| Team Member Full Name | NetID |
|---|---|
| Elizabeth Fitzgerald | efitzge5 |
| Elo Okor | eokor |
| Danielle Radford | dradford |
| | |

## Persistent Storage Design

We are using SQLite database to persist our data. The format of our database consists of 5 tables as listed in Figure 1 as: auth_group, auth_group_permissions, auth_permission, auth_user, and auth_user_groups. The entire schema is also defined in Figure 2. For example, the auth_group has a schema consisting of the id (primary key) and name (secondary key). Each of the tables consists of a similar pattern with the primary key being the id and the secondary keys being the other attributes like password, last login, etc. Refer to tables 1-4 for a reference, and please note that each column is its own table with the keys being the rows of the specific column.

| **auth_user** | **django_ admin_log** |
|---|---|
| id (PK) | id (PK) |
| password | object_id |
| last_login | object_repr |
| is_superuser | action_flag |
| username | change_message |
| last_name | content_type_id |
| email | django_content_type |

| | |
|---|---|
| is_staff | user_id |
| is_active | auth_user |
| date_joined | action_time |
| first_name | |

*Table 1. auth_user and django_admin_log database tables*

| **auth_user_user_permissions** | **auth_user_groups** |
|---|---|
| id(PK) | id (PK) |
| user_id | user_id |
| auth_user | auth_user |
| permission_id | group_id |
| auth_permission | |

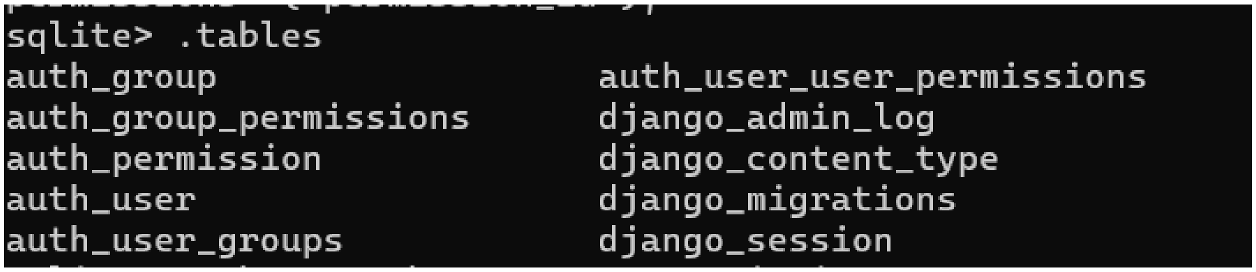
*Table 2. auth_user_user_permissions and auth_user_groups database tables*

| **auth_group_permissions** | **auth_permission** | **django_migrations** |
|---|---|---|
| id (PK) | id (PK) | id(PK) |
| group_id | content_type_id | app |
| auth_group | django_content_type | name |
| permission_id | codename | applied |
| auth_permission | name | |

*Table 3. auth_group_ permissions, auth_permission, and django_migrations database tables*

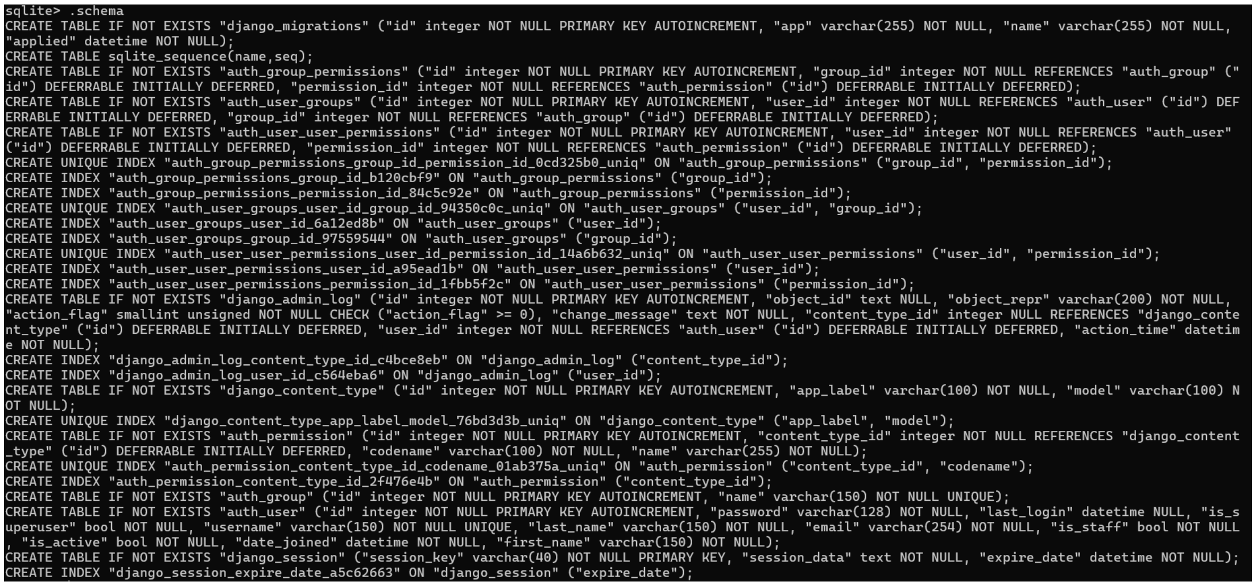
| **auth_group** | **django_content_type** | **django_session** |
|---|---|---|
| id (PK) | id(PK) | session_key (PK) |
| name | app_label | session_data |
| | model | expire_date |

