Code a Story

Documentation

September 5, 2023

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Table of Contents

Introduction

1.1 What is Code a Story?

Code a Story is a web-based application that puts students' literary sequencing and comprehension abilities to the test through story-based coding challenges. These challenges require students to utilize block code to traverse mazes that are based on stories' plots. Code a Story serves as an option to introduce coding to students through pre-existing curriculum, especially for schools that may lack dedicated computer science programs.

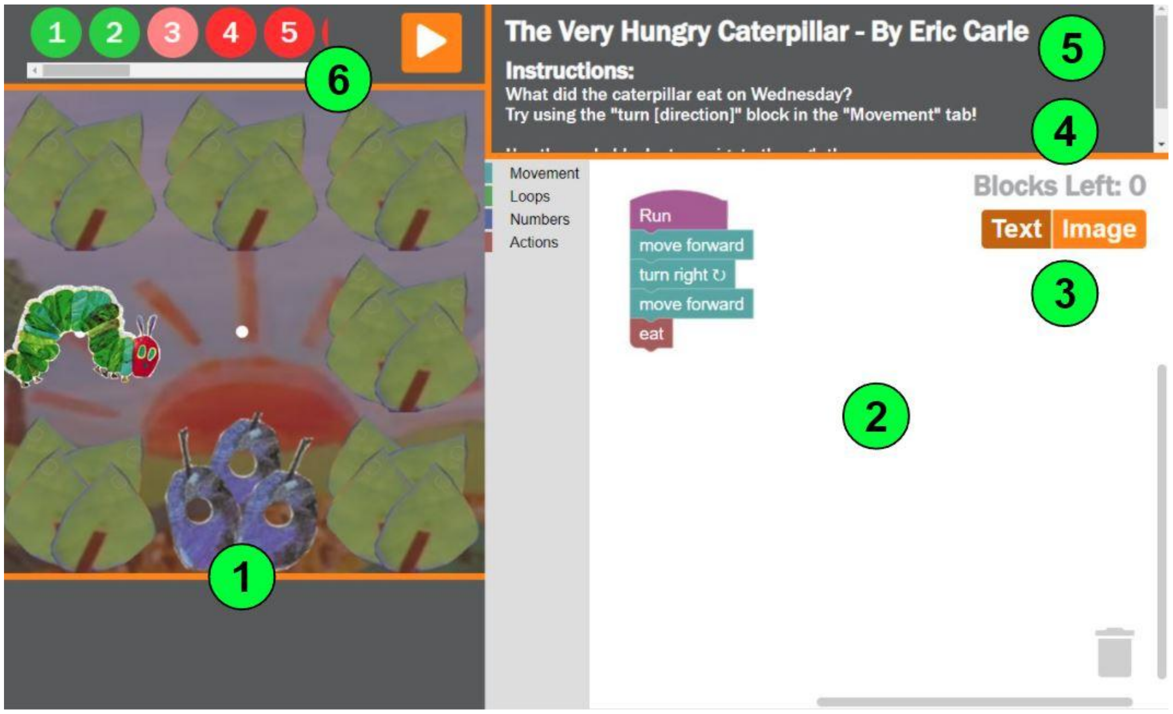
1.2 Development Objective

Our iterative development process of this web-based application prioritizes user-centered design, namely teacher customization. Whether allowing teachers to choose the maze difficulty that best suits their students' abilities or providing a selection of different grade level books, Code a Story is developed in a way that enables easy and effective integration into teachers' existing lesson plans with consideration of their unique class of students.

Student Interface

2.1 Block Coding Interface

Figure 1. Example Maze for *The Very Hungry Caterpillar* by Eric Carle


1
2
3
4
5
6
The Very Hungry Caterpillar - By Eric Carle
Instructions:
What did the caterpillar eat on Wednesday?
Try using the "turn [direction]" block in the "Movement" tab!
Movement
Loops
Numbers
Actions
Run
move forward
turn right ↻
move forward
eat
Blocks Left: 0
Text
Image

Students use block code to traverse mazes that are based on stories' plots. The labeled features of the block coding interface (Fig. 1) are described below.

