

Lesson 10:

Learn Database

development with Knex

Learn database configuration, migrations, and seeds, with Knex.

What is Knex?

- Knex.js is like a translator. It lets you write requests to your database (like asking for data, or creating new data) in JavaScript, which is easier for many developers to work with. Knex.js then translates these JavaScript commands into SQL
- It's designed to be flexible, portable, and fun to use.



KNEX.JS

Why use Knex?

- ****Unified Query Syntax***: Knex provides a unified syntax to work with different SQL databases. This means that, in general, you can switch databases with minimal changes to your code.
- ****Schema Builder***: Knex also provides an interface to build and modify the database schema, making it easier to manage tables and relationships between them.
- ***Transactions***: Support for transactions allows handling complex operations that should either fully succeed or fully fail, ensuring data integrity.
- ***Connection Pooling***: Knex supports automatic connection pooling, managing the database connections for you for improved performance.
- ****Migration and Seeding***: Knex has built-in tools to help you migrate your database schema to new versions and seed it with data for testing.

Why use Knex?

In summary:

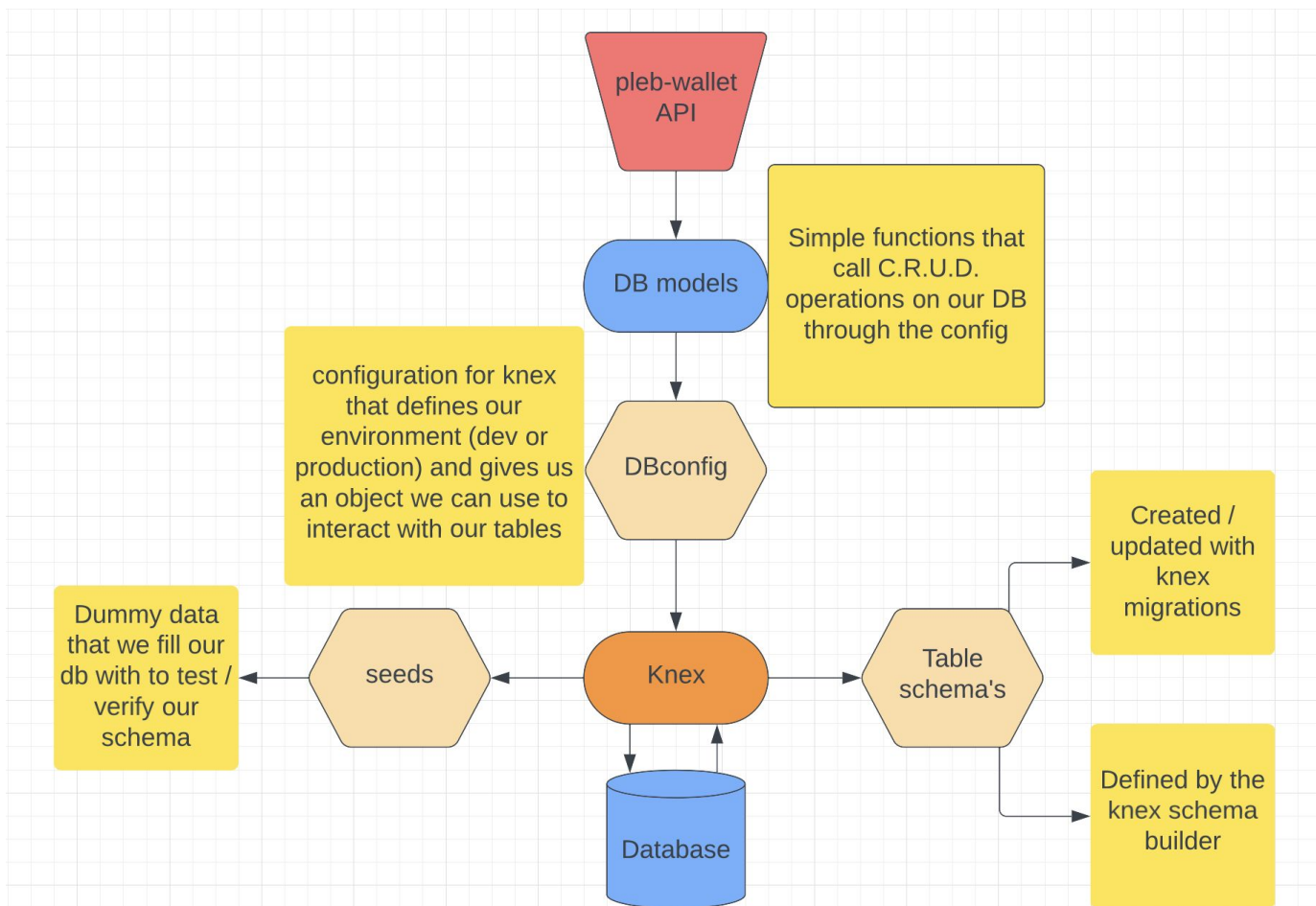
Knex gives a single library with simple local and production configurations that allow us to build our schema once for multiple SQL databases, update that schema in a structured manner with “migrations”, and easily add test data which is called “seeding” or “seeds”.

