# Design Document

ClevrBooks app

Accounting App

Based on Node.js and Vue.js

Contents

[Design Document 1](#_Toc86162093)

[Overview 3](#_Toc86162094)

[Development 5](#_Toc86162095)

[Project Configuration 5](#_Toc86162096)

[NODE.js Backend (Server) 5](#_Toc86162097)

[nodemon setup 5](#_Toc86162098)

[eslint 7](#_Toc86162099)

[.gitignore 8](#_Toc86162100)

[Setting environment variables 9](#_Toc86162101)

[Setting the main app.js 9](#_Toc86162102)

[Database 11](#_Toc86162103)

[Database Schema 11](#_Toc86162104)

[Create table migrations 13](#_Toc86162105)

# Overview

The idea behind this project is to eliminate the dependency from the Quicken application. It regularly requires to login to their server and then wants to install mobile application.

It also provides a great incentive to explore many new web development technologies.

The application will use ‘Node.js/Express’ for the backend, and Vue.js for the UI or frontend.

The ‘Node.js/Express’ framework is written in javascript and provides all the necessary tools to act as a backend for the ‘CRUD[[1]](#footnote-1)’ API and database access with sequelize ORM[[2]](#footnote-2).

This will allow to run the application as a standalone package and will be accessible through ‘localhost or IP address’ which doesn’t require virtual host setup and can be used with VPN connected.

The database will be either SQLite3 or hosted on the current Dublin mySQL server and the data will be populated by exporting all Quicken accounts to CVS files, and reimported to SQL from an Excel or Access VBA script.

CVS Files

Quicken

SQLite3

or

mySQL

Access Database

Excel Worksheet

The frontend will use the Vue.js framework, written in javascript.

The Vue.js frontend will be used to build a Single Page Application (SPA), that will provide authentication and access to the backend API.



# Development

The development IDE will be “Visual Studio Code” from Microsoft with the following extensions:

* ESLint
* Vetur
* Git History

All development will us SQLite3 as the main SQL database.

The project will be hosted on GitHub.

## Project Configuration

The first step consists in creating an empty project on the GitHub server. Once the empty project is created, on the Windows local machine, clone the project in a local directory. On Bangor, the folder is C:\Users\vaill\Documents\work\WebProjects\ClevrBooks

Once cloned, edit the README.md file, commit and push to GitHub.

The following sections will describe the development of the server backend based on node.js and express and the fronter user interface based on Vue.js (Vue3)

## NODE.js Backend (Server)

To build the backend server, first make sure that node.js is installed. The version used for this project is v14.18.0

### nodemon setup

The project will also use ‘nodemon’ to monitor code change and restart the server automatically.

>npm install -g nodemon

From the server subdirectory, initialize the project

>npm init

package name: (server)

    version: (1.0.0)

    description: Backend Server for ClevrBooks app

    entry point: (index.js) src/app.js

    test command:

    git repository:

    keywords:

    author: Claude Vaillancourt

    license: (ISC) MIT

    About to write to C:\Users\vaill\Documents\work\WebProjects\ClevrBooks\server\package.json:

    {

        "name": "server",

        "version": "1.0.0",

        "description": "Backend Server for ClevrBooks app",

        "main": "src/app.js",

        "scripts": {

            "test": "echo \"Error: no test specified\" && exit 1"

        },

        "author": "Claude Vaillancourt",

        "license": "MIT"

    }

    Is this OK? (yes)

### eslint

Install ‘eslint’ as a development module

>npm install –save-dev eslint

Then insert the following commands to the “scripts” attribute of ‘package.json’:

“scripts”: {

“start”: “nodemon –verbose”,

“lint” “eslint \*\*/\*.js”,

“init”: “eslint –init”,

“test”: “echo \”Error: no test specified\” && exit 1”

},

We can now run the “init” script to initialize eslint.

>npm run init

The generated .eslintrc.json file should be like the following:

**{**

**"env": {**

**"browser": true,**

**"commonjs": true,**

**"es2021": true**

**},**

**"extends": "eslint:recommended",**

**"parserOptions": {**

**"ecmaVersion": "latest"**

**},**

**"rules": {**

**"indent": [**

**"error",**

**"space"**

**],**

**"linebreak-style": [**

**"error",**

**"windows"**

**],**

**"quotes": [**

**"error",**

**"single"**

**],**

**"semi": [**

**"error",**

**"never"**

**]**

**}**

**}**

### .gitignore

Finally, create a .gitignore file with the following content:

**.DS\_Store**

**node\_modules**

**\*.sqlite**

**.env**

**# Log files**

**npm-debug.log\***

**# Editor directories and files**

**.vscode**

**\*.suo**

**\*.ntvs\***

**\*.njsproj**

**\*.sln**

**\*.sw?**

### Setting environment variables

To allow access to environment variables through the ‘process.env’ variable, first install the ‘dotenv’ package as follow:

>npm install –-save dotenv

Then, create a ‘.env’ file in the project root directory. The file should contain all runtime environment variables such as:

**# Generic**

**PORT=8080**

**# Database Variables**

**DB\_NAME=clevrbooks**

**DB\_USER=clevrcode**

**DB\_PASSWORD=sqlitepassword**

**DIALECT=sqlite**

**HOST=localhost**

**STORAGE=./clevrbooks.sqlite**

### Setting the main app.js

Create a subfolder ‘src’ containing the ‘app.js’ file that will be the main application code file.

The first line of the app.js file should contain the following to load environment variables:

const result = require('dotenv').config()

if (result.error) {

    console.log("Error: Failed to load environment variables...")

    process.exit()

}

We can now commit and push to GitHub.

## Database

The ClevrBooks database will be hosted by SQLite 3. All access and maintenance of migrations will use the sequelize javascript ORM package.

>npm install –-save sequelize

>npm install –g sequelize-cli

First thing is to run ‘sequelize init’ from the server directory. This will create 3 folders, migrations, models and seeders.

>sequelize init

We are now ready to generate the model tables.

### Database Schema

The core of the database will include 3 tables, Users, Accounts and Entries.

After lot of research, it appears like the only way to create the models is by

#### User Model

To generate the ‘Users’ table, run the following command:

>sequelize model:generate –-name User -–attributes name:string, email:string,password:string

The table name will be set to the model’s name ‘pluralized’, also the ‘id’, ‘createdAt’ and ‘updatedAt’ columns will be added automatically.

This will create a ‘user.js’ file under the ‘models’ directory and a XXXXXXXXXXXXXX-create-user.js file under the ‘migrations’ directory.

Those files must be edited manually to specify table associations and set some column options.

#### Account Model

To generate the ‘Accounts’ table, run the following command:

>sequelize model:generate –-name Account -–attributes name:string, description:string,initBalance:float,currentBalance:float,currency:string

#### Entry Model

### Create table migrations

Once all tables are set properly, we can create the database with the following command:

>sequelize

1. CRUD stands for “Create/Read/Update/Delete”. [↑](#footnote-ref-1)
2. ORM stands for Object-Relational Mapping [↑](#footnote-ref-2)