1. This is where students determine where to move the character and observe what their code is doing. Character, boundary, and goal images and positions may change from level to level. Decoy goals are an option to verify students understand the story's plot. If the student's code causes the character to collide with a boundary or stop at a decoy goal, they will be notified of their error and asked to try again.

2. This is the students' workspace where the coding takes place. Sequences of block code may be formed and manipulated here. Only code attached to the "Run" block will contribute to the character's movement. All other code will be ignored at runtime.

3. Students have the option to use text or image block code. This provides a useful alternative to students who may not be able to read the instructions on code blocks.

4. Limiting the number of blocks that may be used is an option that challenges students to find the most efficient solution.

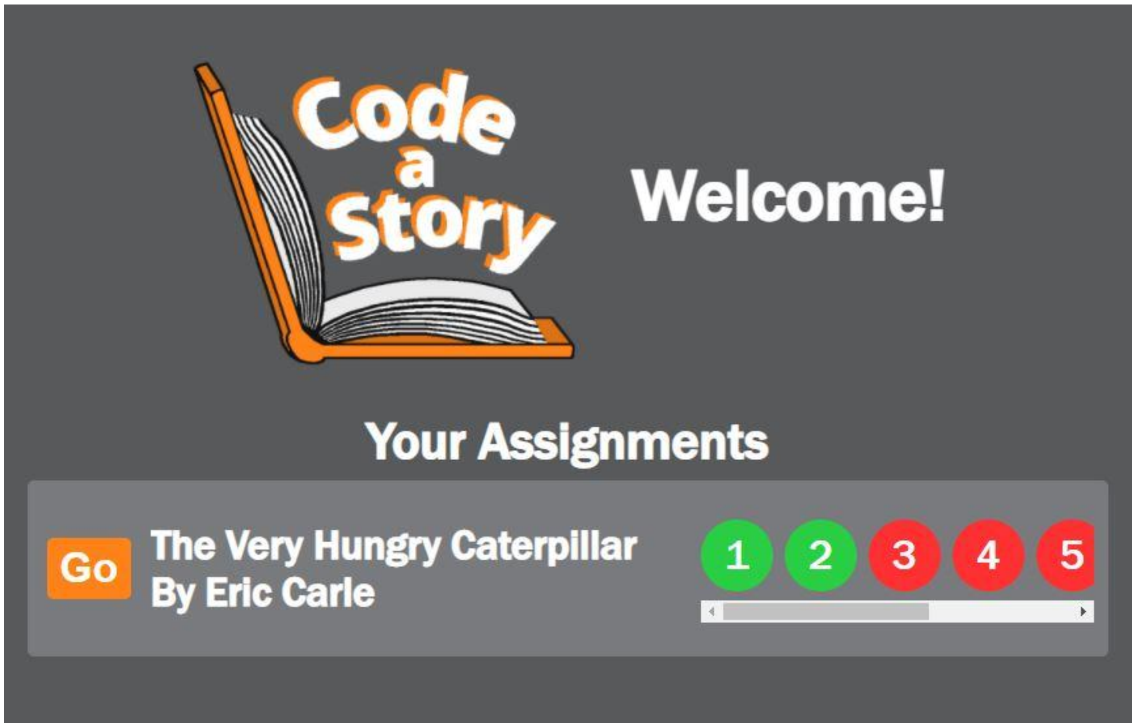
5. Instructions may be unique to a story enabling teachers to ask guiding literary questions or direct students to complete a reading before proceeding.

6. Students can view which levels they have completed (green), the current level they are on (pink), and the incomplete levels (red). Students may progress at their own pace.

Teachers also have the option to enable cutscenes between each level. This allows students to either read a story for the first time or gain a refresher on the plot.

2.2 Assignments

Figure 2. Example Incomplete Maze Assignment.

