify patterns to optimize energy consumption.
* **User Management:** The server-side application supports role-based access control, allowing administrators to manage user permissions and access levels.
* **Custom Alerts and Notifications:** Set up custom alerts for specific energy usage thresholds to receive timely notifications.
* **Secure Data Transmission:** The application ensures secure communication between the desktop client and the server using encryption protocols, which is the author function of the developers of IOTICS, based on RSA protocol with public key - private key exchange.

### **Technology Stack:**

* **Server-Side Application:** Built with Node.js, leveraging Express.js for routing and MongoDB for data storage.
* **Desktop Application:** Developed using Electron.js for cross-platform compatibility and Vue.js for a responsive and dynamic user interface.
* **Data Visualization:** No library or package is used. Only clean code to visualize the data to the user, based on UI/UX.

### **Purpose and Scope:**

The primary goal of this project is to help organizations monitor and optimize their energy consumption. By providing a centralized platform for data collection, visualization, and analysis, **IOTICS** enables businesses to make informed decisions, reduce operational costs, and contribute to sustainable energy practices.

### **Target Audience:**

This documentation is intended for developers, technical users, and system administrators who are involved in setting up, maintaining, or extending the **IOTICS** application. It provides detailed instructions on installation, configuration, and usage to ensure a seamless experience.

# Table of contents

## 

[**1. Project Overview 4**](#_nixr43umned2)

[Core Objectives: 4](#_kuje9q8yot22)

[Key Components: 4](#_mn4faeogyt7y)

[1. Server-Side Application (Node.js): 4](#_vydja115vcyl)

[2. Desktop Application (Electron.js & Vue.js): 4](#_1nawgezhhaw5)

[Technological Advantages: 4](#_fnkx7ymyz6q3)

[Target Users: 4](#_1lfbfmfzqira)

[Future Enhancements: 5](#_h9ixp51buln8)

[**2. Installation and Setup 5**](#_rb8bvvz1t7kn)

[Requirements 5](#_8vmqur9oznd8)

[Installation Steps 5](#_gbaez95rpd78)

[1. Clone the Repository 5](#_eyuip6am0w45)

[2. Set Up the Server-Side Application 5](#_tf9zhq4zp82v)

[3. Install and Set Up the Desktop Application 6](#_aymx3j8r5iaj)

[**3. Project Structure 6**](#_iafl1mfn6xmz)

[1. Server-Side Application (Node.js) 6](#_hg7amrm2vekq)

[Key Directories and Files: 11](#_2n746wfz8kzz)

[2. Desktop Application (Electron.js & Vue.js) 12](#_b31rd1hkb15g)

[Key Directories and Files: 12](#_lv5axtmki7gv)

[**4. API Documentation 13**](#_9jr2mdbyzfb8)

[Base URL 13](#_9sa0y3wml5xj)

[Endpoints Overview 13](#_ev78plsd35dd)

[**5. Interaction Between Server and Desktop Applications 16**](#_9e96gzphsnzu)

[**6. Environment Configuration 17**](#_g9cue8nnswns)

[1. Server-Side Application (Node.js) 17](#_ch0rz56glyip)

[2. Desktop Application (Electron.js & Vue.js) 17](#_740mksr9k4ck)

[**7. Development and Testing 18**](#_8dcxqzg9wgvt)

[**8. License and Authors 23**](#_jsgsbemtwqox)

## 

## Project Overview

**IOTICS** is an innovative energy management application designed to empower businesses of all sizes to monitor, analyze, and optimize their energy consumption. By integrating real-time data from various energy sources, IOTICS provides users with the tools they need to make informed decisions regarding their energy usage, ultimately leading to cost savings and enhanced sustainability.

#### 

#### **Core Objectives:**

* **Enhance Energy Efficiency:** IOTICS aims to identify inefficiencies in energy consumption patterns, enabling users to make adjustments that reduce waste and lower costs.
* **Real-Time Monitoring:** The application provides immediate access to energy data, allowing users to react quickly to any anomalies or unexpected spikes in usage.
* **Data-Driven Decision Making:** By presenting historical and real-time data in an accessible format, IOTICS supports strategic planning and long-term energy management goals.

#### **Key Components:**

###### Server-Side Application (Node.js):

* 1. Responsible for handling data collection, processing, and storage.
  2. Manages user authentication and access control.
  3. Provides a robust API for communication with the desktop application.

###### Desktop Application (Electron.js & Vue.js):

* 1. A cross-platform interface that allows users to visualize and interact with their energy data.
  2. Offers customizable dashboards and reporting features.
  3. Supports notifications and alerts for important changes in energy usage.

#### **Technological Advantages:**

* **Scalability:** Built to accommodate a growing number of devices and users without sacrificing performance.
* **Security:** Implements industry-standard security protocols to ensure data integrity and user privacy.
* **User-Friendly Interface:** Designed with an intuitive layout that simplifies data navigation and analysis.