*Table 4. auth_group, django_content_type, and django_session database tables*



```
sqlite> .tables
auth_group                    auth_user_user_permissions
auth_group_permissions        django_admin_log
auth_permission               django_content_type
auth_user                     django_migrations
auth_user_groups              django_session
```

*Figure 1. Image of all the table names*



```
sqlite> .schema
CREATE TABLE IF NOT EXISTS "django_migrations" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "app" varchar(255) NOT NULL, "name" varchar(255) NOT NULL,
"applied" datetime NOT NULL);
CREATE TABLE sqlite_sequence(name,seq);
CREATE TABLE IF NOT EXISTS "auth_group_permissions" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "group_id" integer NOT NULL REFERENCES "auth_group" ("
id") DEFERRABLE INITIALLY DEFERRED, "permission_id" integer NOT NULL REFERENCES "auth_permission" ("id") DEFERRABLE INITIALLY DEFERRED);
CREATE TABLE IF NOT EXISTS "auth_user_groups" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "user_id" integer NOT NULL REFERENCES "auth_user" ("id") DEF
ERRABLE INITIALLY DEFERRED, "group_id" integer NOT NULL REFERENCES "auth_group" ("id") DEFERRABLE INITIALLY DEFERRED);
CREATE TABLE IF NOT EXISTS "auth_user_user_permissions" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "user_id" integer NOT NULL REFERENCES "auth_user"
("id") DEFERRABLE INITIALLY DEFERRED, "permission_id" integer NOT NULL REFERENCES "auth_permission" ("id") DEFERRABLE INITIALLY DEFERRED);
CREATE UNIQUE INDEX "auth_group_permissions_group_id_permission_id_0cd325b0_uniq" ON "auth_group_permissions" ("group_id", "permission_id");
CREATE INDEX "auth_group_permissions_group_id_b120cbf9" ON "auth_group_permissions" ("group_id");
CREATE INDEX "auth_group_permissions_permission_id_84c5c92e" ON "auth_group_permissions" ("permission_id");
CREATE UNIQUE INDEX "auth_user_groups_user_id_group_id_94350c0c_uniq" ON "auth_user_groups" ("user_id", "group_id");
CREATE INDEX "auth_user_groups_user_id_6a12ed8b" ON "auth_user_groups" ("user_id");
CREATE INDEX "auth_user_groups_group_id_97559544" ON "auth_user_groups" ("group_id");
CREATE UNIQUE INDEX "auth_user_user_permissions_user_id_permission_id_14a6b632_uniq" ON "auth_user_user_permissions" ("user_id", "permission_id");
CREATE INDEX "auth_user_user_permissions_user_id_a95ead1b" ON "auth_user_user_permissions" ("user_id");
CREATE INDEX "auth_user_user_permissions_permission_id_1fbb5f2c" ON "auth_user_user_permissions" ("permission_id");
CREATE TABLE IF NOT EXISTS "django_admin_log" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "object_id" text NULL, "object_repr" varchar(200) NOT NULL,
"action_flag" smallint unsigned NOT NULL CHECK ("action_flag" >= 0), "change_message" text NOT NULL, "content_type_id" integer NULL REFERENCES "django_conte
nt_type" ("id") DEFERRABLE INITIALLY DEFERRED, "user_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED, "action_time" datetim
e NOT NULL);
CREATE INDEX "django_admin_log_content_type_id_c4bce8eb" ON "django_admin_log" ("content_type_id");
CREATE INDEX "django_admin_log_user_id_c564eba6" ON "django_admin_log" ("user_id");
CREATE TABLE IF NOT EXISTS "django_content_type" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "app_label" varchar(100) NOT NULL, "model" varchar(100) N
OT NULL);
CREATE UNIQUE INDEX "django_content_type_app_label_model_76bd3d3b_uniq" ON "django_content_type" ("app_label", "model");
CREATE TABLE IF NOT EXISTS "auth_permission" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "content_type_id" integer NOT NULL REFERENCES "django_content
_type" ("id") DEFERRABLE INITIALLY DEFERRED, "codename" varchar(100) NOT NULL, "name" varchar(255) NOT NULL);
CREATE UNIQUE INDEX "auth_permission_content_type_id_codename_01ab375a_uniq" ON "auth_permission" ("content_type_id", "codename");
CREATE INDEX "auth_permission_content_type_id_2f476e4b" ON "auth_permission" ("content_type_id");
CREATE TABLE IF NOT EXISTS "auth_group" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "name" varchar(150) NOT NULL UNIQUE);
CREATE TABLE IF NOT EXISTS "auth_user" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "password" varchar(128) NOT NULL, "last_login" datetime NULL, "is_s
uperuser" bool NOT NULL, "username" varchar(150) NOT NULL UNIQUE, "last_name" varchar(150) NOT NULL, "email" varchar(254) NOT NULL, "is_staff" bool NOT NULL
, "is_active" bool NOT NULL, "date_joined" datetime NOT NULL, "first_name" varchar(150) NOT NULL);
CREATE TABLE IF NOT EXISTS "django_session" ("session_key" varchar(40) NOT NULL PRIMARY KEY, "session_data" text NOT NULL, "expire_date" datetime NOT NULL);
CREATE INDEX "django_session_expire_date_a5c62663" ON "django_session" ("expire_date");
```

