**ET4132 Final Project Report**

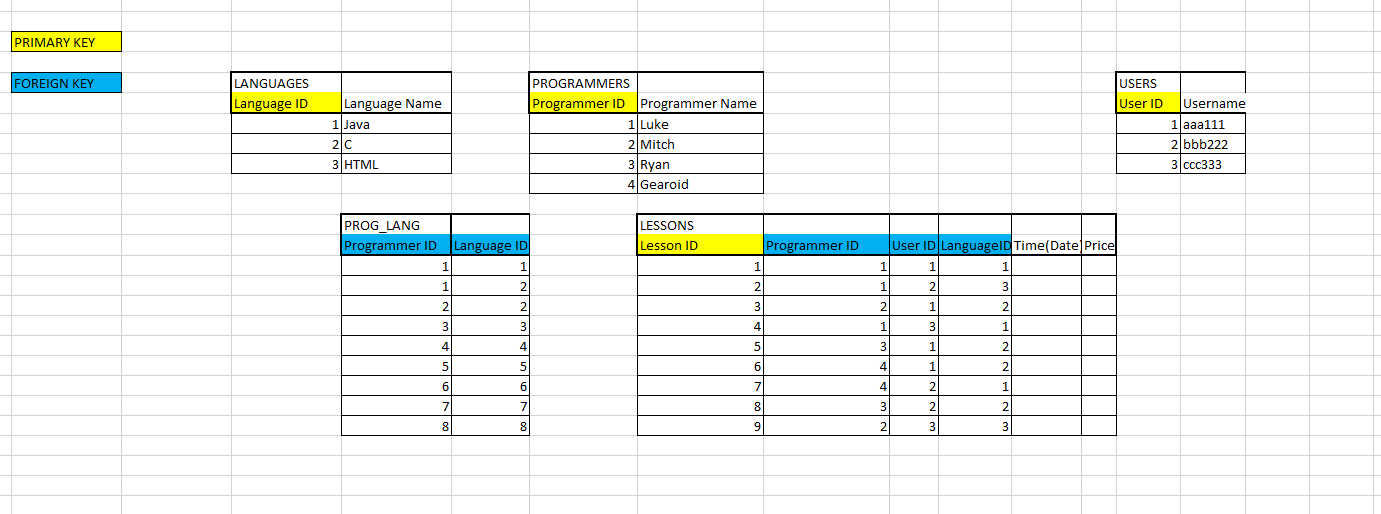
17184614 Luke O’Sullivan Griffin & 17212359 Mike Vriesema

**Introduction & Brief Rationale:**

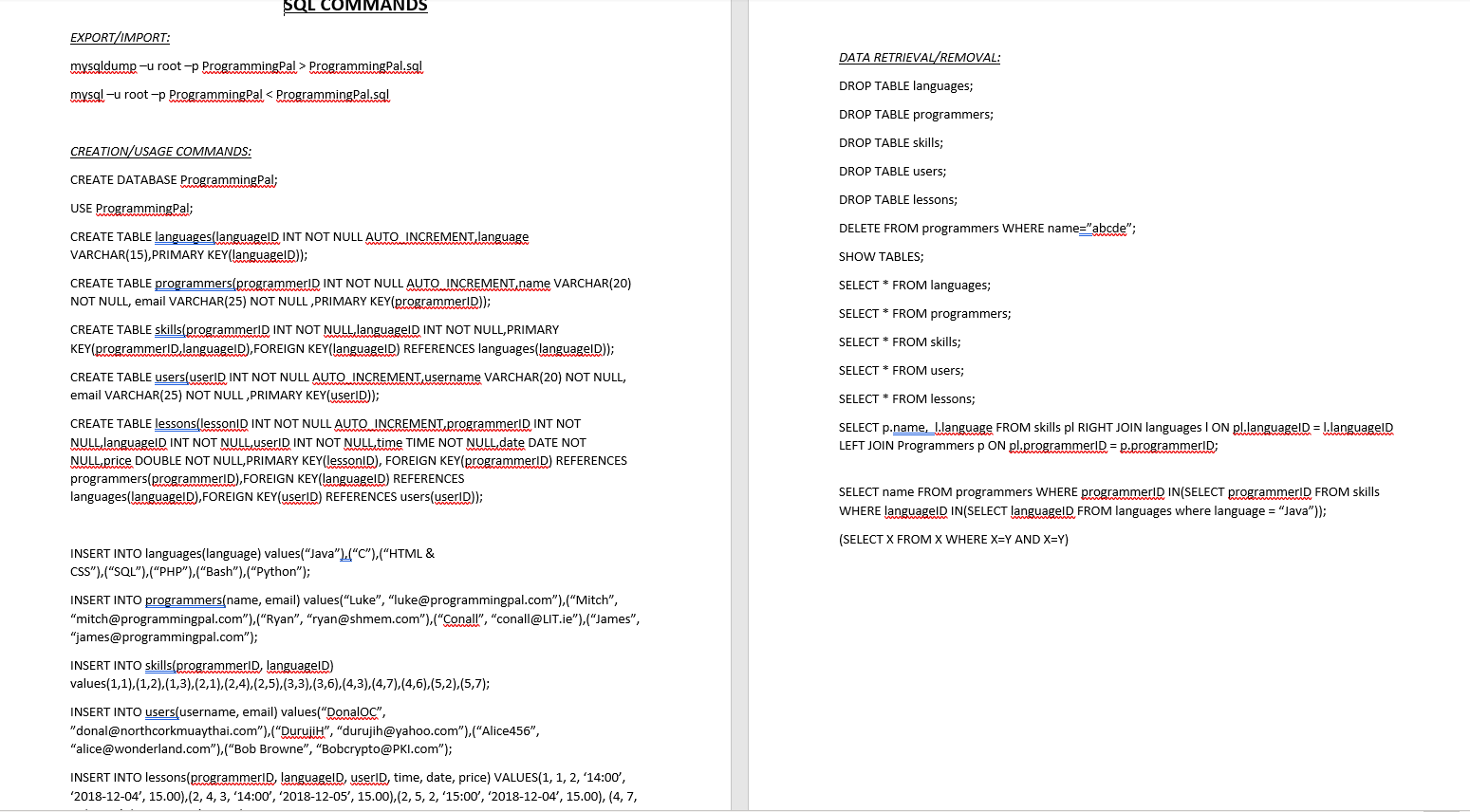
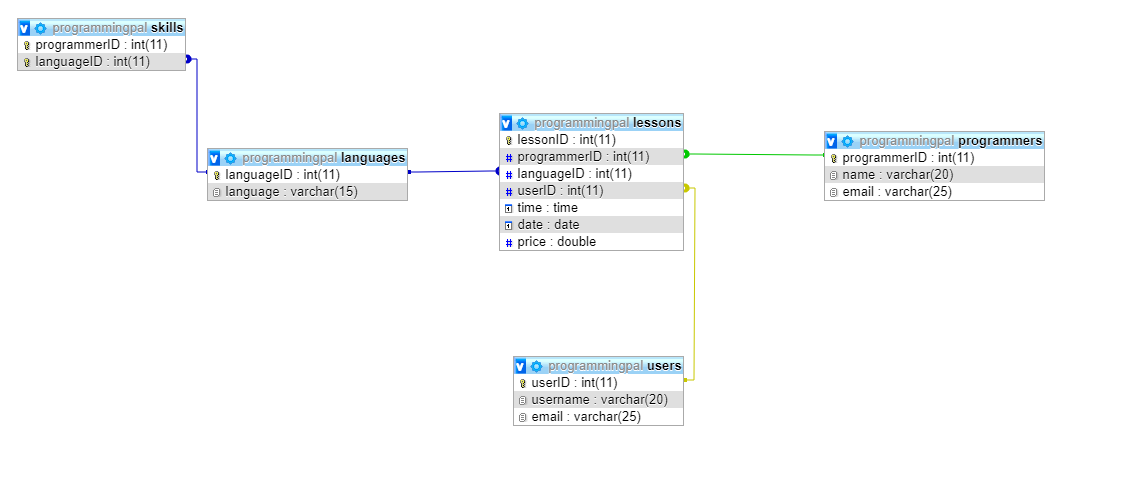
We chose the concept of a programming pal hire/teaching system as the fundamental base for our company.

