Syntext Specification

High Level Summary

Syntext is a minimalistic, user-friendly typing practice solution made with programmers in mind. Users can visit our website and immediately begin honing their typing skills with little to no setup. With real time input validation, the user is encouraged to practice typing in a manner that favors both accuracy and speed. After completing the code on screen, metrics are displayed to give the user a sense of their overall performance. Design and user experience are a core focus of Syntext, favoring features that are simple yet robustly implemented such that our website is responsive and stable.

The Problem

What is the problem?

- People rely too much on IDEs and text editors for autocomplete when programming.
- People know what they need to type but are not nearly as efficient as they would like to be.

Why do we have this problem?

- Technology such as VSCode is very convenient, but we become reliant on it. This is problematic when it comes to high-pressure technical interviews that only take place in text documents.
- Typing speed is something that can only be improved through practice and repetition.

How can we help?

- We provide a clear, clean, and user-friendly interface that enables users to improve their typing speed while also getting their knowledge of the syntax down.
- We give insight to users' accuracy, enabling them to track their progress and compete with friends.

Who will benefit? Who are your product's intended users?

- Programmers! Particularly the ones looking to improve their syntax typing skills for situations like interviews.

Problem Statement: Syntext aims to help all kinds of programmers improve both their syntax typing efficiency while also helping them get their syntax down in the process! We want to develop an aesthetically-pleasing, user-friendly website that includes a programming language typing interface. Whether you're competing with friends, improving your skills, or practicing for interviews, Syntext is the tool to use!

Project Team Breakdown

The team will have two subteams: Front-End Development and Back-End Development. The Front-End Development team will focus on website design and features along with improving the user experience. The Back-End Development team will focus on making a code database and implementing an API to make the website interactive and functional. Elijah is the project manager.

Front-End Development will be using React.js, while Back-End Development will use Node.js, and SQLite for our Database.

Project Goals

MVP Scope:

Design and implement a website that allows users to practice writing code that is given to them in Java (more languages to be added after further development). They will be tested on accuracy, with the front end tracking the precision of user character inputs against the code snippet presented on screen. Red underlining and highlighting will offer feedback to the user when they make an error, so that it can be resolved.

Features that we will be avoiding as they are out-of-scope: compiler, leaderboard, login system, color customization, language options, etc...

High Priority (MVP)

Main page features:

- Typing interface
- Grayed out code snippets in background overlaid with user input
- Timer with some time metric to track speed / accuracy

- Restart button to restart challenge
- Typing cursor to track user progress through code snippet
- Language feature and length selection

Low Priority (Post-MVP):

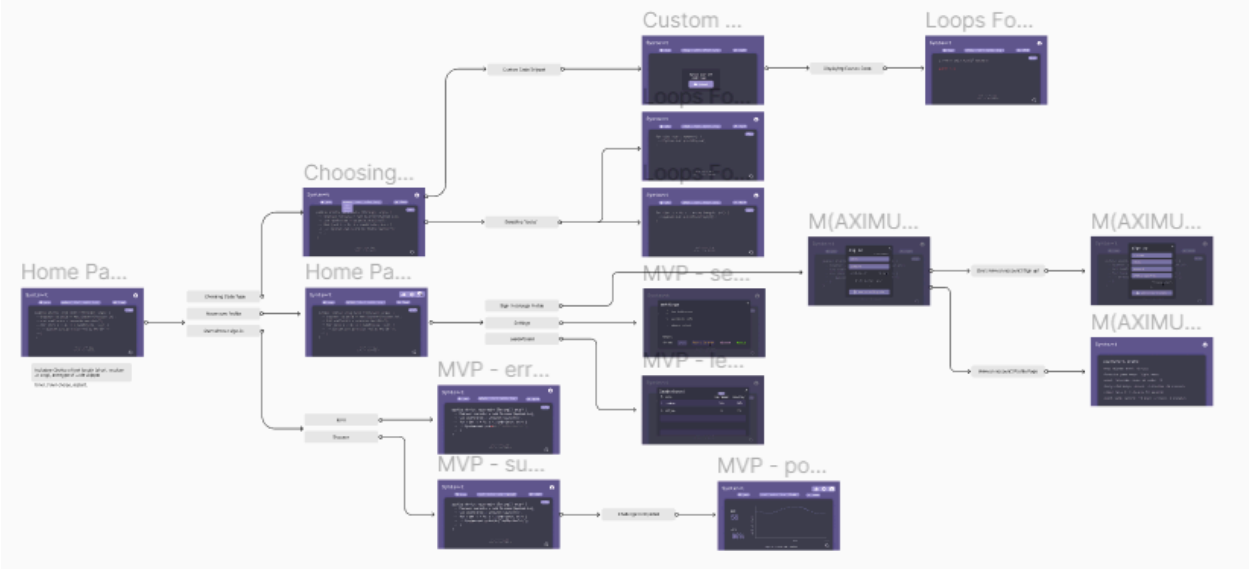
- Settings
    - color themes
    - different language
    - timer on/off
    - font
- Keyboard shortcuts
    - retry/reset timer
    - next code chunk
- User accounts
    - login page
    - leaderboard
    - progress tracking
- Color coded language keywords

KPIs - Key Performance Indicators

1. **Functionality:** There should be no bugs in the product so that the user faces no issues while using it.
    a. Did we account for all edge cases?
    b. Is everything working as it expected?
    c. Does the website respond to user input fast? (code validation)
    d. Does the website load code snippets without delay
2. **User-friendliness:** Our features and the entire website should be easy to understand and interact with. This will be a key focus for our front-end during the design process
    a. Is all text easily readable?
    b. Are the buttons easy to access?