#### **Target Users:**

IOTICS is targeted at facility managers, energy analysts, and business owners looking to gain deeper insights into their energy consumption. It is particularly beneficial for organizations in sectors such as manufacturing, retail, and commercial real estate, where energy costs represent a significant portion of operational expenses.

#### **Future Enhancements:**

Plans for future iterations of IOTICS include integrating machine learning algorithms for predictive analytics, expanding compatibility with additional energy meters, and enhancing mobile accessibility to ensure users can monitor their systems on-the-go.

## Installation and Setup

#### **Requirements**

Before installing IOTICS server-side application, ensure your system meets the following requirements:

* **Node.js**: Version 16.x or higher
* **npm**: Version 6.x or higher (comes bundled with Node.js)
* **MongoDB**: Version 6.x or higher (for the server-side application)
* **Git**: Optional, for cloning the repository

Before installing IOTICS client-side desktop application, ensure your system meets the following requirements:

* **Windows OS**: Version 7.x or higher

#### **Installation Steps**

##### **1. Clone the Repository**

First, authors need to clone the IOTICS repository from GitLab with authorization. Open the terminal and run:

bash

git clone https://gitlab.com/{project\_name\_in\_gitlab}.git

***\* The project name is hidden due to copyright.***

cd {project\_name\_in\_gitlab}

##### **2. Set Up the Server-Side Application**

**Navigate to the Server Directory:** bash  
 cd server

**Install Dependencies:** Use npm to install the required packages:  
 bash  
 npm install

**Configure Environment Variables:** Create a .env file in the server directory and configure the following variables:  
  
PORT={port\_depending\_on\_user\_requirements}

MONGODB\_URI=mongodb://localhost:27017/iotics

JWT\_SECRET={secret\_word\_for\_JWT}

***\* Replace {secret\_word\_for\_JWT} with a strong secret key for JWT authentication.***

**Start the Server:** After setting up the environment, start the server with:  
 bash  
 npm start

##### **3. Install and Set Up the Desktop Application**

**Get and install the .exe file for desktop applications:**  
 Install the .exe file provided by developers on your PC.

Enter the server location IP address.