The reasoning behind this choice is it is something we are both interested in and would hypothetically be a very useful service to struggling computer science students.

The company would provide services to allow you to hire a programmer in a language at a set rate to design an application or hire a programmer to assist in the teaching and learning of new skills in a chosen programming language or from scratch.

**Database Modelling & Design**

We started drafting a possible implementation of the database structure in SQL and immediately wrote all the commands to build it into a document to be tested.

This base set of commands allowed us to quickly rebuild the database during development and adjust table construction before having to use the alter command to modify it once the design was finalised.

This was the finished design and implementation of the database for the project. We adapted the data to be entered it and set in some users to test with.

**Database Implementation: SQL**

We implemented 5 tables into our database and populated them with pre-set data using the following commands:

* CREATE DATABASE ProgrammingPal;
* USE ProgrammingPal;
* CREATE TABLE languages(languageID INT NOT NULL AUTO\_INCREMENT,language VARCHAR(15),PRIMARY KEY(languageID));
* CREATE TABLE programmers(programmerID INT NOT NULL AUTO\_INCREMENT,name VARCHAR(20) NOT NULL, email VARCHAR(25) NOT NULL ,PRIMARY KEY(programmerID));
* CREATE TABLE skills(programmerID INT NOT NULL,languageID INT NOT NULL,PRIMARY KEY(programmerID,languageID),FOREIGN KEY(languageID) REFERENCES languages(languageID));
* CREATE TABLE users(userID INT NOT NULL AUTO\_INCREMENT,username VARCHAR(20) NOT NULL, email VARCHAR(25) NOT NULL ,PRIMARY KEY(userID));
* CREATE TABLE lessons(lessonID INT NOT NULL AUTO\_INCREMENT,programmerID INT NOT NULL,languageID INT NOT NULL,userID INT NOT NULL,time TIME NOT NULL,date DATE NOT NULL,price DOUBLE NOT NULL,PRIMARY KEY(lessonID), FOREIGN KEY(programmerID) REFERENCES programmers(programmerID),FOREIGN KEY(languageID) REFERENCES languages(languageID),FOREIGN KEY(userID) REFERENCES users(userID));

The primary keys are languageID for languages,userID for users,programmerID for programmers,lessonID for lessons and a composite key of (languageID,programmerID) for skills. In lessons userID,langaugeID and programmerID are referenced as foreign key constraints to ensure any lesson values inputted are valid entries in their own corresponding tables.

The following SQL commands are used to fill the database with the preset values for languages and the default team of programmers and their associated skillset.

* INSERT INTO languages(language) values(“Java”),(“C”),(“HTML & CSS”),(“SQL”),(“PHP”),(“Bash”),(“Python”);
* INSERT INTO programmers(name, email) values(“Luke”, “luke@programmingpal.com”),(“Mitch”, “mitch@programmingpal.com”),(“Ryan”, “ryan@shmem.com”),(“Conall”, “conall@LIT.ie”),(“James”, “james@programmingpal.com”);
* INSERT INTO skills(programmerID, languageID) values(1,1),(1,2),(1,3),(2,1),(2,4),(2,5),(3,3),(3,6),(4,3),(4,7),(4,6),(5,2),(5,7);

This creates the finalised database used in our website minus the test user accounts and lesson entries for display purposes.

**Website Design & Development: HTML/CSS**

We decided to go for a simple design approach with minimalistic colours (shades of grey, white and blue).

The header consists of a h1 heading and a subscript for smaller text.

The nav bar is located underneath with customised links to each webpage.

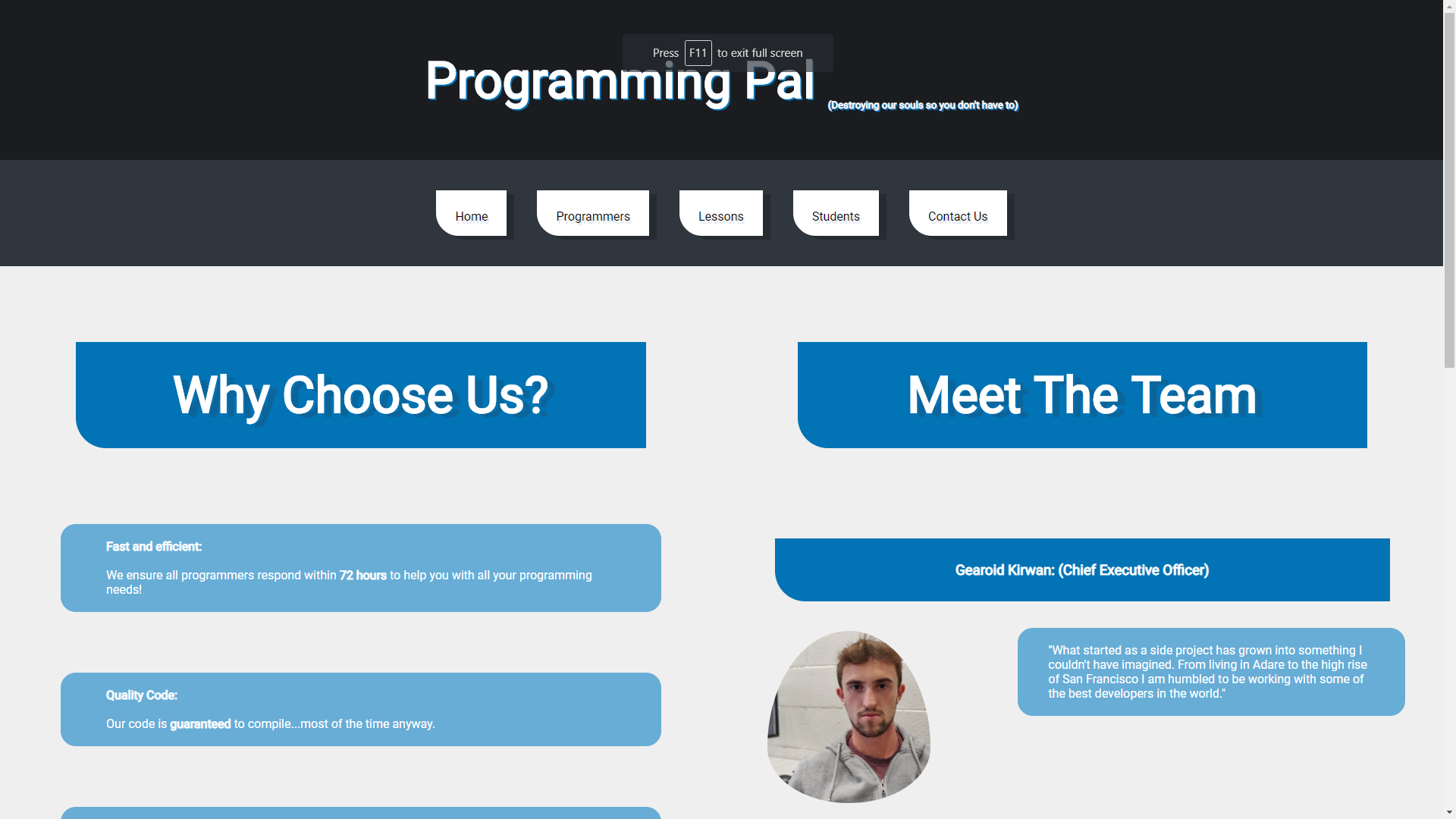
The main body is split into two divs (content and content 2), allowing us to easily place content on either side of the page. The divs are identified in the stylesheet using ID’s and Classes where appropriate.