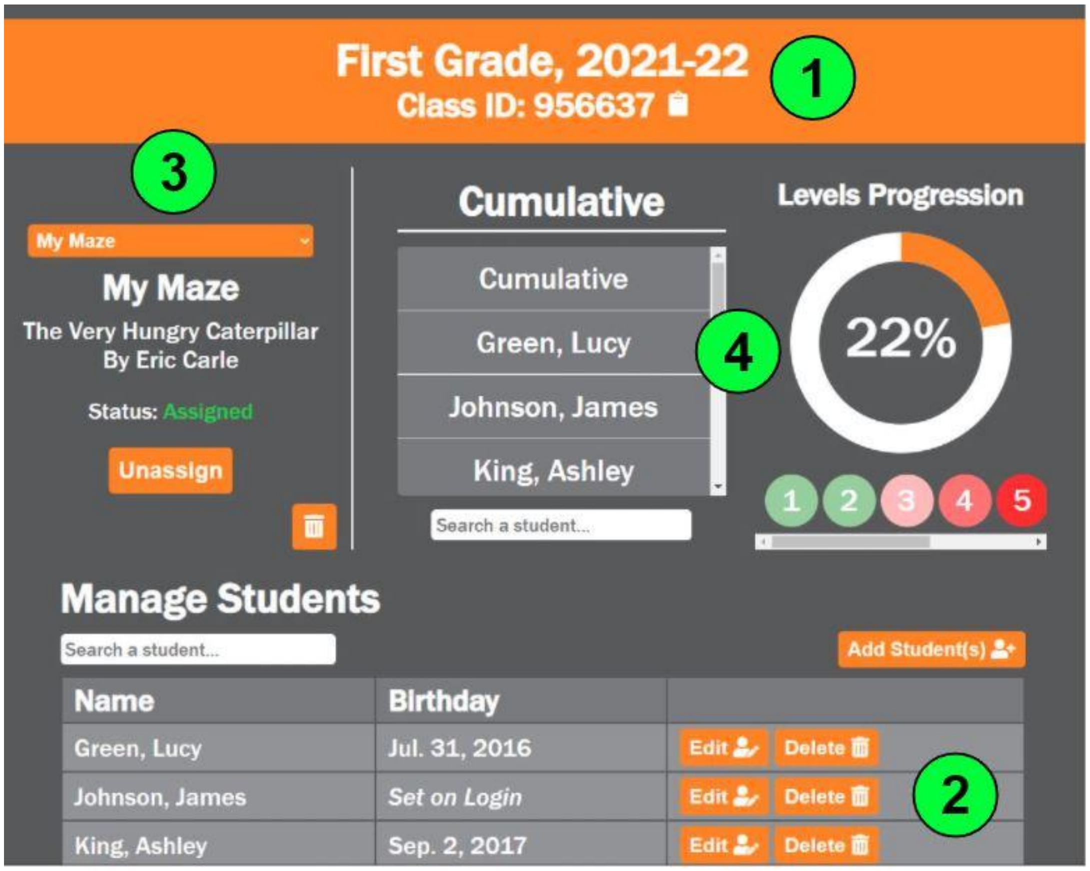


Students access the block coding interface (Fig. 1) through an assignment given by a teacher. The available assignments, as well as the student's progress, are viewable immediately upon login (Fig. 2).

Teacher Interface

3.1 Class Interface

Figure 3. Example Class With Assigned Maze and Progress Data



Name	Birthday	
Green, Lucy	Jul. 31, 2016	Edit Delete
Johnson, James	*Set on Login*	Edit Delete
King, Ashley	Sep. 2, 2017	Edit Delete

Teachers have a centralized interface for each of their classes where they can manage students, assign mazes, and monitor progression (Fig. 3). The labeled features of the class interface are described below.

1. Teachers may create a class within Code a Story for as many groups of students as needed. Whether teaching the same students all year or having several different classes per semester, Code a Story can separate teachers' students and assignments as needed. Each class has a unique "Class ID" which is used by students to login.