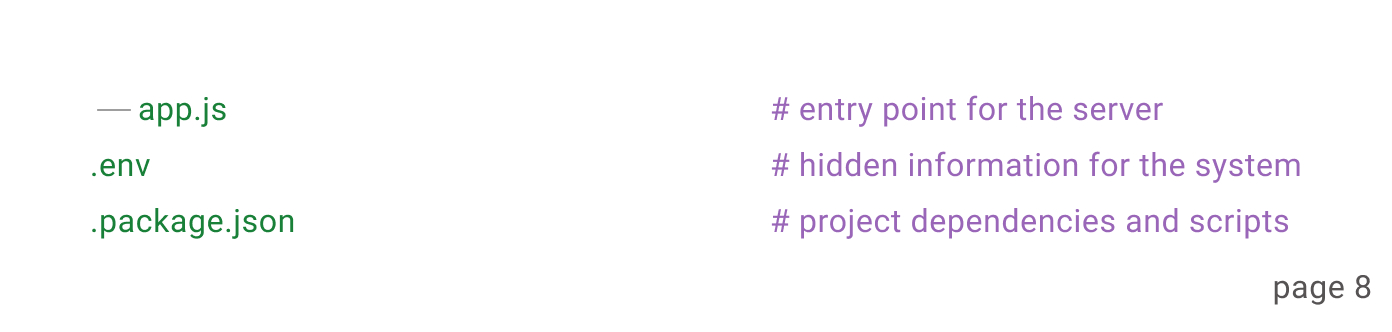
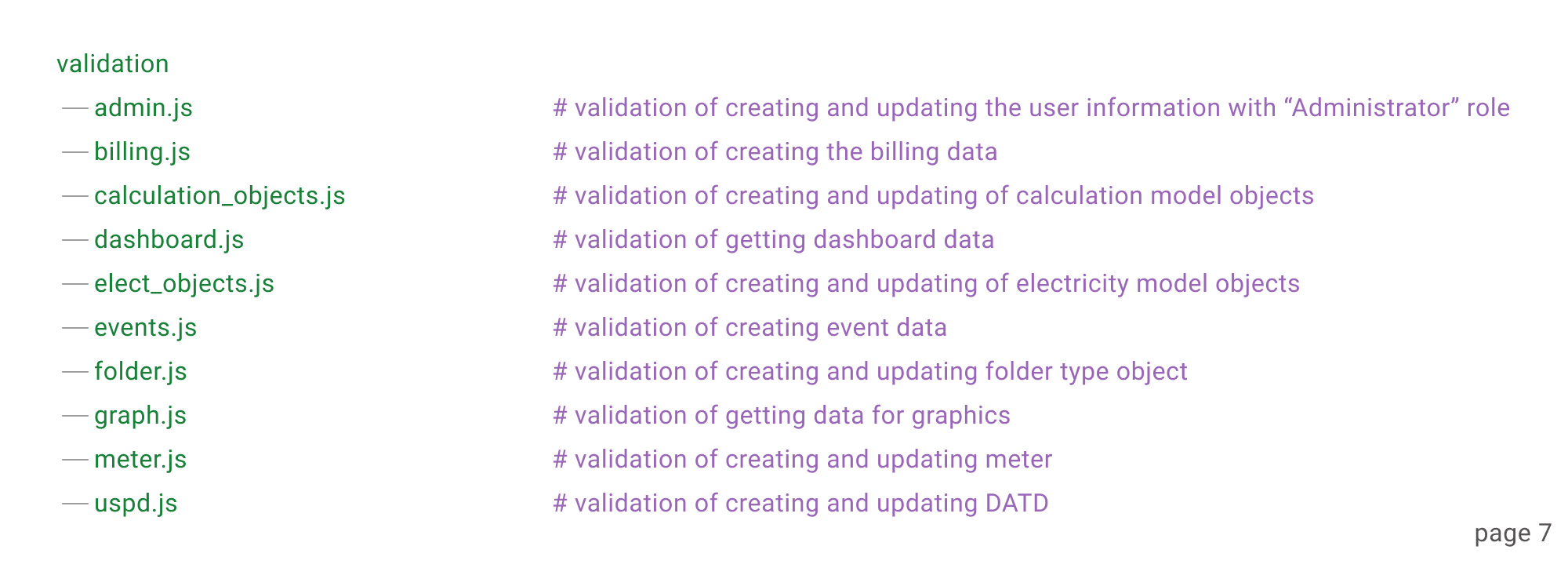
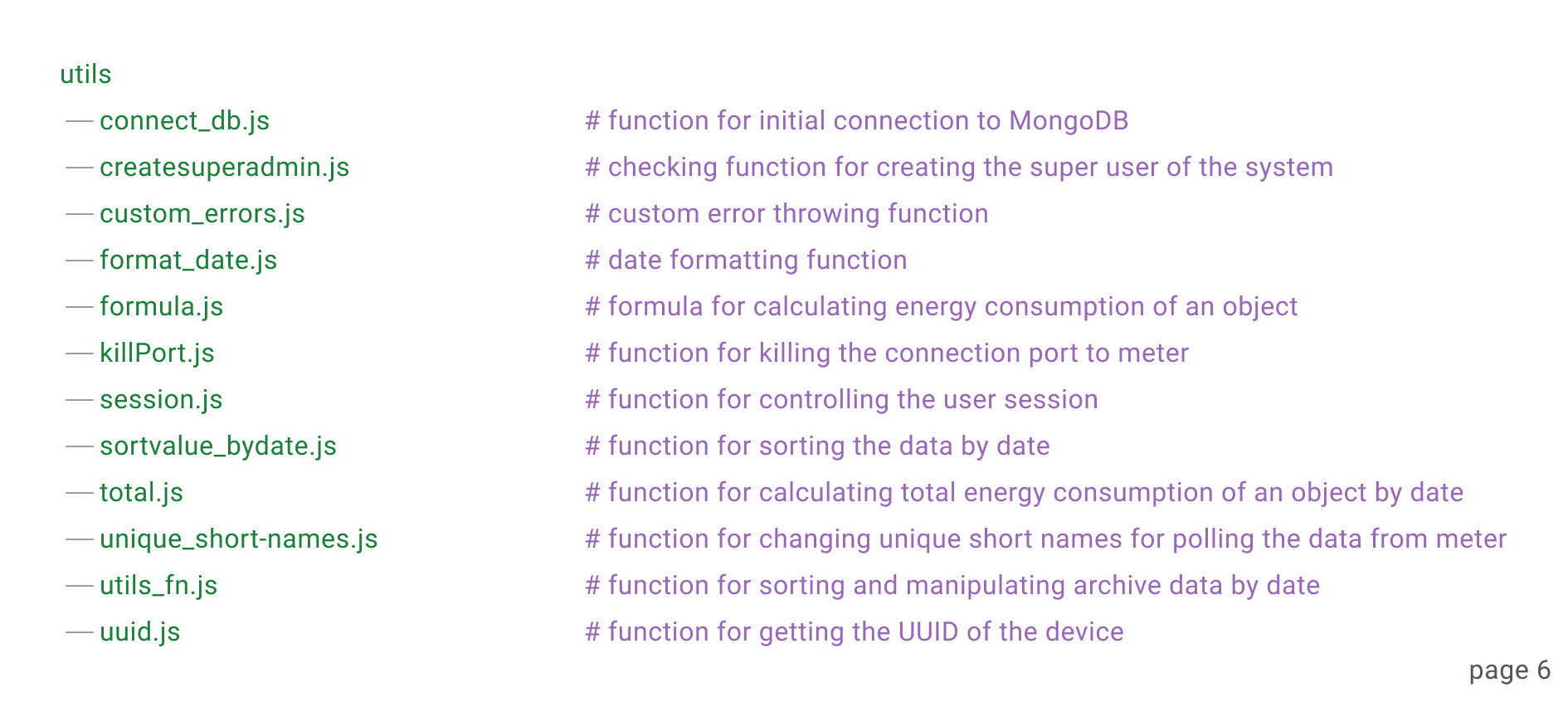