A footer [1.1] is included which includes the address, a mailto link and links to other pages (for ease of access).

Two stylesheets were created: indexStyle.css and formStyle.css. indexStyle is used as the main stylesheet. We decided to create a second one for pages with forms as it reduced clutter and made it far easier to adjust certain elements without affecting other pages without forms.

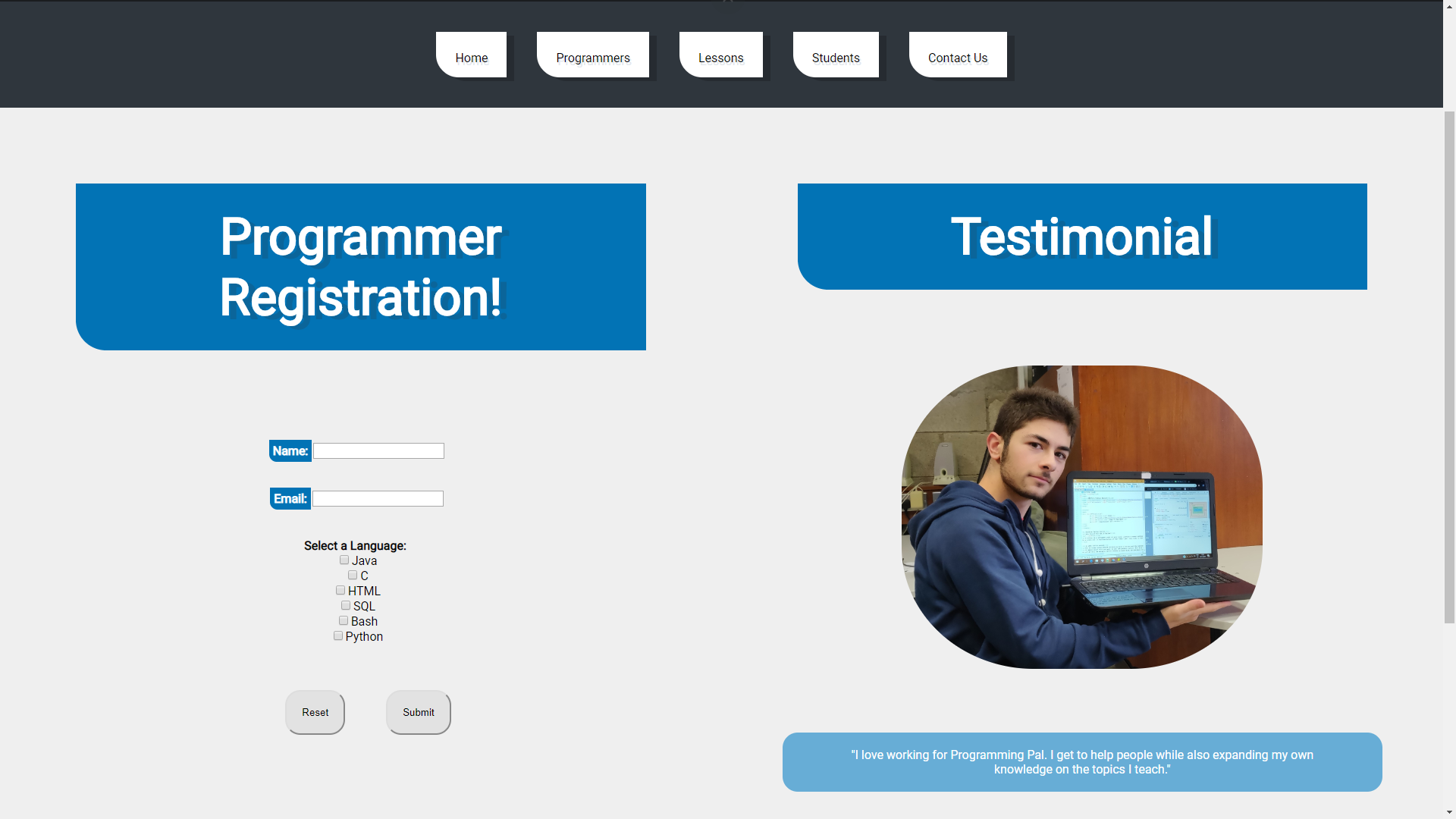
The lessons database information is returned into a table (using echo in php) it is then formatted with CSS and displayed to the end user [1.2].

<http://localhost/project/index.php>

Index.php

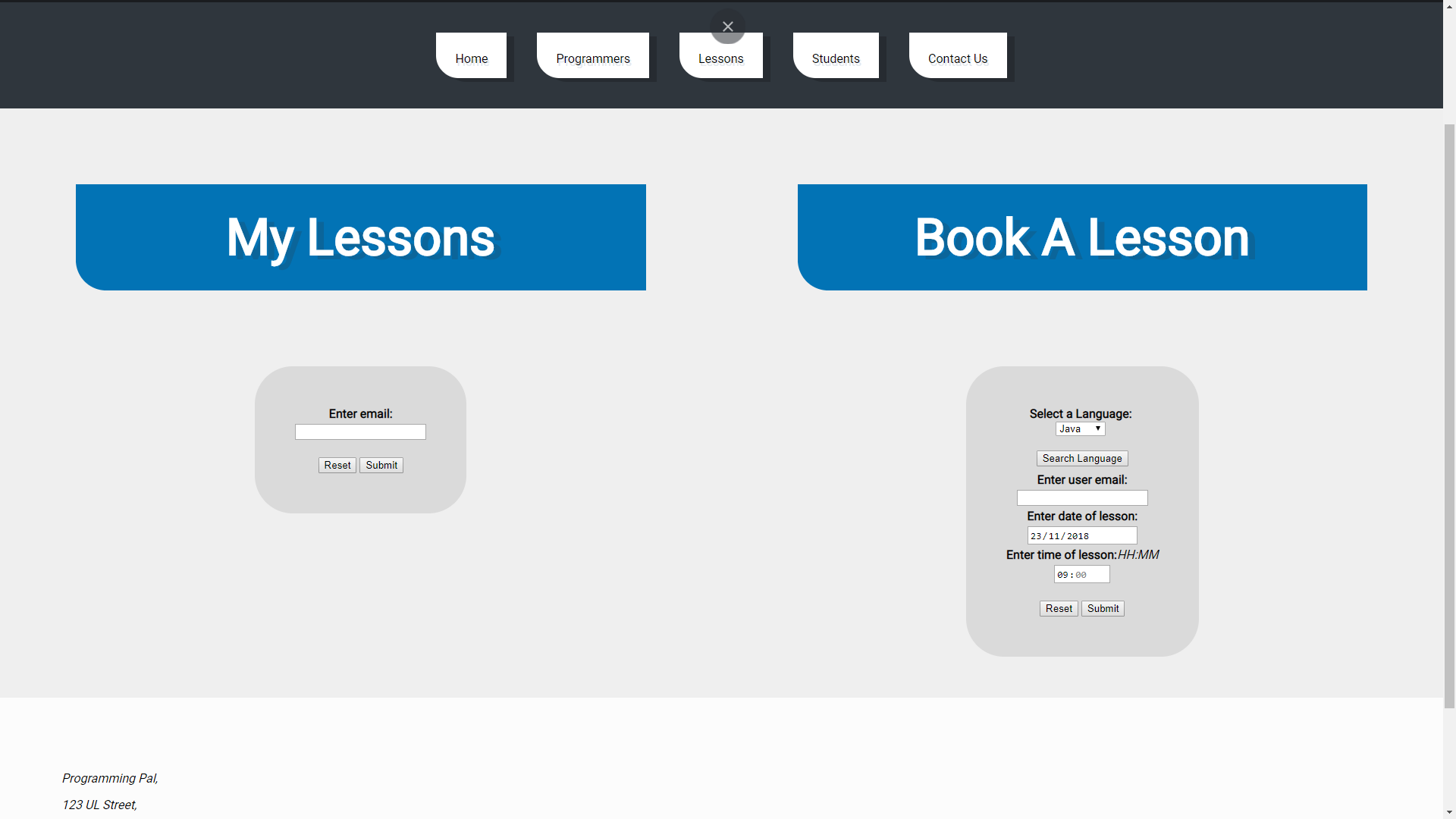
<http://localhost/project/studentForm.php>