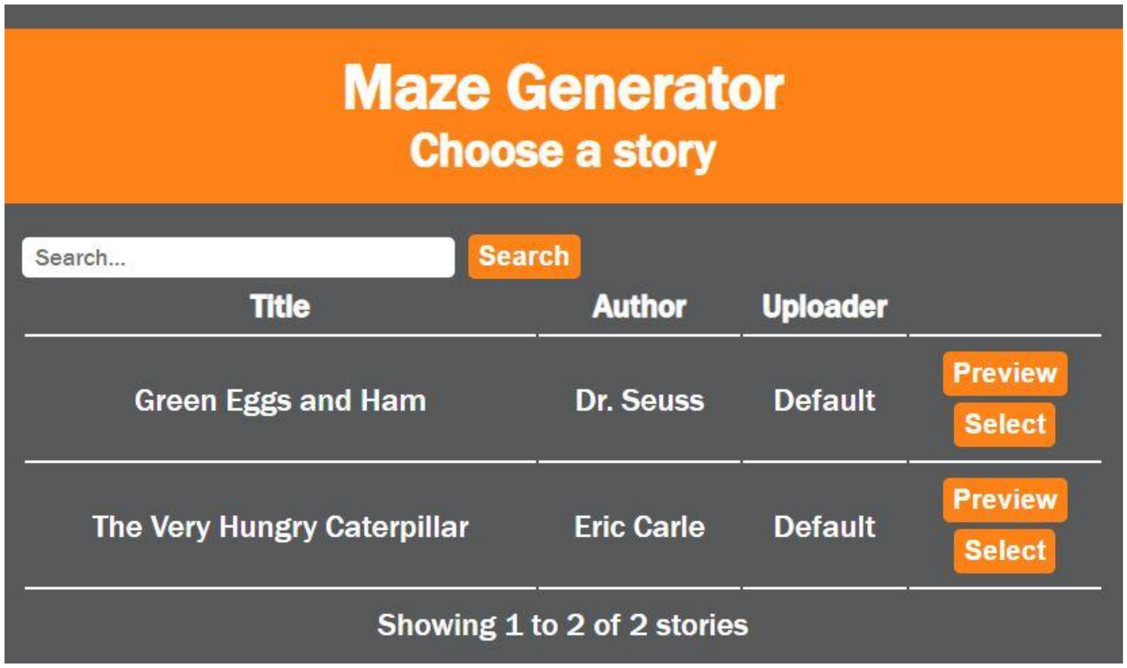
2. Teachers may add, edit, and delete students from the class. The editable information of a student includes their first and last name as well as their birthday which serves as their password to login. The birthday is used solely because it is easy to remember.

3. This is where teachers select a maze to assign/unassign to their students, delete, or view student progression. A maze must be generated through the generator interface (Fig. 4 & 5) in order to show up in the dropdown.

4. Teachers may view both cumulative and individual student progression on the maze. Progress is measured by a percentage indicating how much of the maze is completed. Additionally, the completion of individual levels may be viewed, either complete (green) or incomplete (red). For the cumulative option, the brighter the green, the more students have completed that level while the brighter the red, the more students have not completed that level.