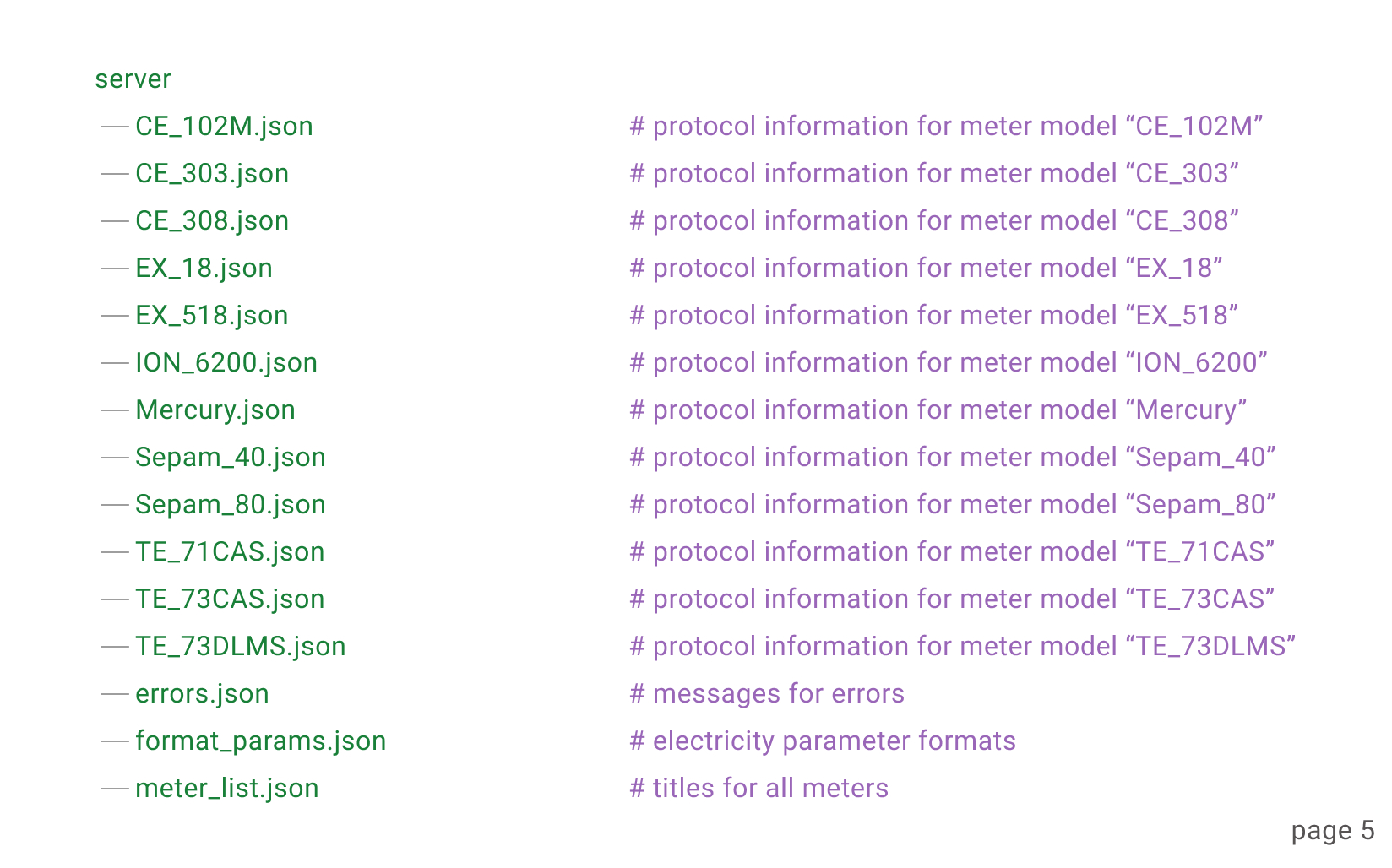
Enter the login and password provided by developers for the first authorization.

