LetterBuddy Software Test Plan (STP):

Introduction:

LetterBuddy is a web application for children's handwriting exercises. Its AI feedback is based on OCR and VLM models. Adults guiding the children on this journey can view their progress. Our objective is to ensure the correct functioning of the user management, generation of exercises, submission, and the adult's progression tracking of their children.

Test Items:

- JWT-based User authentication module.
- Exercise generation.
- Exercise reviewing that provides both a numeric score and written feedback(by a VLM).
- Adult's child management module.
- Adults' statistics on their child's performance.

Features To Be Tested:

1. JWT-based User authentication

   1.1. Adult registration - An adult can register if they provide a valid password, a unique username, and a unique email.

   1.2. Adult and child login—as long as they provide the username and password they registered with. Once they have logged in, a JWT access token will be provided and saved on the front end.

   1.3. Adult & child logout - when logged in, they can log out of their account, cleaning the JWT tokens saved and returning to the splash screen.

   1.4. JWT token refreshing will be done automatically by the frontend once the access token expires.

2. Adult can manage their child users
   2.1. An adult could register a new child who will be connected to their account, as long as they provide a unique username and a valid password for him/her.
   2.2. An adult could view all the children registered by them and thus connected to their account, and only them.
   2.3 An adult could delete one of their children's accounts, completely removing them from the system.

3. An adult can view one of their specific child's exercise submission tables
   3.1. An adult could view the exercise submissions of that specific child, and only them, including their submission date, numeric score, level, and what was requested to write.
   3.2. An adult could sort these submissions by the numeric score, submission date, and what was requested to write.
   3.3. An adult could view more detailed info about a specific child's submission, such as the handwritten text image submitted by them(saved on Cloudinary) and the VLM's deep analysis about the quality of the writing.

4. An adult could view one of their specific child's performance statistics graphs.

5. An adult could view handwriting tips and articles provided from our handcrafted collection.

6. A child can view the alphabet GIFs guidelines to better understand how to write each letter correctly.

7. A child can exercise their handwriting by getting a generated exercise that fits their current level, writing it down, uploading or capturing it, and submitting it.
   7.1. The exercise generated has to fit the child's current level and contains only child-friendly content.
   7.2. The child can either upload a photo of their writing or capture it using the device's camera within the application.
   7.3. The submitted image will be uploaded to Cloudinary.
   7.4. The submitted image will be evaluated using a combination of an OCR model and VLMs. The VLM will evaluate what the child wrote compared to what was requested, saving a total score and conducting a vast textual analysis.
   7.5. According to the average score of the child in his current level, the level will be updated accordingly.
   7.6. A basic positive feedback will be shown, as the child will get his next exercise that fits the updated level.

Features Not To Be Tested:

All of these **services** won't be tested **besides ensuring successful integration and data retrieval** that we require from them:

- DB queries handled by Django's ORM.
- Capture and upload images services.
- GitHub models - via Azure's SDK and Groq's API to send prompts to various VLM models.
- Cloudinary - to store and retrieve submitted exercise images in the cloud.

Testing Strategy:

To test our system affectionately, we will use various strategies, such as:

Unit testing:

We will conduct wide manual testing to check the features we have mentioned.

In addition, we will prepare a set of automated unit tests that will run each time we update our main repository on GitHub, both for the backend and frontend, and thus will reduce the amount of tests we will have to conduct manually.

- **For the frontend:** We will use a combination of e2e and unit tests that will provide good test coverage.
  For end-to-end testing, we will use a framework that provides a way to simulate real user scenarios and validate the entire application.
  We will test important user flows, such as:
  1. User login/sign up.
  2. Child's exercise submission.
  3. An adult adding a new child user.
  4. An adult viewing a child's submission.

  In addition, we will use unit tests specific for the frontend that will check the rendering and functionality of specific React components. To achieve that, we will use 'mock functions' that will simulate the data coming in and out of the component.

- **For the backend:** In these automated unit tests, we will simulate the frontend, which will make specific requests, and we will ensure that the results of the tested requests are as expected. These tests will use a snapshot of the DB - a separate test DB, which will have the same structure as our DB. Thus, based on a specific request, we will know what response we are expecting, based on the data we have inserted into it as part of each test.
  Those tests will check the various features mentioned above, with emphasis on ensuring correct data delivery from the DB. Such tests will be: ensuring a created user could login successfully and

acquire JWT tokens, adult will be able to access and edit only their own children info, a child/adult wouldn't be able to perform actions they are not authorized to, a child which requests an exercise would get one fitted for his level, ect.

Integration tests:

We will have to ensure our system communicates as one unit, as intended. The integrations that should be checked are:
Our front end (and only it) will be able to send requests to our back end (API) and receive their responses.
Our backend will be able to communicate with:

- The DB, stored in Neon.
- Cloudinary - to store the submitted exercises there.
- Github models & Groq - that will supply VLM services via their API.

Test Environment:

Our tests will be conducted as our system will consist of:
Submission Images storage: Cloudinary
Database: PostgreSQL, deployed on Neon.
Backend: Django REST Framework server deployed on Render using Python version 3.9.10.
Frontend: Responsive React web application deployed on Render, and will be tested on both Windows desktop, Android, and IOS through the browsers: Google Chrome, Brave, Firefox, and Microsoft Edge.