Design

View of Product:



You can find our [Figma wireframe] here. It includes both MVP and post-MVP features.

User Journey

1. The product will allow users to practice code typing.
2. Users open the web page and will land straight into the typing page with default challenge settings.
3. Timer begins as soon as the user starts typing. It can be reset by the reset button.
4. If the user makes a mistake and types the wrong character, the incorrect character will be displayed in red and the word will be underlined in red. The user will not be able to move on to the next word without fixing the word.
5. Once the user completes the code snippet, their typing statistics will be displayed in words per minute (wpm). WPM will be calculated by the metric of 5 characters being equivalent to one word.
6. Users can click on their choice of programming language listed on top of the screen.
7. Users can click on the account icon in the upper right corner of the screen to login and keep track of their progress. They will also be able to change settings such as dark or light mode and choose the color theme.
8. Users can close and reopen the tab, and still keep their preferred settings.

Back-End MVP

Our project requires a backend to display code snippets from our database. Our MVP database will be very simple, as we will only be storing ~30 example snippets, and can easily do so manually. In the scope of our MVP, we can continue to add snippets manually, but will be working towards automating this once our MVP is attained.

Deployment Stack

API/Routing: Node, Express
Database: MySQL
Hosting: Heroku (ECO plan), JawsDB addon (Kitefin shared plan)

For our MVP, we need an extremely simple database that stores example code with four different types of code snippets and three different lengths.

Example entry:

id	SnippetType	Length	Data
1	FOR_LOOP	SHORT	LINES [ ]

Length and SnippetType options:

Length
SHORT
MED
LONG

SnippetType
FOR_LOOP
CONDITIONAL
METHOD
MISC.

Lines[ ] format:

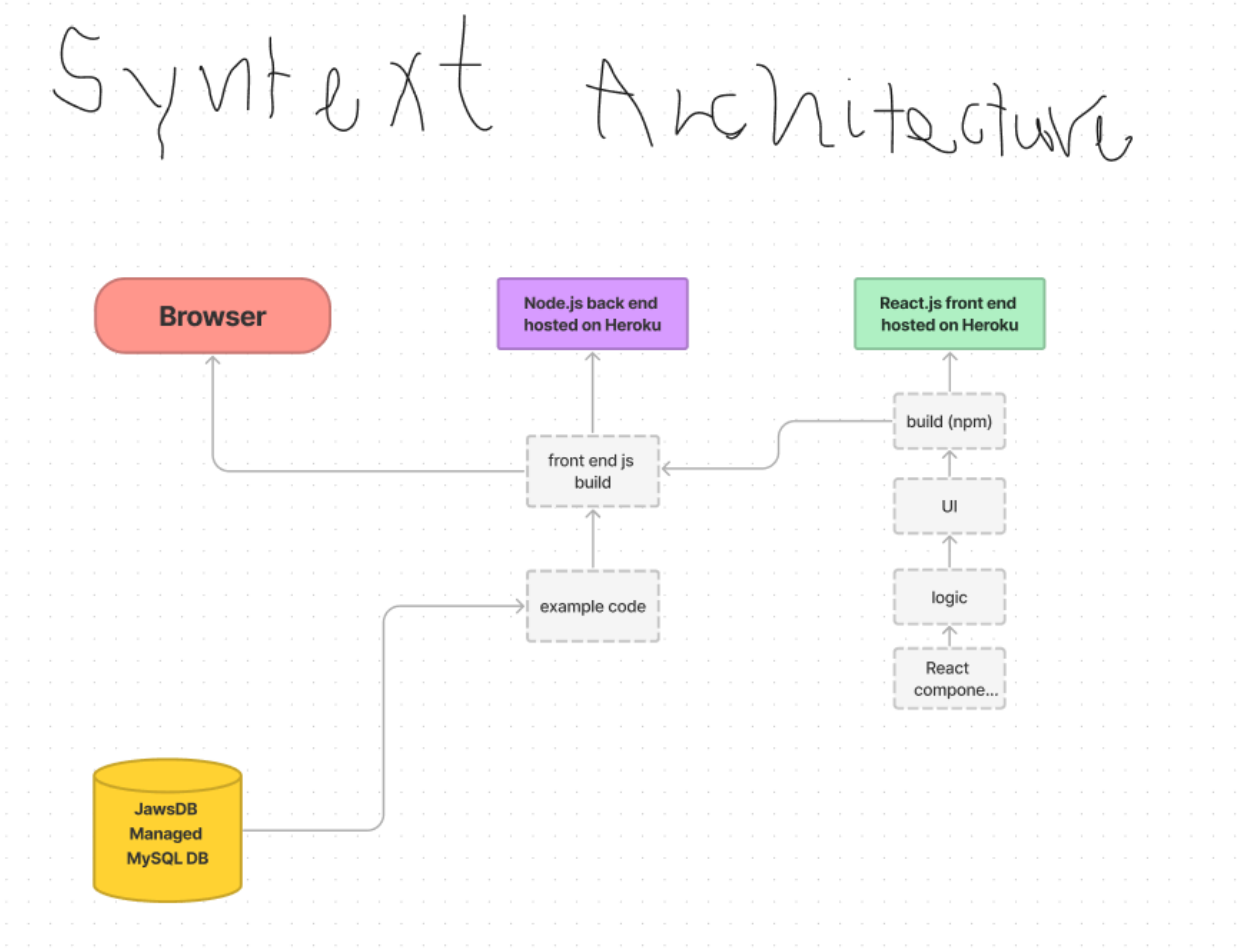
```
[ ‘int j = 20;’ ,  ‘for (int i = 0; i < 10; i++) {‘ ,  ‘\tSystem.out.print(j + “ - ” + i);’ ,  ‘\tif (i > j)’ ,   ‘\t\tSystem.out.println(“ < 0”)’ ,  ‘\telse’ ,  ‘\t\tSystem.out.println(“ > 0”)’ ,  ‘\tj -= 1;’ ,  ‘}’ ]
```