Create new users in the system or use the same authentication data for further usage of an application.

## Project Structure

The IOTICS project is organized into two main components: the server-side application and the desktop application. Below is an overview of the directory structure and key files for each component.

#### **1. Server-Side Application (Node.js)**



##### **Key Directories and Files:**

* **src/**: Contains all the source code for the application.
  + **connection/**: Contains connection code files.
  + **controllers/**: Contains the business logic for handling requests and responses.
  + **middleware/**: Contains the files of functions for data polling from meters.
  + **models/**: Defines the data structure and schema for MongoDB.
  + **repository/**: Contains the logic of CRUD for MongoDB.
  + **router/**: API endpoints for handling HTTP requests.
  + **server/**: Contains meter protocol information files.
  + **utils/**: Utility functions that can be reused throughout the application.
  + **validation/**: Contains data validation functions for all data used by the system.
  + **app.js**: Initializes and configures the Express application.
* **.env**: Contains environment-specific variables for configuring the application (e.g., database connection strings).
* **package.json**: Lists the project's dependencies, scripts, and metadata.

—-

#### **2. Desktop Application (Electron.js & Vue.js)**

Desktop application is built as an executive file, thus, the codes are hidden by Electron.js and can not be displayed to the user. But, all the business logic of the interface of an application is built with Vue.js.



##### **Key Directories and Files:**

* **build/icon**: Contains icons and built version of an application.
* **public**: Contains static files that are served directly.
  + **index.html**: The main HTML template for the application.
* **src/renderer**: Contains all the source code for the desktop application.
  + **assets/**: Stores static assets such as images and stylesheets.
  + **components/**: Reusable Vue.js components for building the user interface.
  + **events/**: Reusable functions for printing PDF files, decoding and asynchronous functions.
  + **locale/**: Contains language files.
  + **plugins/**: Contains route, language and license controlling files.
  + **router/**: Configures routes for navigation between different views.
  + **views/**: Defines the different views or pages within the application.
  + **App.vue**: The root Vue component for the application.
  + **main.js**: Initializes the Electron application and sets up the main process.
* **package.json**: Lists the project's dependencies, scripts, and metadata.
* **webpack.config.js**: Configuration for Webpack to bundle and build the application.

## API Documentation

The IOTICS server-side application provides a RESTful API that allows clients to interact with the system. This documentation covers the available endpoints, request parameters, response formats, and usage examples.

#### **Base URL**

The base URL for all API endpoints is:

http://{ip\_depending\_on\_client\_request}/api

#### **Endpoints Overview**

1. **User Management**
   * GET /admin/list - Retrieve a list of all users
   * POST /admin/create - Register a new user
   * PATCH /admin/update/:id - Update a user's information
   * PUT /admin/activate/:id - Activate deactivated user’s account
   * DELETE /admin/remove/:id - Deactivate a user account
2. **Login Management**
   * GET /single - Retrieve an information of the user
   * POST /login - Login into the system
   * POST /login - Logout from the system
   * POST /refresh\_token - Refresh the access token
   * POST /handshake - Handshake between server-side application and client desktop application with security algorithm
3. **Calculation Model Object Data Management**
   * GET /calculation/list - Retrieve the list of calculation model objects
   * GET /calculation/single/:id - Retrieve an information of calculation model object
   * GET /calculation/factories - Retrieve the list of only main calculation model objects
   * GET /calculation/report/:id - First type report data
   * GET /calculation/report-second/:id - Second type report data
   * GET /calculation/report-third/:id - Third type report data
   * GET /calculation/report-fourth/:id - Fourth type report data
   * POST /calculation/create - Create new calculation model object
   * PUT /calculation/update/:id - Update an information of calculation model object
   * PATCH /calculation/attach-params/:id - Attach electricity parameters to calculation model object
   * DELETE /calculation/delete/:id - Remove calculation model object
4. **Electricity Model Object Data Management**
   * GET /electricity/list - Retrieve the list of electricity model objects
   * GET /electricity/meter-use - Retrieve the list of meters for electricity model objects
   * GET /electricity/single/:id - Retrieve an information of electricity model object
   * GET /electricity/factories - Retrieve the list of only main electricity model objects
   * GET /electricity/report/:id - First type report data
   * GET /electricity/report-second/:id - Second type report data
   * GET /electricity/report-third/:id - Third type report data
   * GET /electricity/report-fourth/:id - Fourth type report data
   * POST /electricity/create-folder - Create new electricity model object
   * POST /electricity/create-meter - Attach to pseudo meter model object real meter
   * PUT /electricity/update-folder/:id - Update an information of electricity model object
   * PUT /electricity/update-meter/:id - Change real meter attached to pseudo meter model object
   * PATCH /electricity/attach-params/:id - Attach electricity parameters to electricity model object
   * DELETE /electricity/delete/:id - Remove electricity model object
5. **Dashboard Data Management**
   * GET /dashboard/data/:id - Retrieve the data of electricity model objects
   * GET /dashboard/calculation/:id - Retrieve the data of calculation model objects
   * GET /dashboard/real-time/:id - Retrieve the data of electricity model objects in real time
   * GET /dashboard/report/real-time/:id - Retrieve the data of calculation model objects in real time
6. **Graphs Data Management**
   * GET /graph/data/:id - Retrieve the data of electricity model objects with pagination
   * GET /graph/data-calculation/:id - Retrieve the data of calculation model objects with pagination
   * GET /graph/data-list/:id - Retrieve full data of electricity model objects
   * GET /graph/data-calculation-list/:id - Retrieve full data of calculation model objects
   * GET /graph/data-archive/:id - Retrieve an archive data of electricity model objects with pagination for data table
   * GET /graph/data-current/:id - Retrieve current data of electricity model objects with pagination for data table
   * GET /graph/calculation-archive/:id - Retrieve an archive data of calculation model objects with pagination for data table
   * GET /graph/calculation-current/:id - Retrieve current data of calculation model objects with pagination for data table
7. **Vector Diagram Data Management**
   * GET /vector/data/:id - Retrieve current data of meter for vector diagram drawing
8. **Meter Information Data Management**
   * GET /meter/list - Retrieve the list of all meters added to the poll server
   * GET /meter/one/:id - Retrieve an information of meter
   * GET /meter/params/:type - Retrieve the list of electricity parameters of meter
   * GET /meter/meter-list - Retrieve the list of all meter types
   * GET /meter/port-list - Retrieve the list of serial ports
   * POST /meter/create - Create meter with TCP connection
   * POST /meter/create-com - Create meter with serial port connection
   * PATCH /meter/update/:id - Update an information of meter with TCP connection
   * PATCH /meter/update-com/:id - Update an information of meter with serial port connection
   * DELETE /meter/remove-meter/:id - Remove meter
   * DELETE /meter/remove-folder/:id - Remove meters of chosen folder
9. **Billing Data Management**
   * GET /billing/table/:id - Retrieve the billing data for data table
   * GET /billing/:id - Retrieve the billing data for reports
10. **Connection Port Status Management**
    * GET /port/:port - Retrieve the status information of connection port
11. **Meter Events Data Management**
    * GET /event/types/:type - Retrieve the list of events of all types
    * GET /event/list/:id - Retrieve the list of events of chosen type
    * GET /event/report/:id - Retrieve the event data for reports
    * GET /event/last - Retrieve last event data
12. **Poll Server Folders Information Data Management**
    * GET /folder/list - Retrieve the list of all folders
    * GET /folder/one/:id - Retrieve an information of chosen folder
    * GET /folder/parents-list - Retrieve the list of all child folders of chosen folder
    * GET /folder/status/:id - Retrieve the list of all active meters of chosen folder
    * POST /folder/create - Create new folder
    * PATCH /folder/update/:id - Update an information of chosen folder
13. **External Dashboard Data Management**
    * GET /gm/all - Retrieve an information of all meters for external dashboard
    * GET /gm/single/:id - Retrieve an information of chosen meter for external dashboard
14. **External Dashboard Data Management**
    * GET /journal/last-inserted/:meter\_id - Retrieve an information of last data polling from chosen meter
    * GET /journal/last-successful-inserted/:meter\_id - Retrieve an information of last data polling with successful status from chosen meter
    * GET /journal/list/:meter\_id - Retrieve the status of data polling from chosen meter
15. **External Dashboard Data Management**
    * GET /license/data - Retrieve an information of installed license
    * GET /license/data-desktop - Retrieve a UUID of PC
    * POST /license/new-license - Install new license
16. **Mnemo Scheme Data Management**
    * GET /schema/all - Retrieve an information of all meters for mnemo scheme
    * GET /schema/single/:id - Retrieve an information of chosen meter for mnemo scheme
17. **Messages Management**
    * GET /send-message - Retrieve last message for user

## Interaction Between Server and Desktop Applications

The IOTICS platform is designed with a client-server architecture where the desktop application communicates with the server-side API to access and manage data. This interaction is crucial for maintaining synchronization between user actions on the desktop app and the centralized data stored on the server.

1. **User Actions on the Desktop Application:**
   * The user performs actions such as logging in, adding devices, or viewing energy data.
   * These actions trigger API calls to the server to either retrieve or update data.
2. **Server API Response:**
   * The server processes the requests, interacts with the database, and returns the appropriate response (e.g., data retrieval, confirmation of a successful operation).
3. **Data Display and Updates:**
   * The desktop application receives the server's response and updates the UI accordingly, ensuring that the displayed data is current and accurate.

## Environment Configuration

Proper environment configuration is essential for the smooth operation of the IOTICS application. This section outlines the environment variables and settings required for both the server-side and desktop applications.

