Alex Bortoc

CSCI 49201

Project Report

Original project idea for Bookshelf was to create a website that used a PostgreSQL database to store and retrieve data. However, that turned out to be more complicated than I imagined to do in Ocaml. As a result, website portion of the project was not completed and instead the project compiles several executable files that interface with the database and perform functions such as: create predefined tables, delete these tables, add and remove individual objects(books), truncate tables, and lastly retrieve all objects(books) that are currently stored. A function to update the objects in the database was initially planned to be implemented, however even with just 3 columns that would mean creating 7 functions that update some combination of the columns and repeats a lot of code. In the end, the project became rather limited in scope, but nonetheless exposed me to several concepts in Ocaml that were not covered in class.

The project is built using Caqti library. The library provides an interface to connect relational databases with Ocaml. It has drivers for three relational databases: MariaDB, PostgreSQL, SQLite3. Caqti itself uses Lwt library, which provides threading to it. Lwt works based on promises. Promise is just as it sounds a guarantee that something will happen in time. As per official documentation “a promise is just a reference that will be filled asynchronously.” (https://ocsigen.org/lwt/4.1.0/manual/manual) In my project, Caqti sets up a local connection to the PostgreSQL database with pool of maximum number of connections. Each function is a connection call to the database that technically can be run asynchronously in theory, in theory because as of right now each function is in a separate executable file and thus are run one at a time. As mentioned above, Caqti uses Lwt promises and thus each function is a promise. A nice

thing about promises is that the syntax for different types is nearly identical, with the exception of actually specifying the type.

At the same time, along with being a powerful tool Lwt is rather complicated to learn and understand. I took many wrong turns in the process of developing the project. For example, a function that is used to retrieve the objects from the database took me some time to figure out. The function returns a list of records and I thought that the module Lwt_result was going to be the thing to use to hold the list, given that the function definition is “val get_all_read: unit -> (book list, error) result Lwt.t and according to the Lwt manual Lwt_result “provides helpers for values of type ('a, 'b) result Lwt.t.” (https://ocsigen.org/lwt/4.1.0/api/Lwt_result) One of the helpers is called return that seemed to fit my criteria, however it turned out that the main module Lwt also has a helper called return, which was the actual helper that I needed. Thus. most of my struggles came down to figuring out Lwt. What lead me to use Caqti and try to figure out Lwt was that in the beginning of the project I ran into problems trying to use PG’Ocaml library, which was my initial choice for a PostgreSQL library. My problems were with one of its dependencies – campl4, a parser that according to the developers of the library allows one to write SQL statements directly in the code. For reasons that I could not figure out, I could not make it work and as it turned out it is also no longer being actively maintained.

One big issue with the project at the moment is that separate executables are not a feasible solution, there needs to be an interface, either console or web. Console is not a good choice because setting up a database for a project that works on console is not user friendly, but at least it would allow to run functions from a single place. The original intention of web interface is better as it would allow to obfuscate database setup on the backend and create a single place to interact with the database.