Knex table vs SQLite table

```
CREATE TABLE users (  
  id INTEGER PRIMARY KEY AUTOINCREMENT,  
  name TEXT NOT NULL  
);
```

```
knex.schema.createTable('users', function(table) {  
  table.increments('id');  
  table.string('name').nullable();  
});
```

Full knex setup from a high level



A few resources upfront:

All of the knex commands we will be using can be found in the knex cheat sheet here - <https://devhints.io/knex>

I also wrote a full walkthrough guide for a simple nodejs backend using knex here -

<https://github.com/AustinKelsay/node-backend-walkthrough>

Install packages

Open up the terminal in pleb-wallet-backend and run the following:

```
npm i sqlite3 pg knex
```

knex: A SQL query builder for Node.js that provides a convenient way to interact with databases.

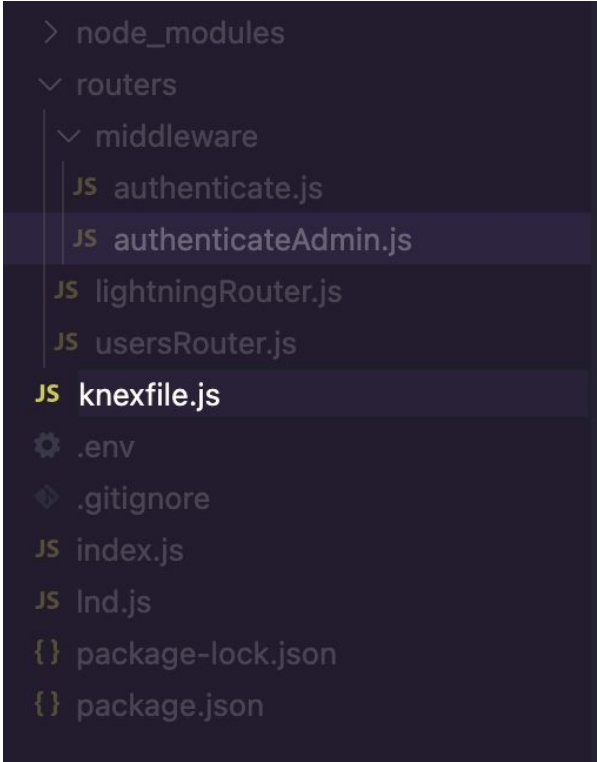
sqlite3: A Node.js module that provides an SQLite database driver for use with Knex.

pg: A PostgreSQL database driver for use with Knex, which enables you to connect and interact with PostgreSQL databases from your Node.js application.

Setting up our database config with settings for a local SQLite db and a production Postgres db

Create knexfile.js for our knex configurations

To use Knex to manage your database, you need to create a knexfile.js file in the root directory of your project. This file will contain the configuration settings for your database, including the connection settings for a local SQLite database we will use for development and a production Postgres database that we will deploy later



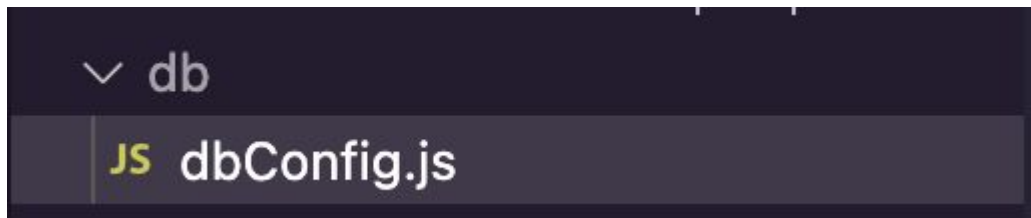
```
> node_modules
  > routers
    > middleware
      JS authenticate.js
      JS authenticateAdmin.js
      JS lightningRouter.js
      JS usersRouter.js
    JS knexfile.js
  .env
  .gitignore
  JS index.js
  JS lnd.js
  {} package-lock.json
  {} package.json
```

Knexfile.js

```
module.exports = {
  development: {
    client: "sqlite3",
    connection: {
      filename: "./db/dev.sqlite3",
    },
    useNullAsDefault: true,
    migrations: {
      directory: "./db/migrations",
    },
    seeds: {
      directory: "./db/seeds",
    },
  },

  production: {
    client: "pg",
    connection: process.env.DATABASE_URL,
    migrations: {
      directory: "./db/migrations",
    },
    seeds: {
      directory: "./db/seeds",
    },
  },
};
```


Create a db directory and dbConfig.js



```
const knex = require("knex");

const config = require("../knexfile");

const env = process.env.NODE_ENV || "development";

const db = knex(config[env]);

module.exports = db;
```

What is a migration?

"Migrations" are like a set of commands or a to-do list for your database. They tell your database how to create, change, or remove tables and columns.

Why Use Migrations?

- Migrations help us keep track of changes we make to the database over time.
- They're especially useful when working with a team, as they allow everyone to see what changes have been made and why.
- Think of them as a version control for your database structure.

How do migrations work with knex?