3.2 Maze Generator Interface

Figure 4. Story Selector



Maze Generator
Choose a story

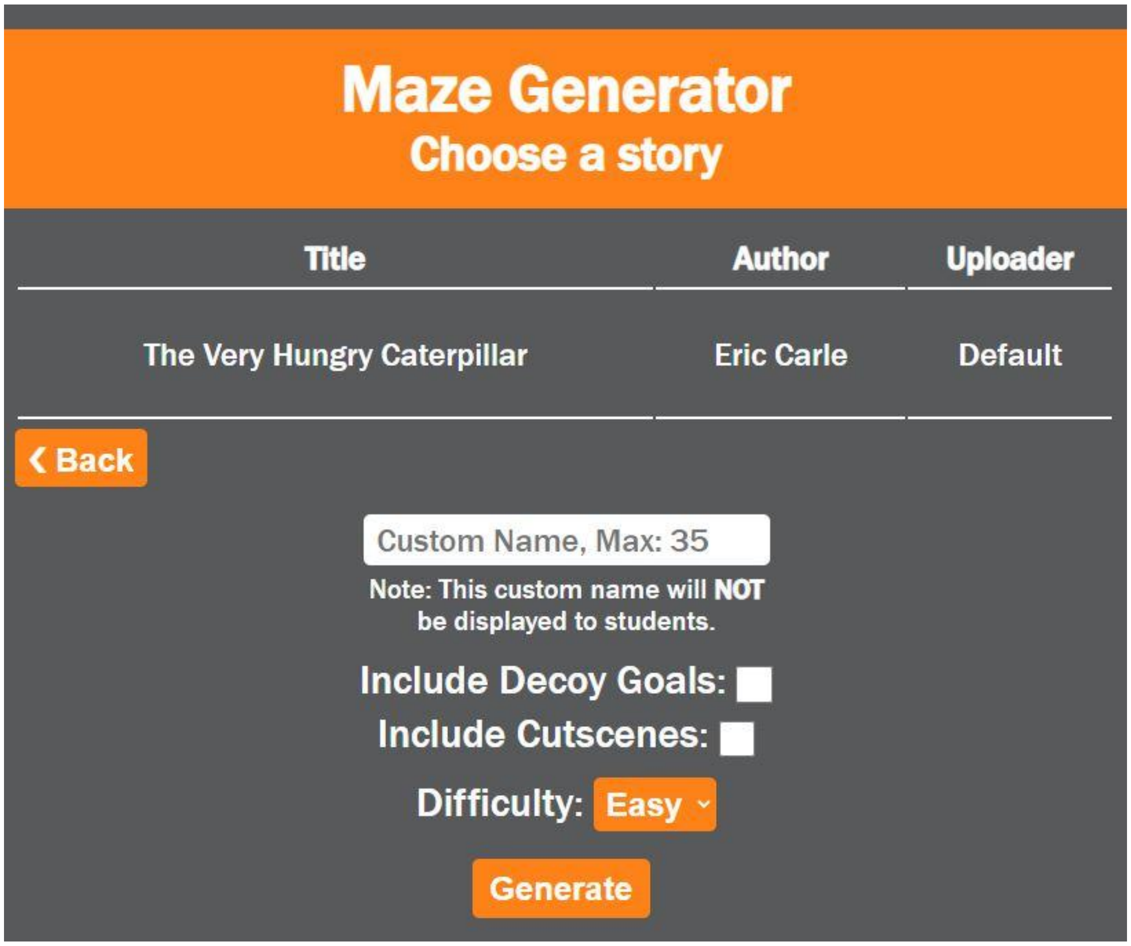
Search... | Search

Title	Author	Uploader	
Green Eggs and Ham	Dr. Seuss	Default	Preview Select
The Very Hungry Caterpillar	Eric Carle	Default	Preview Select

Showing 1 to 2 of 2 stories

In order to assign their students a maze, a teacher must first create one utilizing the maze generator interface (Fig. 4 & 5). First, a teacher must select a story to base the maze off of. Figure 4 shows this story selection interface with two example options, *Green Eggs and Ham* by Dr. Seuss and *The Very Hungry Caterpillar* by Eric Carle.

Figure 5. Generation Settings



Maze Generator
Choose a story

Title	Author	Uploader
The Very Hungry Caterpillar	Eric Carle	Default

Custom Name, Max: 35

Note: This custom name will **NOT** be displayed to students.

Include Decoy Goals: ☐

Include Cutscenes: ☐

Difficulty: Easy

After previewing the images and selecting the desired story, a teacher must choose the generation settings and provide a custom maze name that will only be visible to the teacher and used for future reference (Fig. 5).