||
V

Display format:

```
int j = 20;
for (int i = 0; i < 10; i++) {
    System.out.println(j + “ - ” + i);
    if (i > j)
        System.out.println(“ < 0”);
    else
        System.out.println(“ > 0”);
    j -= 1;
}
```

We plan to source our code snippets from our CSE course projects because they already adhere to our stylistic parameters and are easily accessible.



Testing our backend

We will use various tools to test our backend, including DBBrowser to verify the formatting of resources stored in our database, and a “testing front-end” that consists of a simple text display area to check if our front-end and back-end are successfully communicating. The following milestones will be used to test our backend during development:

1. Store one-line example snippet and verify formatting
2. Display one-line example on testing front-end
3. Store multi-line example snippet and verify formatting
4. Display multi-line example on testing front-end
5. Identify possible edge cases via thorough testing
6. Store all edge cases and verify formatting
7. Display all edge cases on testing front-end

After reaching our MVP on a local deployment of Syntext, we will need to do additional testing to verify our hosting solutions are working reliably, and that we won't exceed our allocated resources on Heroku after deployment. First, we will need to repeat the above testing steps remotely to ensure our deployment is functioning properly. Next, we need to extensively test our application from a user's perspective and identify bugs to fix.

Back-End Post-MVP

id	username	password	leaderboard	settings
number	elimelt	d35a...g65f	stats	settings

stats = {
  best_wpm: 1000,
  avg_wpm: 5,
  rounds_done: 1
}

settings = {
  Tab_ind: true,
  Auto_tab: true,
  Smooth_cursor: true,
  Theme: matrix
}

Initially, we will simply store hashed passwords, and then for a user to login we will compare the stored hashed password with their input after applying the same hash function.

If time permits, we will implements "salting" our hashed passwords to make our site less vulnerable to brute force attacks, but this is hardly a necessity for Syntext.

Leaderboard outline:

When the leaderboard button is pressed by the user, the leaderboard will open and default to ordering by best words per minute.

The leaderboard will load 50 instances of user data per request, and it can scroll infinitely by loading the next 50 when the user gets close to the bottom of the list.

The user will be able to click on the different columns (avg_wpm, best_wpm, rounds_done) to sort by those columns instead. Doing this will requery the database and reset the list to start at the top again.

When requesting leaderboard data from the server, complete an initial request for top 1000 users, but only display 50 at a time to make sure leaderboard data does not get skewed as the user loads more.

General Timeline:

1. ~~**December 1st, 2022** (General Meeting):~~
    a. ~~Finish Project Specification~~
    b. ~~Finalize UI~~
    c. ~~Finish Research phase~~
2. ~~**End of Winter Break:**~~
    a. ~~Have clear front and back end responsibilities~~
    b. ~~Detailed tasklist for each member~~
3. **End of Winter Quarter: Complete MVP 1**
    a. Front End
        i. Main landing page
        ii. Functioning token validation for user's typing accuracy
    b. Back End
        i. Database
        ii. API
    c. Revise of additional features we want to implement
4. **End of Spring Quarter: Additional Features**
    a. Front-End:
        i. Settings: color, language, autofill, code chunk size, timer on/off, font, etc.
        ii. Keyboard shortcuts for restart, next, etc.
        iii. A login/menu page.
        iv. Color coded IDE identifiers.
    b. Back-End:
        i. Adding in different programming languages to database

    ii. Storing user data (login & stats)
    iii. Leaderboard
c. Done!