- Each migration file contains instructions for making a specific change to the database (the up function) and for undoing that change (the down function).
- When you run a migration, it applies the changes from the up function.
- If you need to undo a migration, you can run a command that applies the changes from the down function, reversing the up changes.

```
exports.up = function(knex) {  
  return knex.schema.createTable('users', function(table) {  
    table.increments('id');  
    table.string('name');  
    table.string('email');  
  })  
}  
  
exports.down = function(knex) {  
  return knex.schema.dropTable('users');  
}
```

Create our first migration for “Users”

- In the root of your project open up the terminal and run `npx knex migrate:make create_users_table`
- This will create a new migration file in the db directory (determined by your knexfile config)
- Knex gives us this command to help keep our database updates structured
- We get a timestamp in the name of the file and some boilerplate up and down functions to use

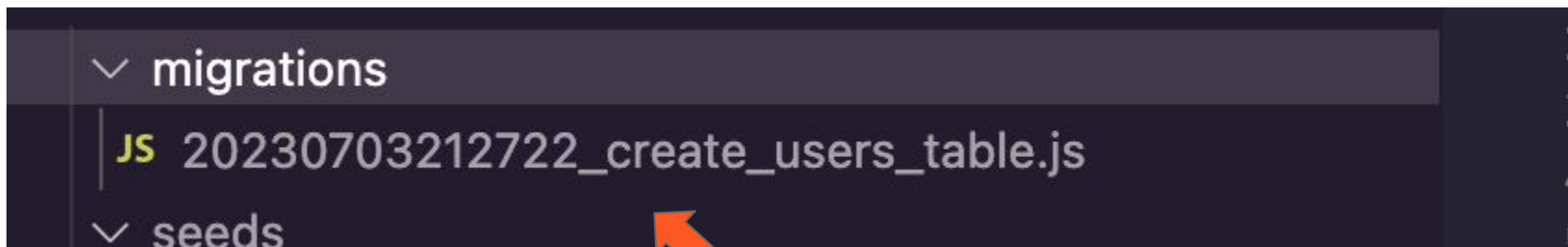


The screenshot shows a code editor with a file explorer on the left and a migration file on the right. The file explorer shows a project structure with a `db` directory containing `migrations` and `dbConfig.js`. The `migrations` directory is expanded, showing a file named `JS 20230703212722_create_users...`. The migration file on the right is `20230703212722_create_users_table.js` and contains the following code:

```
1  /**
2   * @param { import("knex").Knex } knex
3   * @returns { Promise<void> }
4   */
5  exports.up = function(knex) {
6
7  };
8
9  /**
10   * @param { import("knex").Knex } knex
11   * @returns { Promise<void> }
12   */
13  exports.down = function(knex) {
14
15  };
16
```


Now let's add our Users schema

1. Open up the new migration file that was just created with knex
2. Add in the schema code from the next slide



Running our latest migration

Now that we have created this migration and setup the user schema inside we can run a command to “migrate latest” and have our knex run the latest migration files through our database

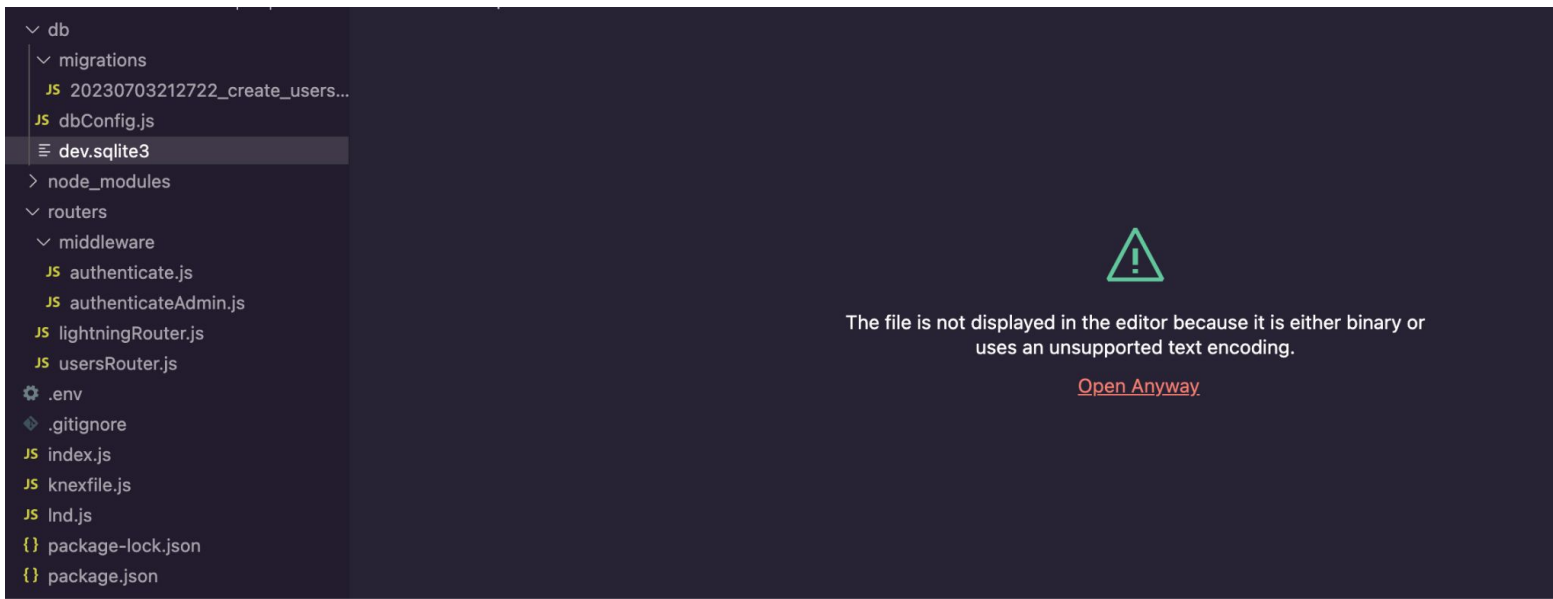
Run `npx knex migrate:latest` in your terminal and you should see this:

```
austinkelsay@Austins-MBP backend-course-walkthrough % npx knex migrate:latest
Using environment: development
Batch 1 run: 1 migrations
austinkelsay@Austins-MBP backend-course-walkthrough %
```