Form.php



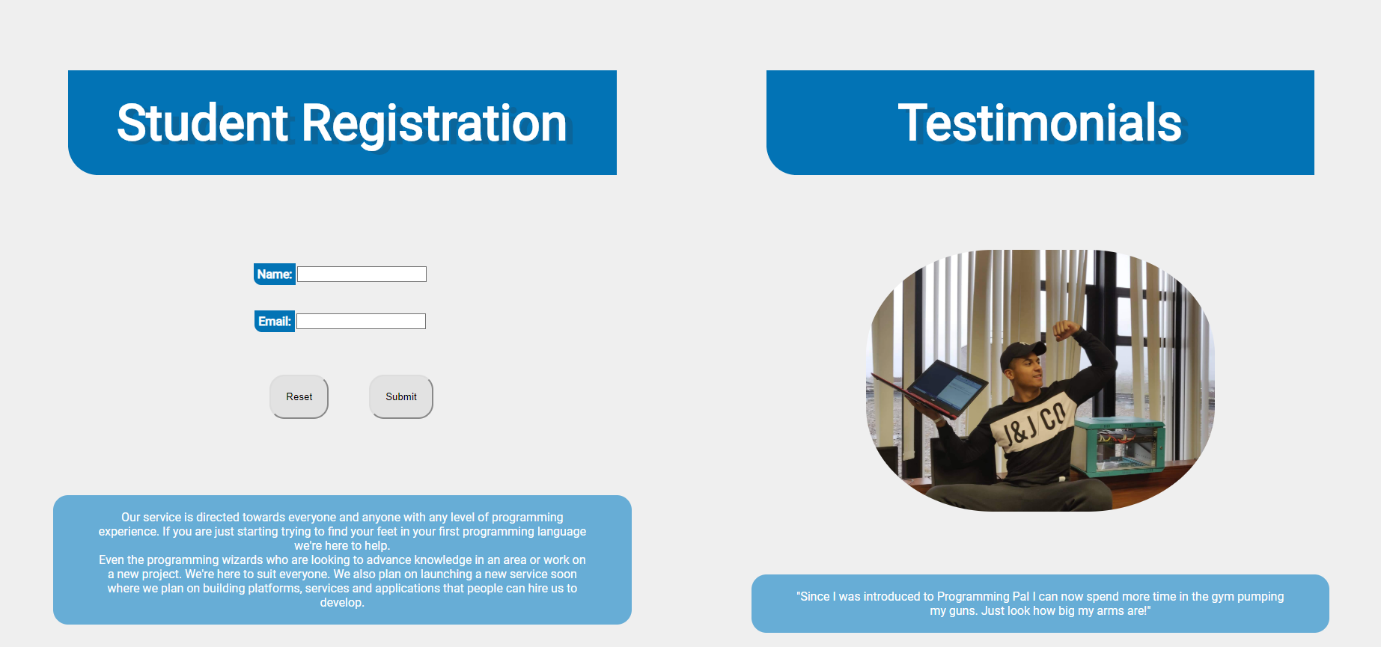
<http://localhost/project/lessons.php>

Lessons.php



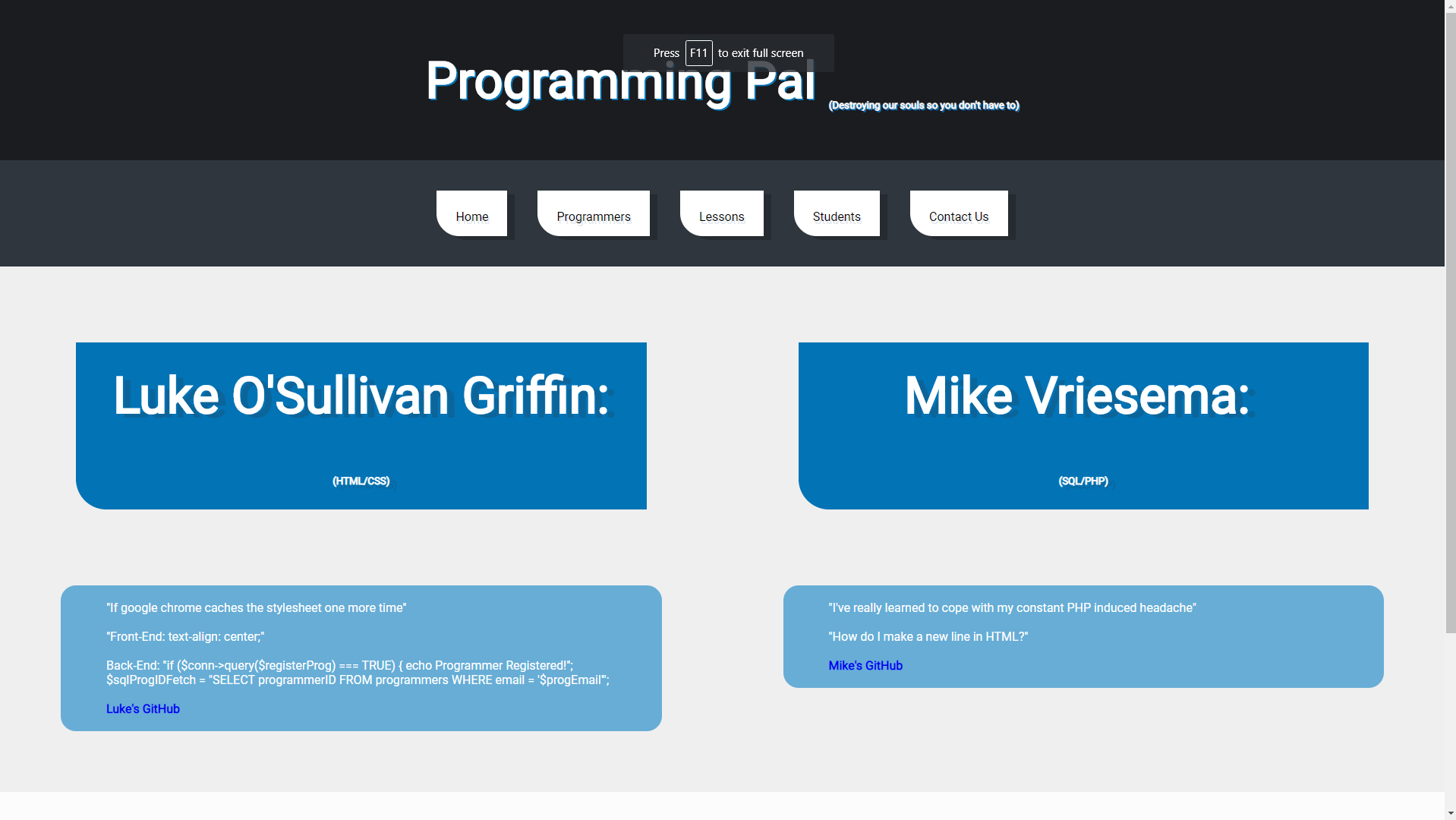
<http://localhost/project/studentForm.php>