#### **Server-Side Application (Node.js)**

The server-side application uses environment variables to configure settings like database connections, API keys, and server ports. These variables are typically stored in a .env file in the root directory of the project.

* 1. **Environment Variables**

| **Variable Name** | **Description** | **Example Value** |
| --- | --- | --- |
| **PORT** | **The port number on which the server will run.** | **8887** |
| **DB** | **The MongoDB connection string.** | **mongodb://localhost:27017/iotics** |
| **SECRET\_KEY** | **Secret key used for signing tokens.** | **i1ujhb2i3ujhiuh2i1u5hjo12!@oik12** |

* 1. **Example .env File**

**PORT = 8887**

**DB = mongodb://localhost:27017/iotics**

**SECRET\_KEY = i1ujhb2i3ujhiuh2i1u5hjo12!@oik12**

#### **Desktop Application (Electron.js & Vue.js)**

The desktop application uses environment variables to configure settings such as API endpoints and application mode. These variables are defined in the vue.config.js file and can be accessed in the Vue.js components.

* 1. **Environment Variables**

| **Variable Name** | **Description** | **Example Value** |
| --- | --- | --- |
| **VUE\_APP\_BASE\_URL** | **The base URL for the server-side API.** | **http://192.168.0.119:1200** |
| **VUE\_APP\_BASE\_CERTIFICATE** | **The base URL for the license file retrieving.** | **http://192.168.0.119:1100** |

* 1. **Example .env File**

**VUE\_APP\_BASE\_URL = http://192.168.0.119:1200**

**VUE\_APP\_BASE\_CERTIFICATE = http://192.168.0.119:1100**

## Development and Testing

This section outlines the development and testing practices followed in IOTICS, including development workflows, coding standards, testing methodologies, and automated testing setups for both the server-side and desktop applications.

1. **Development Workflow**
   1. **Version Control and Branching Strategy**
      1. **Version Control System:** The project uses Git for version control, with GitHub as the remote repository.
      2. **Branching Strategy:**
         1. **Main Branch (main):** The stable production-ready codebase. Only tested and approved code is merged into this branch.
         2. **Development Branch (dev):** The primary branch for integrating new features and updates. Developers branch off from dev to implement new features.
         3. **Feature Branches:** For individual features or bug fixes, developers create a branch named feature/feature-name or bugfix/issue-name and merge back into dev after completion and testing.
         4. **Hotfix Branches:** Urgent fixes for production issues are managed in branches named hotfix/issue-name and merged into both main and dev after resolution.
   2. **Code Reviews and Pull Requests**
      1. All changes must go through a code review process before being merged into the dev or main branch.
      2. **Pull Requests (PRs):**
         1. Include a detailed description of changes and reference any relevant issue numbers.
         2. Ensure all checks (linting, unit tests, integration tests) pass before requesting a review.
         3. At least one team member must review and approve the PR before merging.
   3. **Coding Standards**
      1. Follow the **Airbnb JavaScript Style Guide** for consistent code quality.
      2. Use ESLint with a pre-configured set of rules to enforce coding standards.
      3. Write clear, descriptive comments and JSDoc for all functions and classes.

Example JSDoc:

/\*\*

\* Calculates the total energy consumption for a device.

\* @param {Array} readings - An array of energy readings.

\* @returns {number} The total energy consumption in kWh.

\*/

function calculateTotalConsumption(readings) {

// Function logic

}

1. **Server-Side Application (Node.js)**

The server-side application follows a comprehensive testing strategy, including unit tests, integration tests, and automated testing pipelines.

1. **Testing Frameworks and Libraries**
   1. **Jest:** For unit and integration testing.
   2. **Supertest:** For testing HTTP endpoints.
   3. **Mocking Libraries:** Use sinon or jest-mock for mocking dependencies like database calls or third-party APIs.
2. **Unit Testing**
   1. **Purpose:** Verify the functionality of individual functions, classes, or components in isolation.
   2. **Structure:** Each unit test should focus on a single function or class method.
   3. **Coverage:** Aim for high coverage, especially for critical components such as business logic, data validation, and API handlers.

Example Unit Test (Jest):

const { calculateTotalConsumption } = require('../utils/energyUtils');

test('calculates total consumption correctly', () => {

const readings = [5, 10, 15]; // Mock data

const result = calculateTotalConsumption(readings);

expect(result).toBe(30);

});

1. **Integration Testing**
   1. **Purpose:** Test the interaction between multiple components and ensure they work together as expected.
   2. **Structure:** Integration tests should cover scenarios that span multiple functions, services, or database interactions.
   3. **HTTP Endpoint Testing:** Use Supertest to make HTTP requests to your API endpoints and validate responses.