Notice the message “Using environment: development” 👁

Development DB created!

- Now that we have run the latest migration knex automatically generates a local dev.sqlite3 database with our new table inside.
- If you remember in our config we define the name/path for our 'development' db
- There is still no data so nothing to explore yet, but the structure of our table is there and ready to use!



What are seeds?

When working with databases, we often need to add some initial data to the database.

This could be for testing, or to add some default values.

In Knex, we use something called "seeds" to add this initial data.

What are Seeds?

- Seeds are files with code that add data into our database tables.
- They are usually used to populate the database with default or dummy data, which is very useful during development and testing.

Why Use Seeds?

- Seeds can help us to test our application by creating predictable and repeatable data sets.
- They allow us to have a known state of the database to run our tests against.
- Seeds can also be used to populate reference data, like a list of countries or user roles.

Creating a seed file for our “Users” table

Open up your terminal and run `npx knex seed:make 01_users`

- Much like knex migrate:make seed:make will create a new file for us with a bit of boilerplate code to use for creating our test data

NOTE: Seeding your db will erase all previous data which is why we will only be doing it in development for testing purposes



```
1 /**
2  * @param { import("knex").Knex } knex
3  * @returns { Promise<void> }
4  */
5 exports.seed = async function(knex) {
6   // Deletes ALL existing entries
7   await knex('table_name').del()
8   await knex('table_name').insert([
9     {id: 1, colName: 'rowValue1'},
10    {id: 2, colName: 'rowValue2'},
11    {id: 3, colName: 'rowValue3'}
12  ]);
13 };
14
```

Add some test users to our new seed file

```
const bcrypt = require("bcryptjs");

exports.seed = async function (knex) {
  // Deletes ALL existing entries
  await knex("users").del();

  await knex("users").insert([
    {
      id: 1,
      username: "Alice",
      password: bcrypt.hashSync("pass1", 14),
      adminKey: "1234",
    },
    {
      id: 2,
      username: "Bob",
      password: bcrypt.hashSync("pass2", 14),
      adminKey: null,
    },
  ]);
};
```

Running our user seeds

Now that we have created our seed file with two users that represent the exact data we will be expecting we can run it through the database into the users table and then make sure everything looks right.

Run `npx knex seed:run` in your terminal and you should see this:

```
austinkelsay@Austins-MBP backend-course-walkthrough % npx knex seed:run
Using environment: development
Ran 1 seed files
```


Our full knex db setup

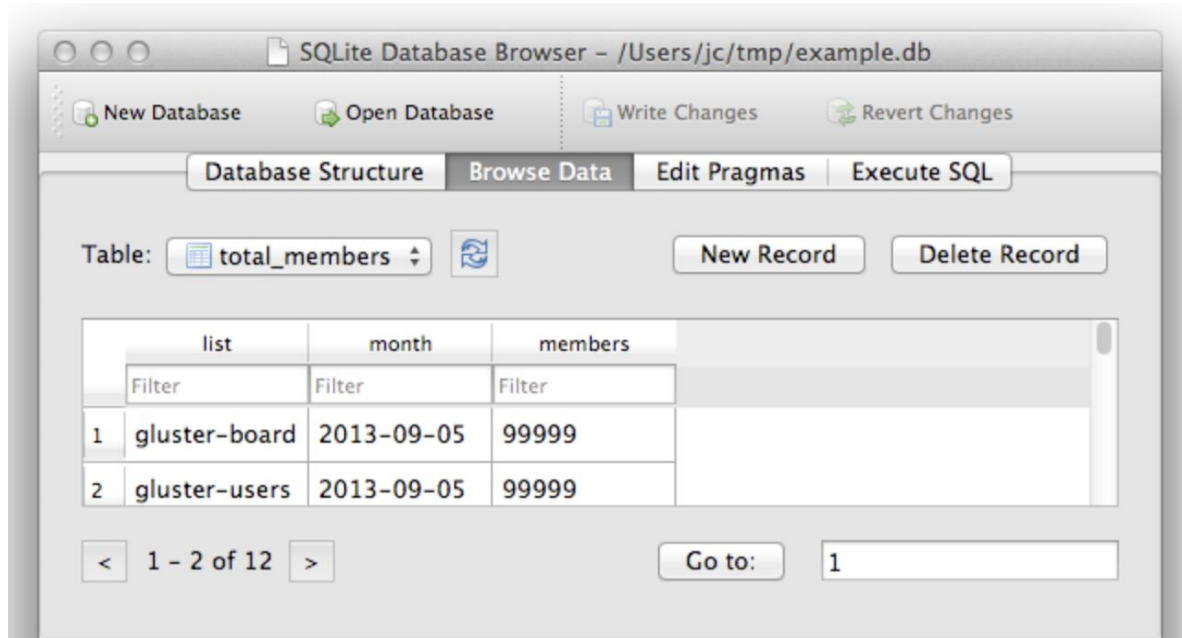
We did it! We went through the entire knex database setup for our backend:

- Creating a new table
- Running a migration for our new table
- Generating our local development db
- Creating a seed file with example data
- Ran that example data through our newly created table into our development database

Now let's actually view our database

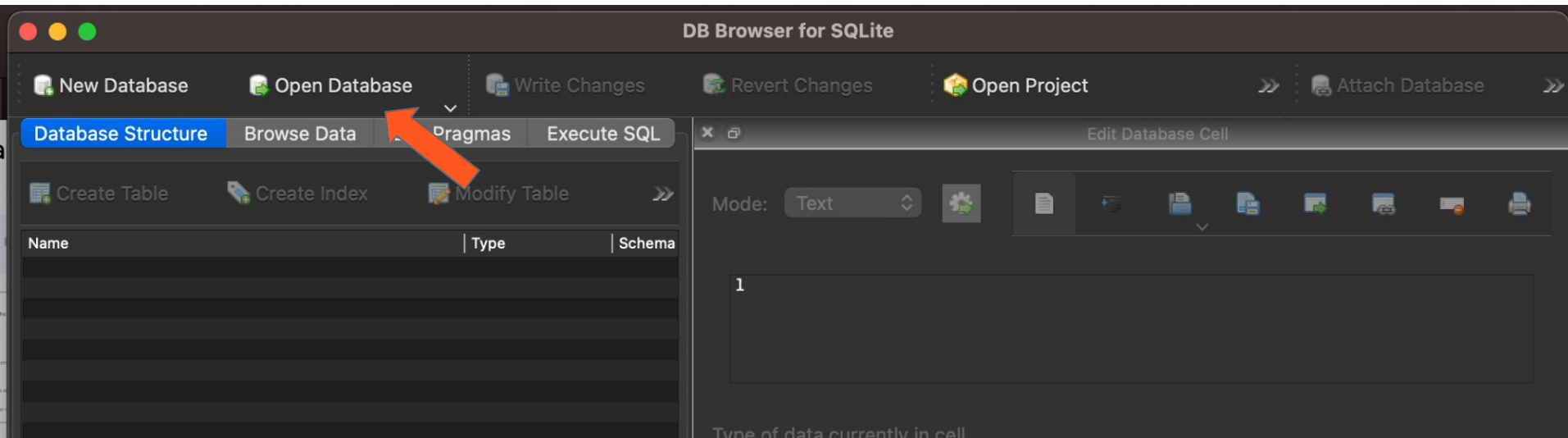
Download sqlite db browser

<https://sqlitebrowser.org/dl/>



Open up DB Browser

Click “Open Database” and navigate to the dev.sqlite3 db we created in our backend repo



View the users table

Click on the “Browse Data” tab and select the users table for the table dropdown

We can see our 2 users with their hashed passwords and adminKey's!

The screenshot shows the DB Browser for SQLite application window. The title bar reads "DB Browser for SQLite - /Users/austinkelsay/Desktop/backend-course". The interface includes a toolbar with buttons for "New Database", "Open Database", "Write Changes", and "Revert Changes". Below the toolbar are four tabs: "Database Structure", "Browse Data" (which is selected and highlighted in blue), "Edit Pragmas", and "Execute SQL". In the "Browse Data" tab, a dropdown menu labeled "Table:" shows "users" selected. To the right of the dropdown are several icons for refreshing, filtering, and other table operations. Further right is a "Filter i..." input field. The main area displays a table with the following data:

	id	username	password	adminKey
	...	Filter	Filter	Filter
1	1	Alice	\$2a\$14\$SIq9AaudxS2GCQYmGQdH7.LhiF91b....	420
2	2	Bob	\$2a\$14\$1BweaFW2s2E6R9yTKP8ho....	1