*Figure 2. Image of the Schema for the entire database*

## Demonstration of the Features

## Feature A

### Sub Feature 1 – Sign-up Page

Figure 3 shows a screenshot for the sign-up page for our Wordle app. The user can enter their first name, username, email address, and password. However, only the username and password is required.

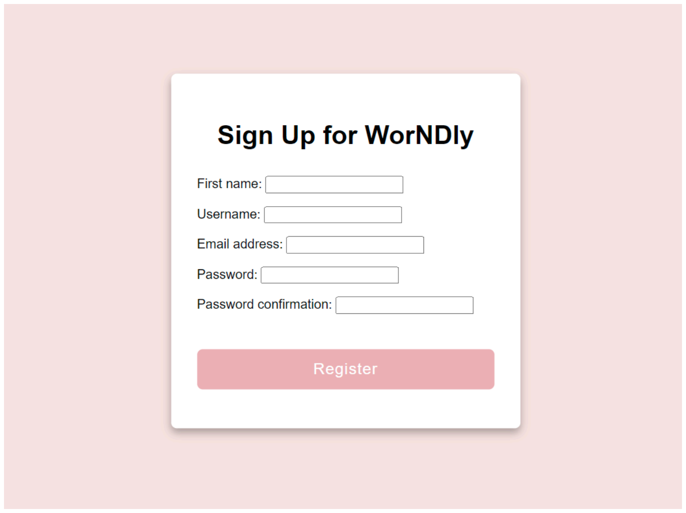


*Figure 3. Screenshot for Sub Feature 1 showing the sign-up page*

### Sub Feature 2 – Login Page

Figure 4 shows the login page for the app. Similar to the sign-up page, the user can enter their username and password used in the sign-up page to log back into the app.