Example Integration Test:

const request = require('supertest');

const app = require('../app'); // Express app

describe('GET /api/devices', () => {

test('returns a list of devices', async () => {

const response = await request(app).get('/api/meters);

expect(response.statusCode).toBe(200);

expect(response.body).toBeInstanceOf(Array);

});

});

1. **Test Automation and CI/CD**
   1. **Continuous Integration (CI):** Use GitHub Actions, CircleCI, or Travis CI to automatically run tests on each push or pull request.
   2. **Continuous Deployment (CD):** Automatically deploy to staging or production environments after passing all tests and manual approvals.

Example GitHub Actions Workflow:

name: Node.js CI

on: [push, pull\_request]

jobs:

build:

runs-on: ubuntu-latest

strategy:

matrix:

node-version: [14.x, 16.x]

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Node.js

uses: actions/setup-node@v2

with:

node-version: ${{ matrix.node-version }}

- name: Install dependencies

run: npm install

- name: Run tests

run: npm test

1. **Desktop Application (Electron.js & Vue.js)**

The desktop application requires testing for both the main and renderer processes, along with UI and functional testing for Vue.js components.

1. **Testing Frameworks and Tools**
   1. **Mocha & Chai:** For testing the main process.
   2. **Jest & Vue Test Utils:** For unit testing Vue.js components.
   3. **Spectron:** For end-to-end testing of Electron applications.
   4. **Cypress:** For UI testing and validating user interactions.
2. **Unit Testing**
   1. **Vue.js Component Testing:**
      1. Test individual Vue components using Jest and Vue Test Utils.
      2. Verify component rendering, computed properties, and methods.

Example Vue Component Test:

import { shallowMount } from '@vue/test-utils';

import DeviceList from '@/components/DeviceList.vue';

describe('DeviceList.vue', () => {

it('renders device names', () => {

const devices = [{ name: 'Device 1' }, { name: 'Device 2' }];

const wrapper = shallowMount(DeviceList, {

propsData: { devices }

});

expect(wrapper.text()).toContain('Device 1');

expect(wrapper.text()).toContain('Device 2');

});

});

* 1. **Main Process Testing:**
     1. Use Mocha & Chai to test functionality in the Electron main process, such as window creation and IPC communication.

Example Main Process Test:

const { app, BrowserWindow } = require('electron');

const assert = require('chai').assert;

describe('Main Process', () => {

it('should create a window', () => {

const win = new BrowserWindow ();

assert.isNotNull(win);

});

});

1. **End-to-End Testing**
   1. **Use Spectron for testing the entire Electron application, including main and renderer processes.**
   2. **Test workflows such as user login, data visualization, and device configuration.**

Example End-to-End Test:

const { Application } = require('spectron');

const path = require('path');

const app = new Application ({

path: path.join(\_\_dirname, '..', 'dist\_electron', 'IOTICS.app')

});

describe('IOTICS App', () => {

beforeEach(() => app.start());

afterEach(() => app.stop());

it('opens a window', async () => {

const count = await app.client.getWindowCount();

assert.strictEqual(count, 1);

});

it('displays the correct title', async () => {

const title = await app.client.getTitle();

assert.strictEqual(title, 'IOTICS');

});

});

1. **Test Coverage and Reporting**
   1. **Code Coverage:** Use tools like nyc (for Node.js) and jest --coverage (for Vue.js) to generate test coverage reports.
   2. **Reporting:** Generate test reports and coverage summaries and integrate them into the CI/CD pipeline for review.

Example of generating coverage reports with Jest:

jest --coverage

This will create a coverage directory with detailed reports on test coverage for each file and function in the project.

## License and Authors

This section provides information on the licensing terms under which IOTICS is distributed and acknowledges the contributors who have played a role in its development.

1. **License**

IOTICS is licensed under the MIT License. This license allows you to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, subject to the following conditions:

* 1. **Permission:** Permission is granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software.
  2. **Conditions:** The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
  3. **Disclaimer:** The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the software or the use or other dealings in the software.

A copy of the full MIT License text is provided below:

MIT License

Copyright (c) [2024] [IOTICS]

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "IOTICS"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.

1. **Authors and Contributors**

IOTICS is licensed under the MIT License. This license allows you to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, subject to the following conditions:

* 1. **Project Lead**
     1. **Bakhodir Abdurakhmonov -** CEO / Product manager.
     2. **Eshmirzaev Salokhbek -** Main frontend developer.
     3. **Ahmadjonov Akmal -** Main backend developer.
     4. **Nasyrov Jamshid -** UI/UX designer.
     5. **Abdullaev Azamjon -** Frontend developer.
  2. **Core Contributors**
     1. **Ahmadjonov Akmal -** Main backend developer.
     2. **Eshmirzaev Salokhbek -** Main frontend developer.
  3. **Contact**
     1. **Email:** [b.abduraxmanov.b@gmail.com](mailto:b.abduraxmanov.b@gmail.com)
     2. **Website:** [www.iotics.uz](https://www.iotics.uz)