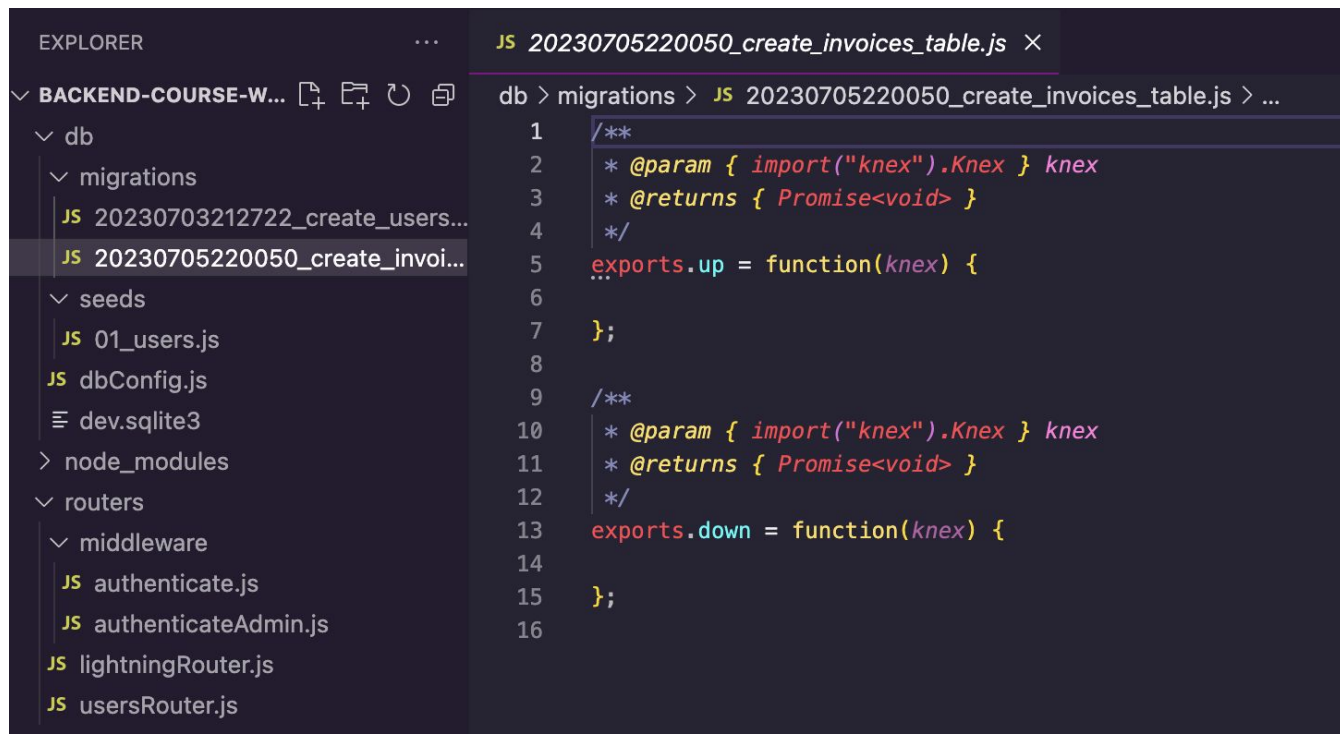
Process for updating our db with knex

We have gone through an initial setup creating our first schema, generating our dev db, and running some seeds through it but now we need to add our Invoices table which will require an update to our current database.

All we really need to do is run a new migration, though when doing so we should always be thoughtful and careful about what affects our changes might have on the current structure of the database and the backend in general.

Create a new migration for Invoices

- In the root of your project open up the terminal and run `npx knex migrate:make create_invoices_table`



Add our knex Invoices schema pt-1

```
exports.up = function (knex) {  
  // Begin creating a new table named "invoices"  
  return knex.schema.createTable("invoices", function (table) {  
    // Creates a primary key column named "id", which will auto-increment its value for each new record  
    table.increments("id").primary();  
  
    // Creates a string column named "payment_request" that cannot be null and must be unique across all  
    records  
    table.string("payment_request").nullable().unique();  
  
    // Creates an integer column named "value" that cannot be null  
    table.integer("value").nullable();  
  
    // Creates a string column named "memo" for additional notes or comments  
    table.string("memo");  
  
    // Creates an integer column named "fees"  
    table.integer("fees");  
  });  
}
```

Add our knex Invoices schema pt-2

```
// Creates a boolean column named "send" that cannot be null
table.boolean("send").nullable();

// Creates a boolean column named "settled" that cannot be null
table.boolean("settled").nullable();

// Creates a timestamp column named "settle_date" with a default value of 0
table.timestamp("settle_date").defaultTo(0);

// Creates a timestamp column named "created_at" that defaults to the current time
table.timestamp("created_at").defaultTo(knex.fn.now());

// Creates an integer column named "user_id" that cannot be null
table.integer("user_id").unsigned().nullable();

// This sets up a foreign key constraint, where "user_id" in the "invoices" table references the "id" column in
the "users" table
table.foreign("user_id").references("id").inTable("users");

});

};
```


Add our knex Invoices schema pt-3

```
exports.down = function (knex) {  
  // This function will drop the "invoices" table, undoing the effects of the `up` function  
  // This is useful for resetting the database to its prior state if something goes wrong  
  return knex.schema.dropTableIfExists("invoices");  
};
```