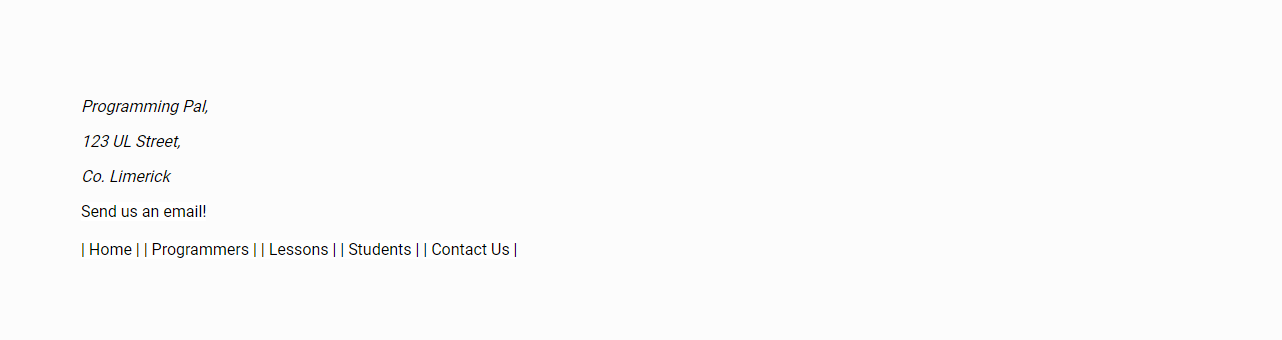
StudentForm.php



<http://localhost/project/contact.php>

Contact.php

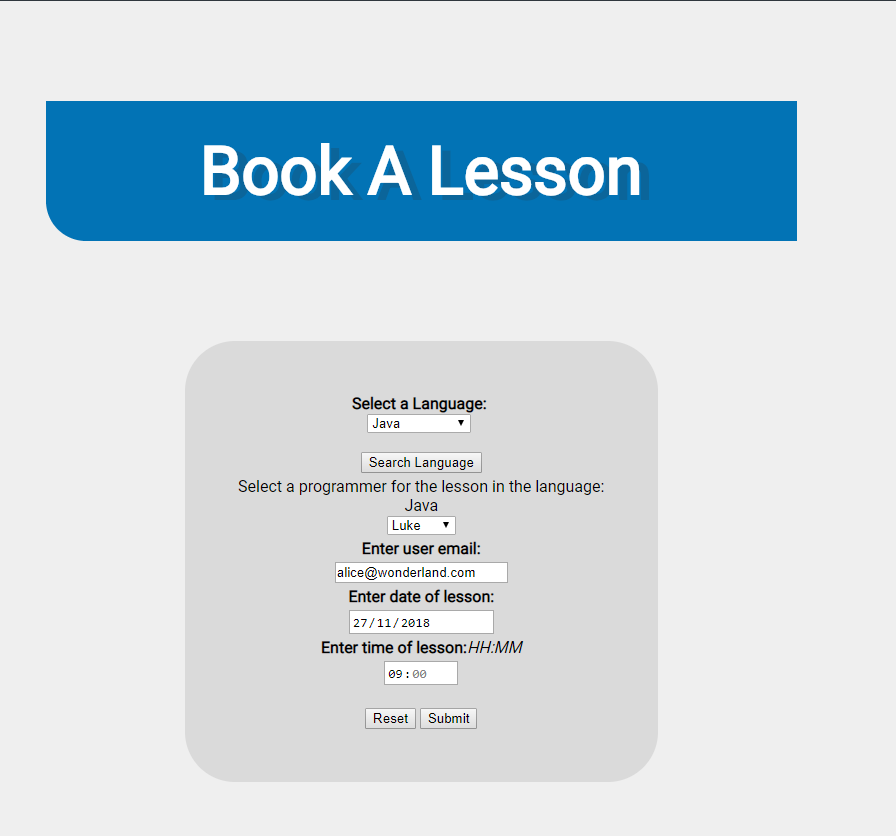


Footer [1.1] 

Lessons Table [1.2]

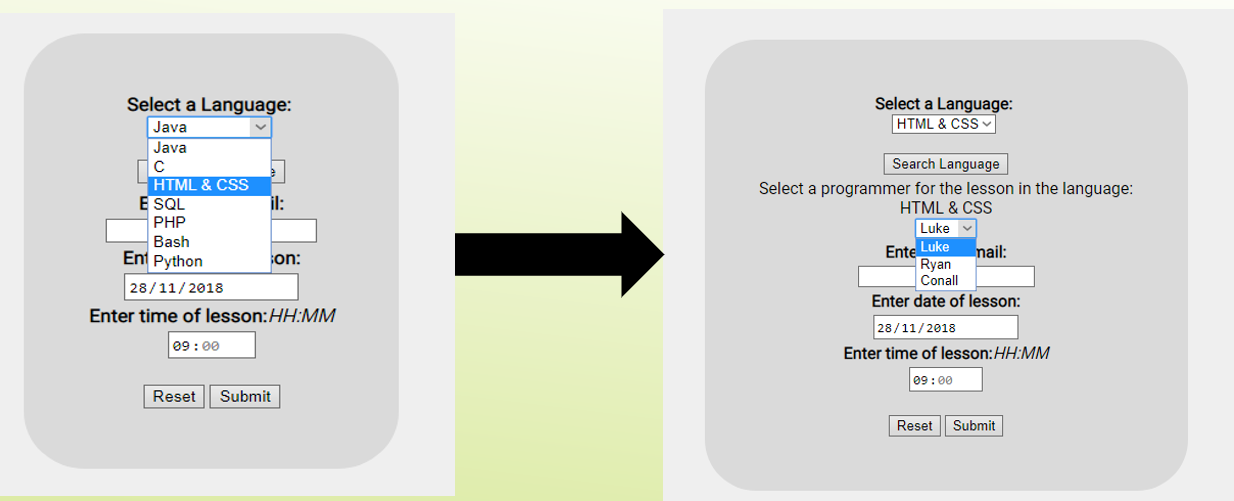
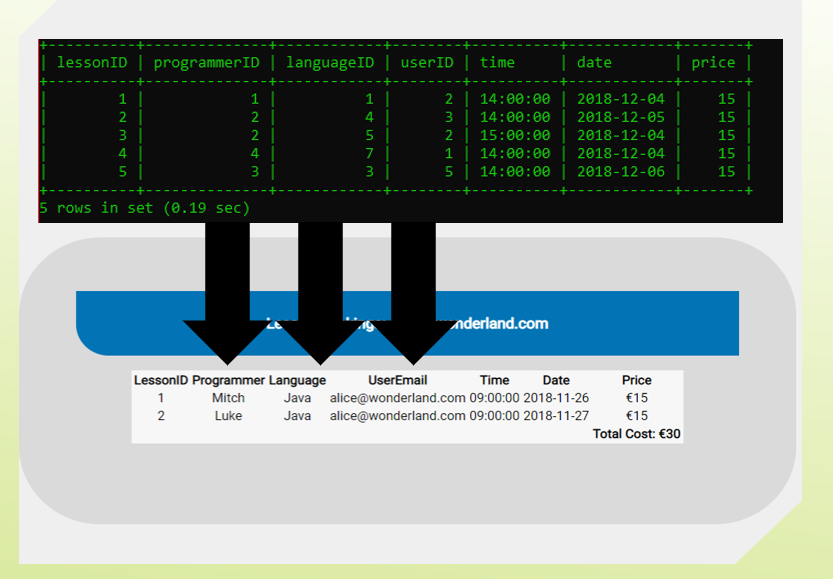