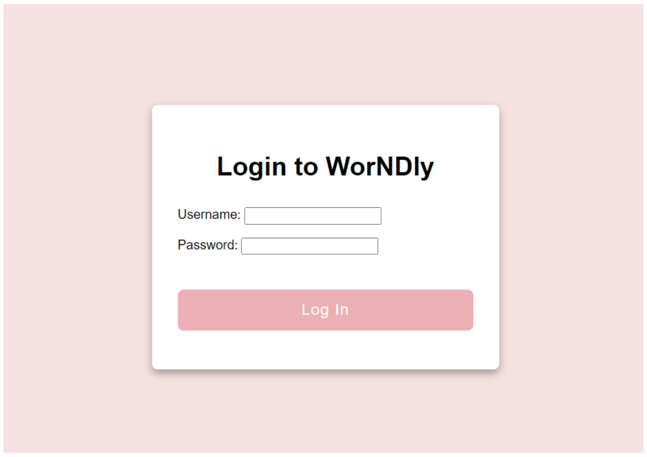
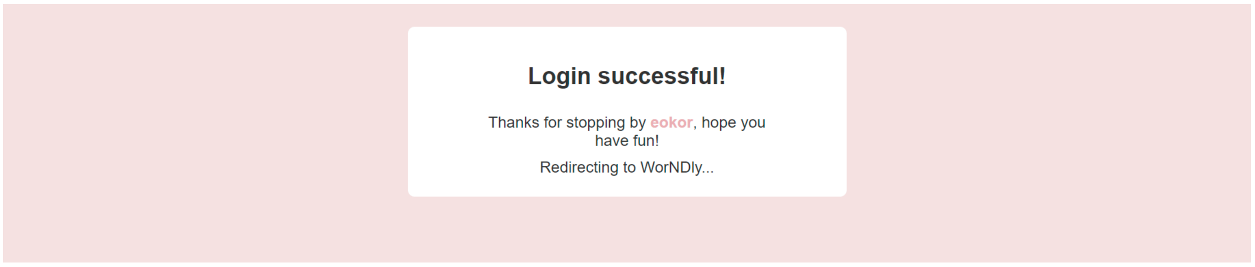


*Figure 4. Screenshot for Sub Feature 2 showing the login page*

## Sub Feature 3 – Success Page

Figure 5 shows a screenshot for the success page which appears after the user is able to either successfully create a new account on the website, or when the user successfully logs in to their existing account. The page will redirect to the start page after 5 seconds have passed.



Login successful!

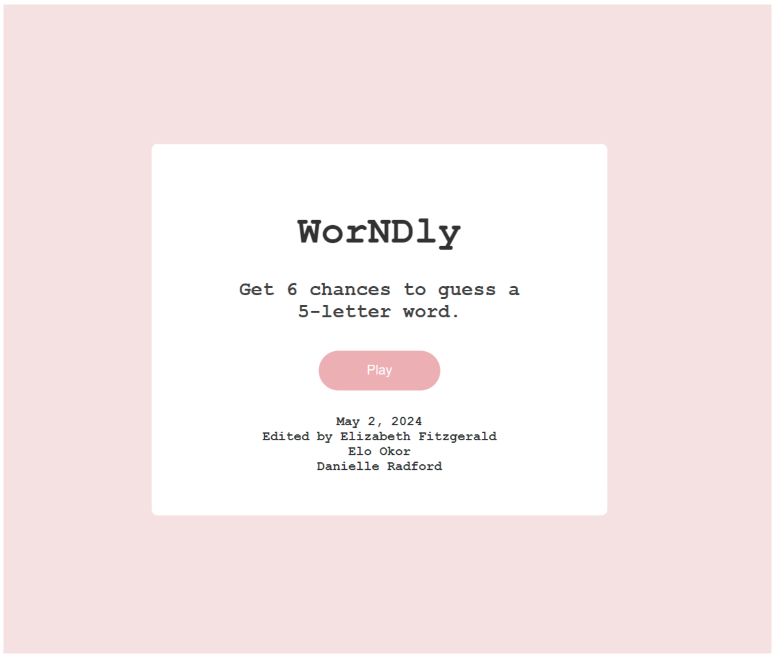
Thanks for stopping by eokor, hope you have fun!

Redirecting to WorNDly...

*Figure 5. Screenshot for Sub Feature 3 showing the success page*

## Sub Feature 4 – Start Page

Figure 6 shows the start page, which displays a short description of how to play WorNDly. It also displays the current date and the creators of this app under the Play button.