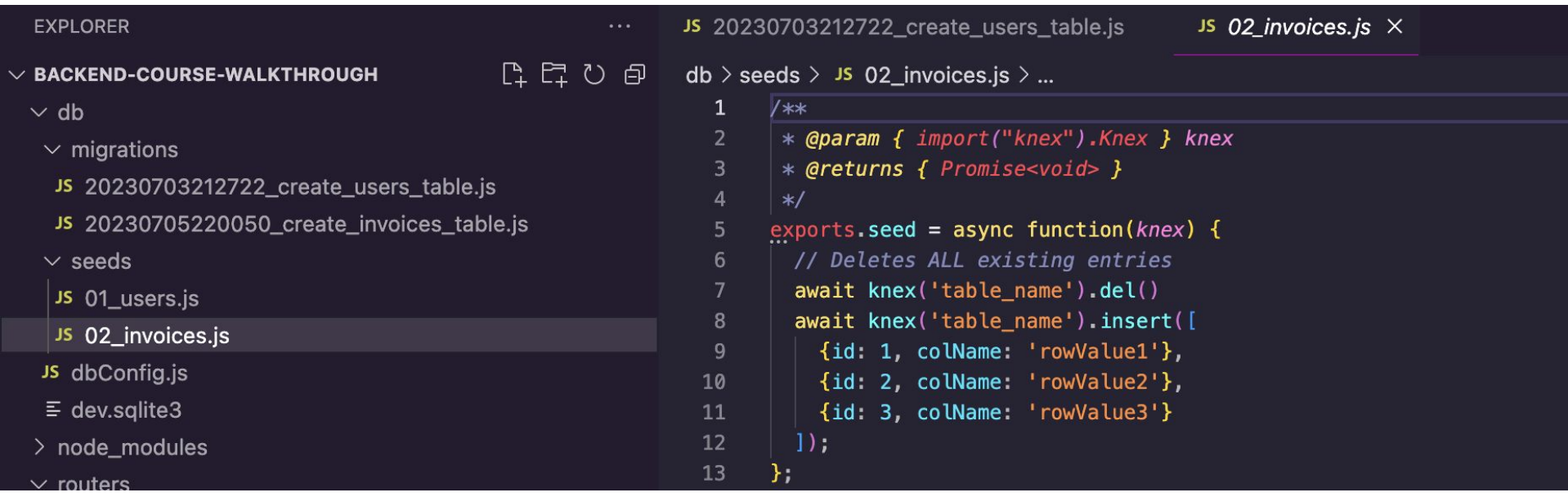

Run the latest migration

Run `npx knex migrate:latest` in your terminal

```
austinkelsay@Austins-MBP backend-course-walkthrough % npx knex migrate:latest
Using environment: development
Batch 2 run: 1 migrations
austinkelsay@Austins-MBP backend-course-walkthrough %
```

Now let's add some Invoice seeds

run `npx knex seed:make 02_invoices`



EXPLORER

✓ BACKEND-COURSE-WALKTHROUGH

- db
 - migrations
 - 20230703212722_create_users_table.js
 - 20230705220050_create_invoices_table.js
 - seeds
 - 01_users.js
 - 02_invoices.js
 - dbConfig.js
 - dev.sqlite3
 - node_modules
 - routers

db > seeds > 02_invoices.js > ...

```
1 /**
2  * @param { import("knex").Knex } knex
3  * @returns { Promise<void> }
4  */
5 exports.seed = async function(knex) {
6   // Deletes ALL existing entries
7   await knex('table_name').del()
8   await knex('table_name').insert([
9     {id: 1, colName: 'rowValue1'},
10    {id: 2, colName: 'rowValue2'},
11    {id: 3, colName: 'rowValue3'}
12  ]);
13 };
```

Add some invoices seeds

```
exports.seed = async function (knex) {  
  // Deletes ALL existing entries  
  await knex("invoices").del();  
  await knex("invoices").insert([  
    {  
      payment_request: "lnbcrt1ulp...",  
      memo: "yo",  
      value: 100,  
      settled: false,  
      send: false,  
      user_id: 1,  
    },  
    {  
      payment_request: "lnbcrt2ulp...",  
      value: 100,  
      fees: 10,  
      send: true,  
      settled: true,  
      settle_date: knex.fn.now(),  
      user_id: 2,  
    },  
  ]);  
};
```

Run invoice seeds

Run `npx knex seed:run`

```
austinkelsay@Austins-MBP backend-course-walkthrough % npx knex seed:run
Using environment: development
Ran 2 seed files
austinkelsay@Austins-MBP backend-course-walkthrough %
```