Book a Lesson [1.3]

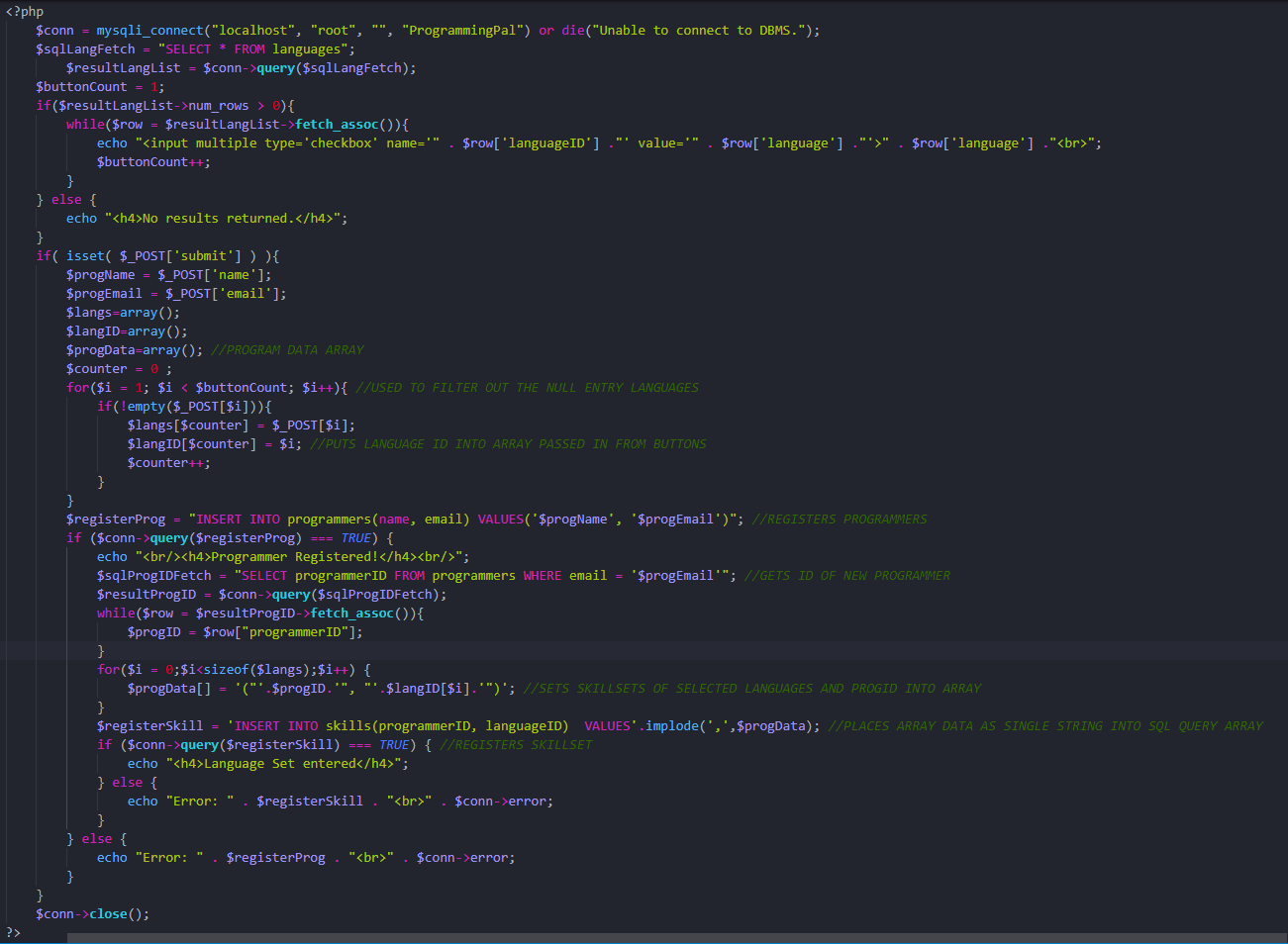


**Website & Database Connections: PHP**

Extensive PHP is used in the 3 form pages to generate dropdowns and populate radio buttons with dynamic values from the database. We also use a variety of fetches to retrieve relevant info from other tables when returning a form to make the output more user friendly. Examples of this is in lessons where we pull the lesson data and fetch the programmer and language from their ID values in their corresponding tables.

****

Significant challenges included the programmer registration where the dynamically filled checkboxes get sent to the form with null values if not selected. Key points here are that we may not know how many languages there are total (unless hardcoded) and we do not know how many will be selected by the programmer. To overcome this, we use a variable to keep track of how many languages enter the checkboxes. We also use this to take their ID values also. When the form is submitted, we use a for loop to filter out null values from the post array and carry the selected languages’ IDs into a separate array for processing later. We then register the programmer using the inputted email and name and retrieve the new ID for the programmer. Then we are able to build the skillset array. We use a for loop to input the programmerID and the unique languageID for each language selected and add it to the new data array. All the programmer skillset information is now in this array occupying separate elements. To minimise slow down caused by repetitive database access and putting an insert command with a loop we use the .implode command to issue the contents of the array as a single entry so we can enter all values in one SQL query.



FORM.PHP

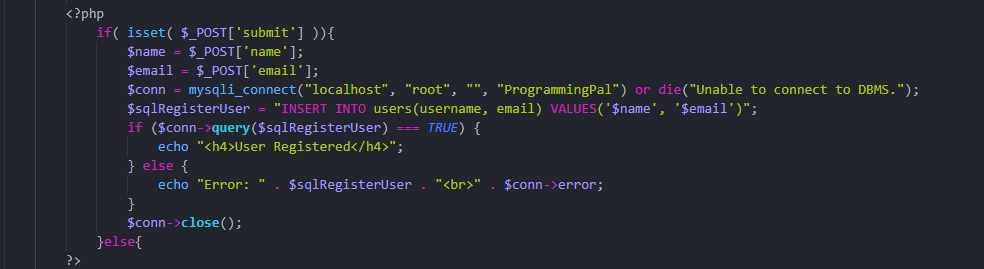
Key Elements:

-Pulls languages from database into checkboxes dynamically

-Registers programmer and pulls the new ID back

-Funnels null buttons submitted out and fills an array with selected languages and the new ID