WorNDly

Get 6 chances to guess a 5-letter word.

May 2, 2024
Edited by Elizabeth Fitzgerald
Elo Okor
Danielle Radford

*Figure 6. Screenshot for Sub Feature 4 showing the success page*

## Feature B

### Wordle Gameplay

Figure 7 shows the implementation of gameplay. When the user inputs a word, they can see what letter is incorrect and correct based on the color coding conventions. If correct, it will also let the user know if the placement is correct. The user can either use their own keyboard or the one displayed on the site. While not implemented, the user should also be able to select the language in the top-left corner of the screen.

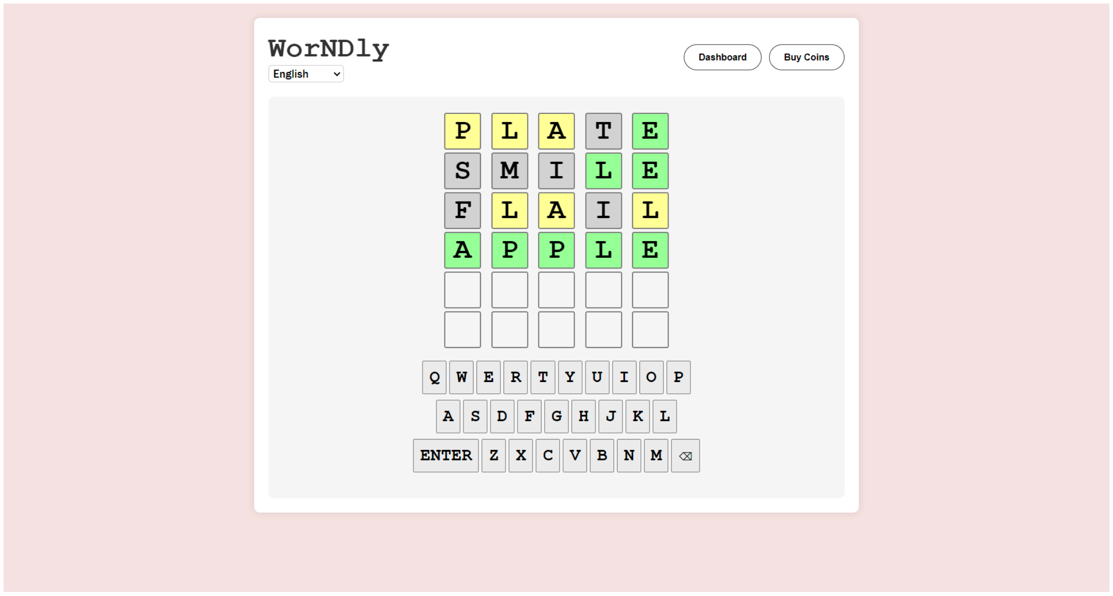


*Figure 7. Screenshot for Feature B*

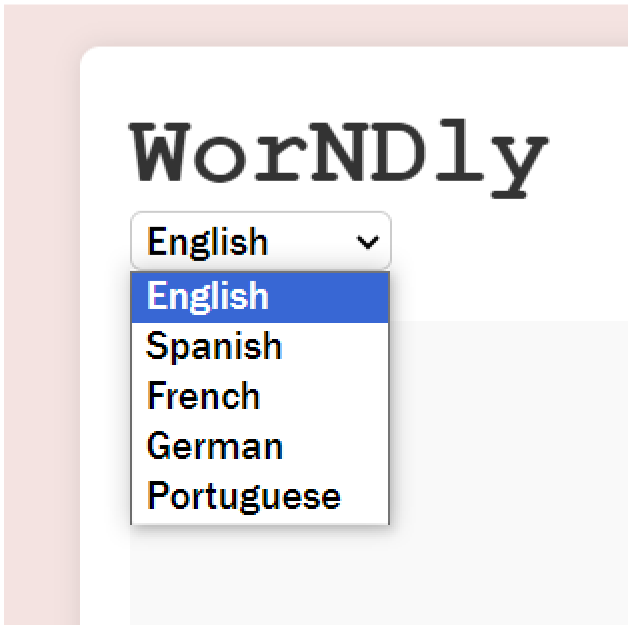


*Figure 8. Screenshot for the language selection*

## Feature C

Figure 9 shows a screenshot of the design for the dashboard. While not implemented, it should have the statistics for the amount of games played, win percentage, and the guess distribution for how many guesses it took the user to complete the game.