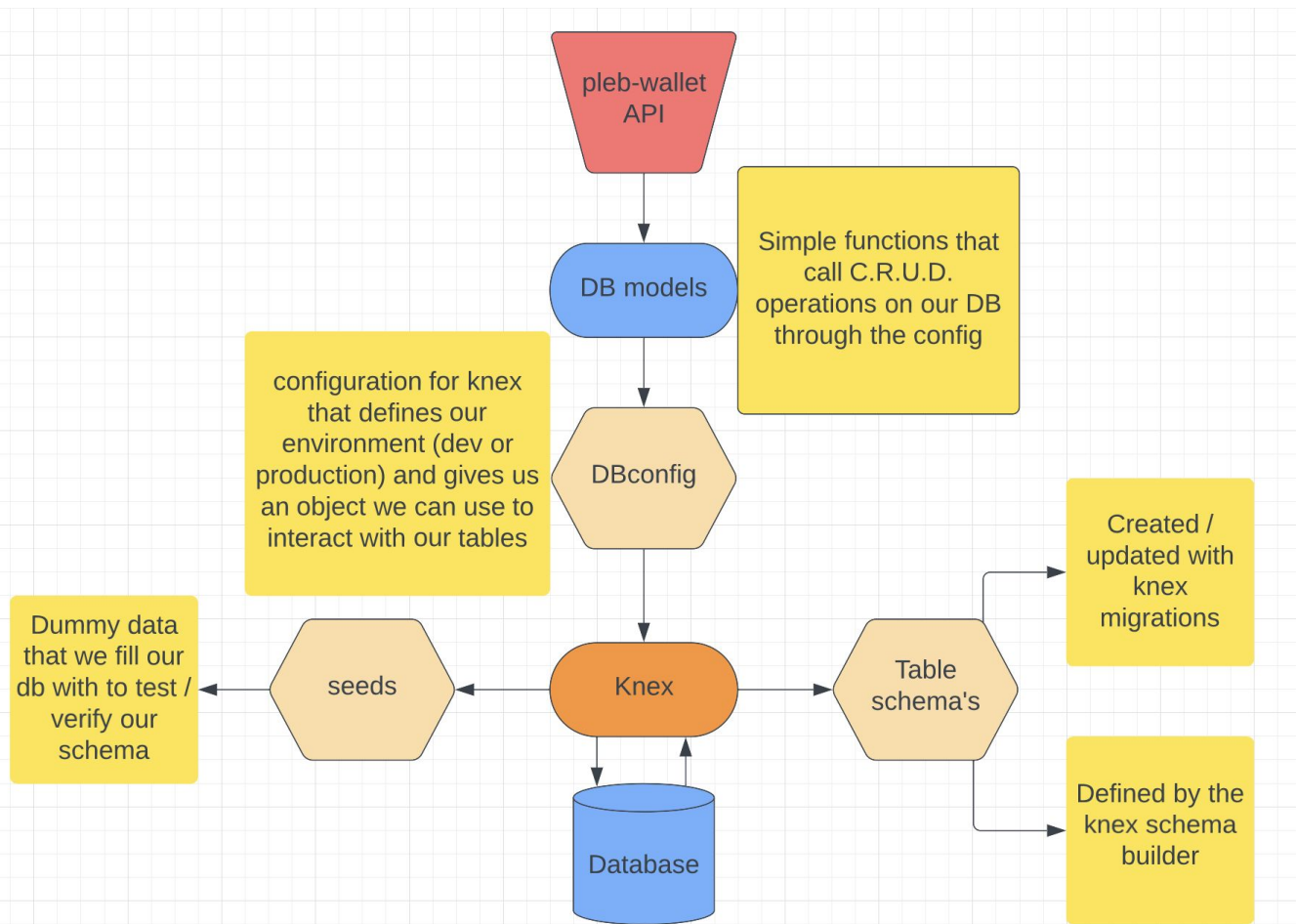
Now let's check out our development DB one more time

Looks good on my side!

The screenshot shows the 'DB Browser for SQLite' application window. The title bar indicates the database path is '/Users/austinkelsay/Desktop/backend-course-w'. The interface includes a toolbar with buttons for 'New Database', 'Open Database', 'Write Changes', 'Revert Changes', and 'Open'. Below the toolbar are tabs for 'Database Structure', 'Browse Data' (which is selected), 'Edit Pragmas', and 'Execute SQL'. The 'Table:' dropdown is set to 'invoices'. A toolbar below the dropdown contains icons for refreshing, filtering, and other table actions. The main area displays a table with the following data:

	id	payment_request	value	memo	fees	send	settled	settle_date	created_at	user_id
...	Filter		Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Inbcrt1u1p...	100	yo	NULL	0	0	NULL	2023-07-05 22:23:58	1
2	2	Inbcrt2u1p...	100	NULL	10	1	1	2023-07-05 22:23:58	2023-07-05 22:23:58	2

Review / reflection



Resources

- Knex Cheatsheet (Guide) - <https://devhints.io/knex>
- Node backend walkthrough (guide / template) - <https://github.com/AustinKelsay/node-backend-walkthrough>
- Learn Knex JS With Postgres (video) - <https://www.youtube.com/watch?v=wfrn21E2NaU>
- Knex JS Tutorial for beginners (article) - <https://blog.shahednasser.com/knex-js-tutorial-for-beginners/>
- What are database migrations (article) - <https://www.prisma.io/dataguide/types/relational/what-are-database-migrations#state-vs-change-based-migrations>
- Database seeding with knex (article) - <https://dev.to/cesareferrari/database-seeding-with-knex-51gf>