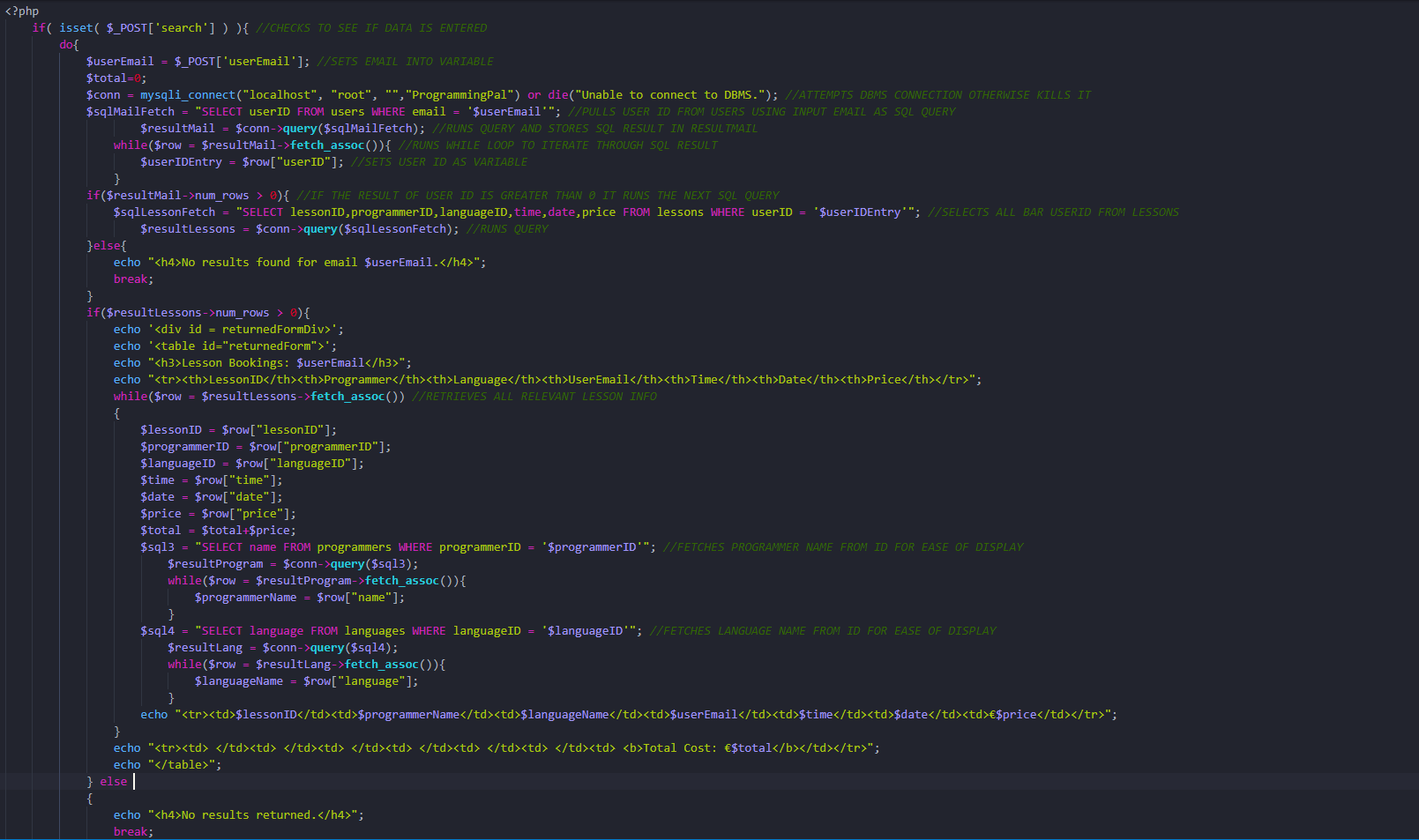
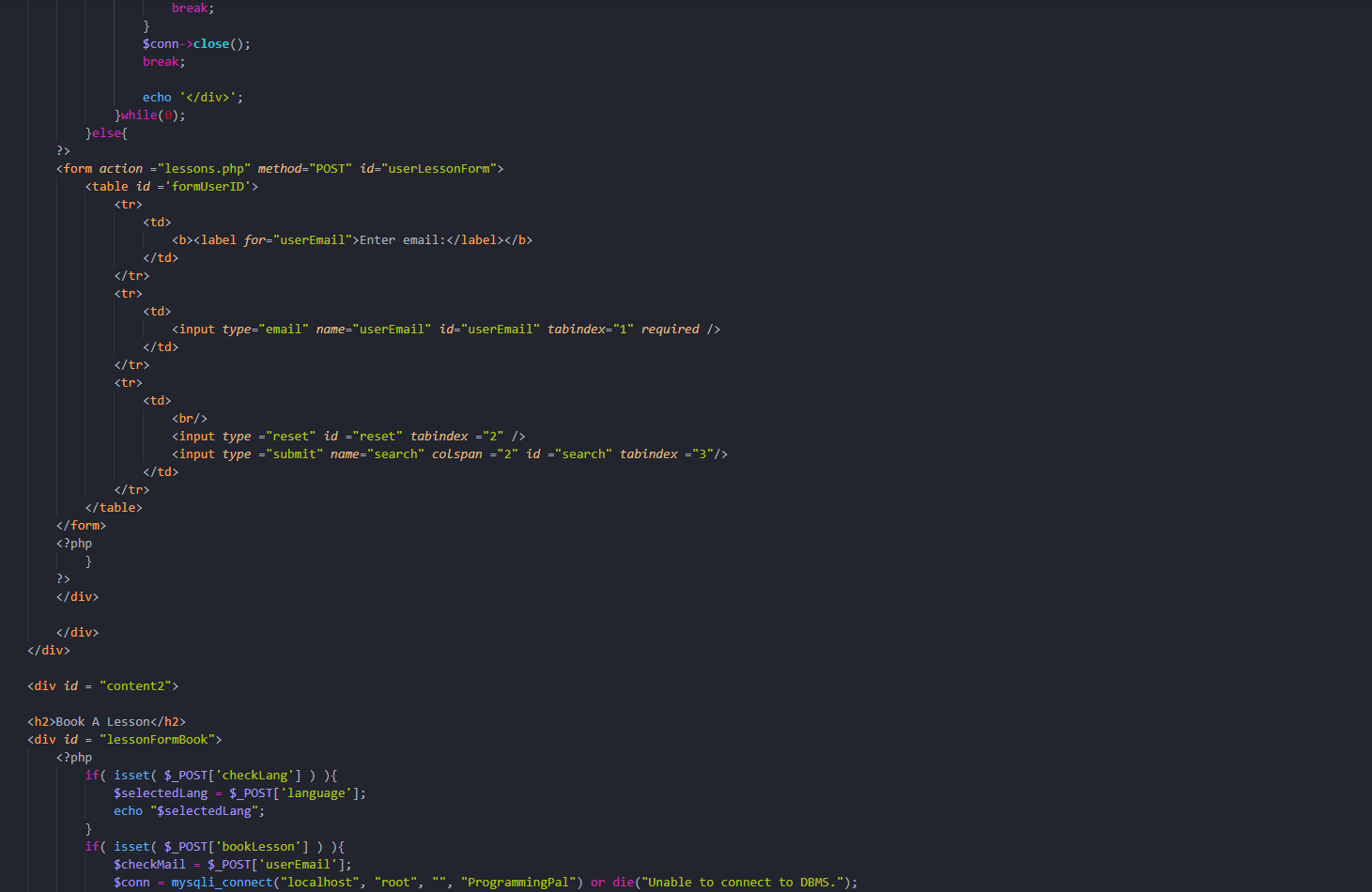
-Inputs the skillset into skills table using the array data via the. implode command.