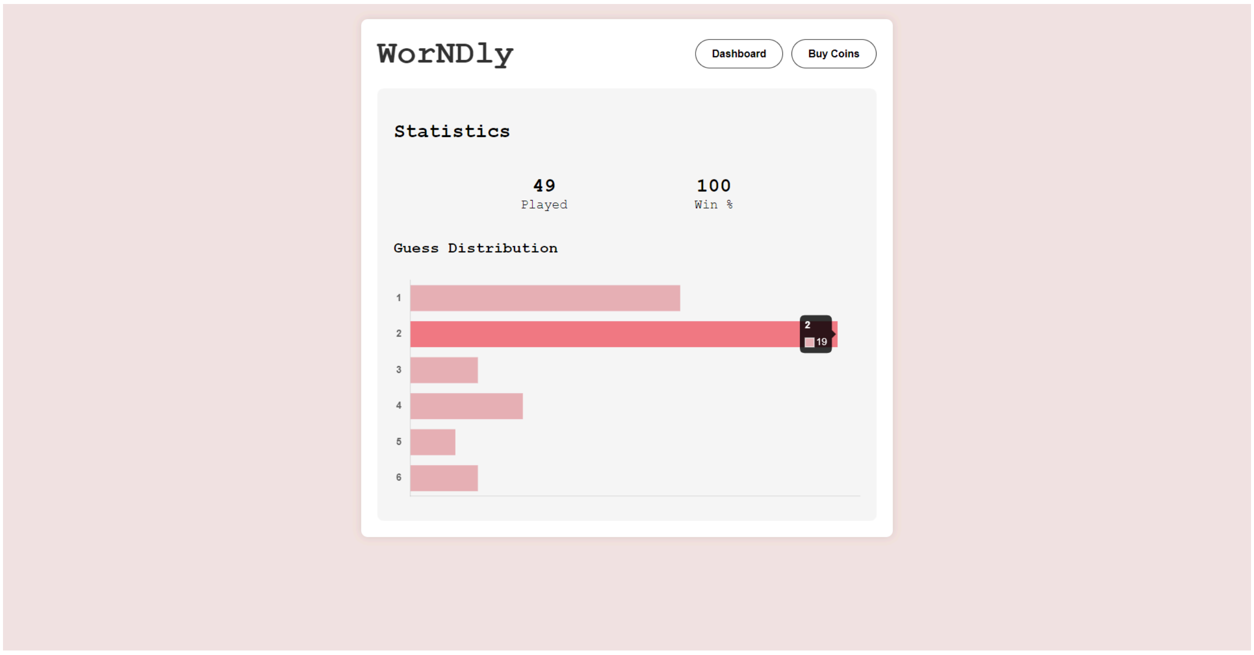


*Figure 9. Screenshot for Feature C*

## Feature D

Figures 10, 11, and 12 show the screenshot of the interface for utilizing Kratos currency. The current balance is displayed using an API to get the logged-in user's balance if they have an account in the backend. The user can either use the plus/minus buttons to change the number or type into the input box to change it directly. When the purchase button is clicked, a backend function uses an API call to make a purchase.