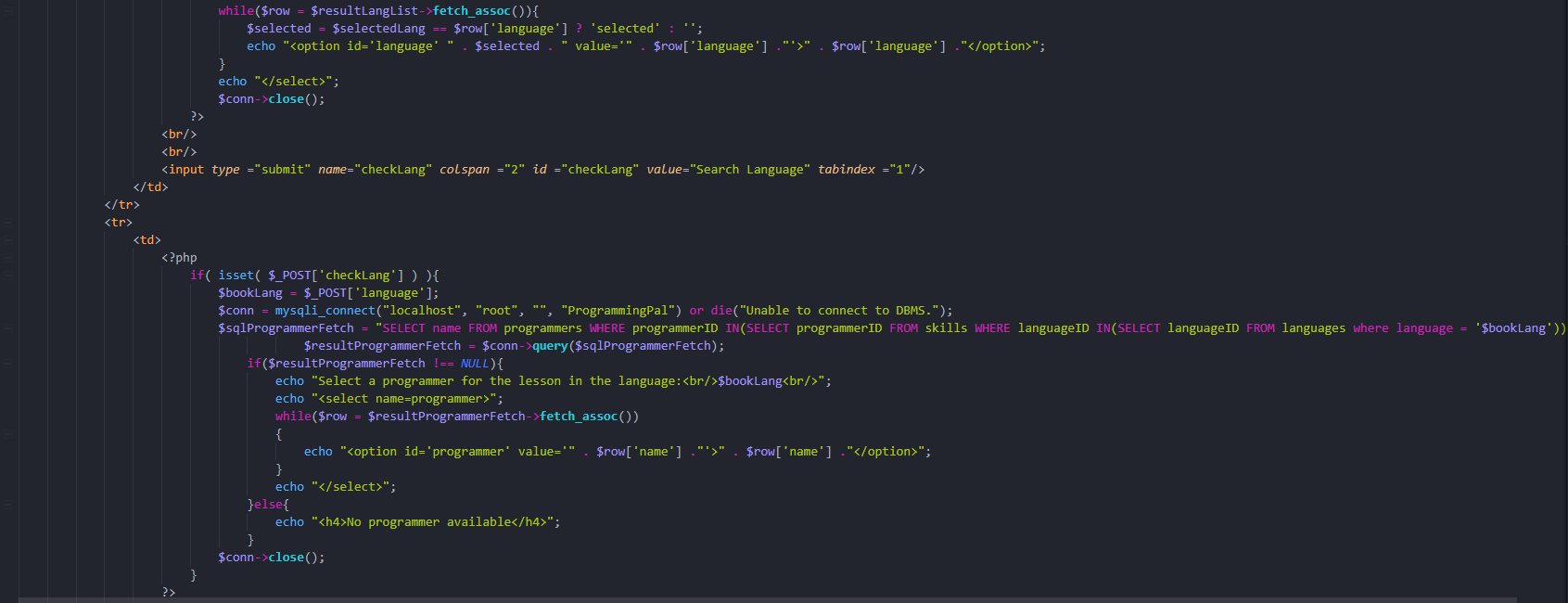
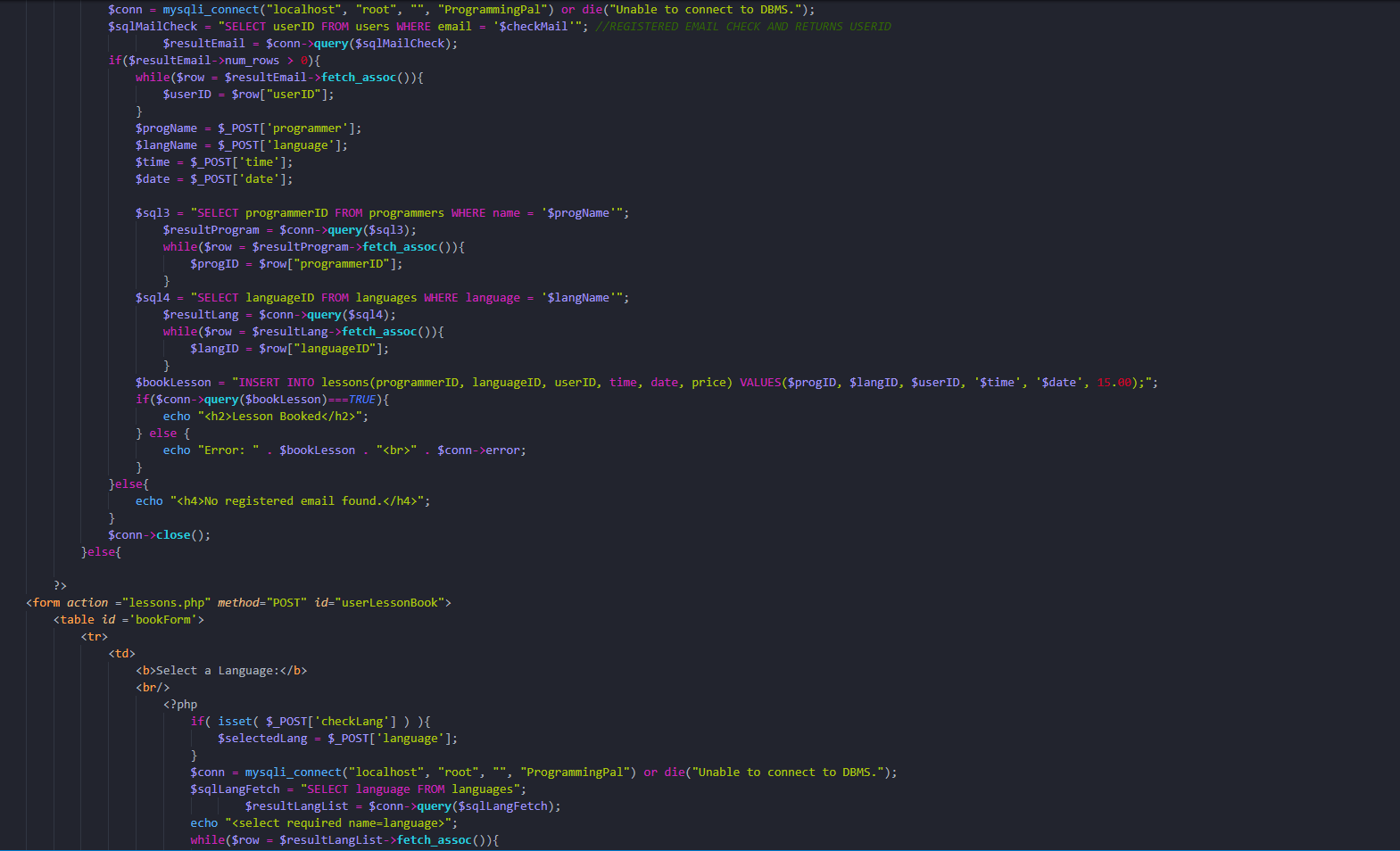
STUDENTFORM.PHP

Key Elements:

-Basic PHP insert form with 2 variables

-Straightforward connect to database and query SQL command



LESSONS.PHP

Key Elements:

- View bookings with several SQL fetching from other tables to make output user friendly.

-Language dropdown which then queries programmers for selected language.

-Default value date for todays date with locked hour selection slots between 9-6.

-Email verification for booking and view bookings.

-4 blocks of PHP run to 2 forms on the webpage - 3 for booking and 1 for show lessons.

**Results**

The system is fully operational and works as expected. It was an enjoyable project that enabled us to use our creativity. Challenges we faced included the difficulty of adjusting CSS and viewing changes because of google caching, some of the dynamic user form elements that are populated using data from the DB. The user entry is functional, and the finished website is simplistic in its elements opting for a less is more approach.

**Conclusions**

The chosen DB structure is suitable because firstly it works. It is low maintenance, manageable, and scalable. The properties of each table are suitable for a small start up company like the one we chose for the project and allows easy storage of various form inputs and user data.