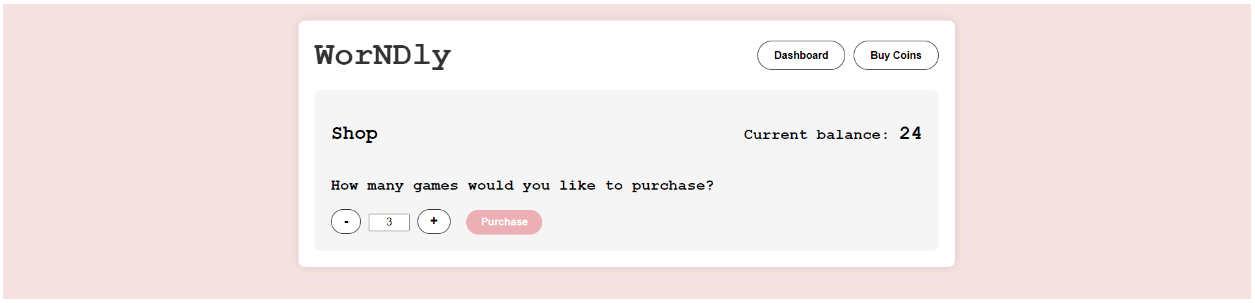
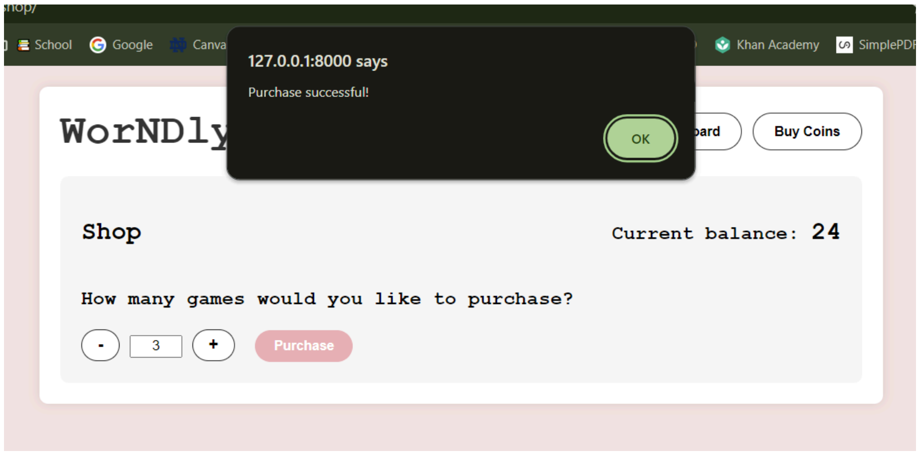


*Figure 10. Screenshot for Feature D*


127.0.0.1:8000 says

Purchase successful!

*Figure 11. Screenshot of purchase confirmation*

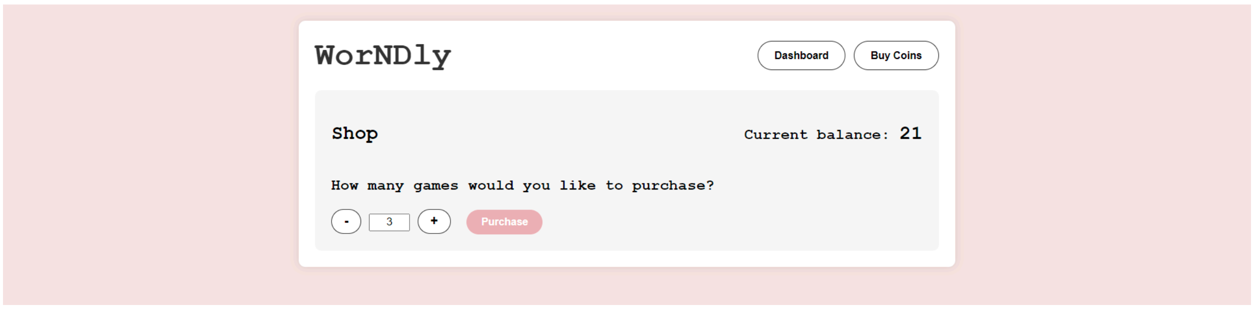


*Figure 12. Screenshot of page after clicking "Ok"; Note the decreased current balance*

## Project’s Learned Lessons

1. We used Javascript, Python, CSS, HTML, and Django (sqlite because Django uses it) in this project. If we were to start from scratch, **we would not** make different choices because we would not know how to implement it any other way. I think the bulk of the logic in JavaScript and Python was helpful because that was easy to implement with other aspects. For example, using JavaScript, we could easily connect to html using templates.

2. Dealing with the database and attempting to connect all of the different pieces was difficult. For example, getting the data from the database to display to the user via fancy graphs or also converting from Python to JavaScript, specifically when trying to read from a file. Also because we were doing work separately, when combining them everything was not always implemented the same. For example, some of the wording was different for files so when referencing them we had to go back and forth between them.

3. The largest positive effect on the project’s outcome was meeting and talking things through. As a group, we did our best to ensure that everyone was on the same page. On the other hand, we should have managed our time better.