After generation, the maze will be added to the teacher's account, not a specific class. In other words, the teacher may assign the same maze to several of their classes.

Development

4.1 Tech Stack

HTML, CSS, and JavaScript are used for front-end development. PHP and MySQL are used for back-end development.

Code a Story relies on several dependencies. Font Awesome is the icon library dependency. Learn more at https://fontawesome.com/. Blockly is the block-based coding library dependency. Learn more at https://developers.google.com/blockly. JS-Interpreter is a dependency that complements Blockly (e.g., highlighting blocks as they execute). Learn more at https://developers.google.com/blockly/guides/app-integration/running-javascript#js-interpreter and https://neil.fraser.name/software/JS-Interpreter/docs.html. jQuery is a JavaScript library dependency primarily used for its easier event handling and AJAX capabilities. Learn more at https://jquery.com/.

4.2 Prerequisites

Install XAMPP from https://www.apachefriends.org/. When prompted to "Select Components," only "Apache," "MySQL," "PHP," and "phpMyAdmin" are required. This enables you to locally run PHP scripts and a MySQL database. Note: There are alternatives to XAMPP, but XAMPP is easy to install and run.

Ensure you are able to access the production files and database by following Sections 7.1 and 6.1, respectively.

4.3 Setup

To set up your local development environment, follow the steps below:

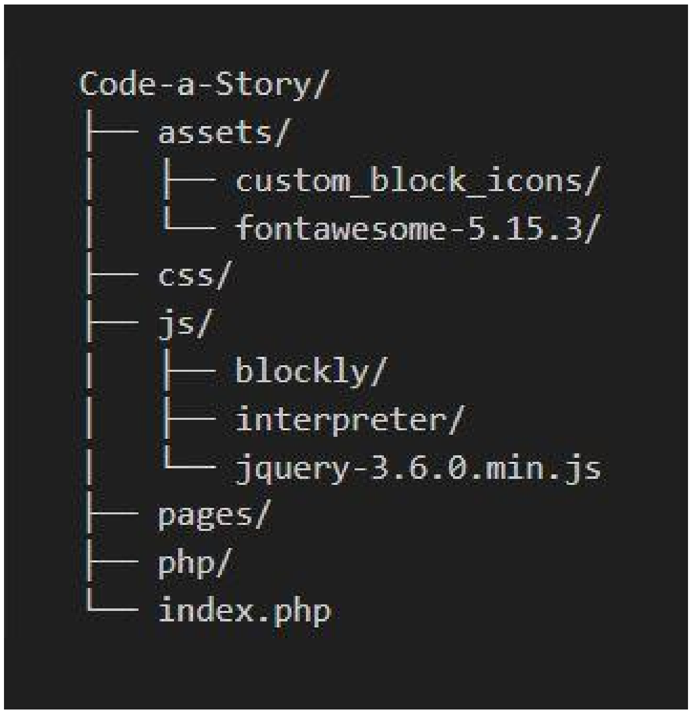
1. Access the production files (Section 7.1).
2. Copy all production files to your local computer in a new directory.
3. Open php/sqlConnect.php with a text/code editor.
4. Set *$DB_USER* to "root" and *$DB_PASSWORD* to an empty string.
5. Save php/sqlConnect.php and close the file.

6. Access the production database (Section 6.1).
7. Click “codeastory_database” on the left sidebar.
8. Click the “Export” button at the top of the page. Ensure a message like “Exporting tables from ‘codeastory_database’ database” appears.
9. Click the “Go” button on the bottom right of the page. This will download the database’s structure (and contents) as a SQL file.
10. Open XAMPP and click the “Start” button for “Apache” and “MySQL”. These need to be running to view Code a Story in a browser.
11. Open http://localhost/phpmyadmin/ in a browser, and if prompted, enter “root” as username and leave password empty.
12. Click “New” on the left sidebar, enter “codeastory_database” for the “Database name” field, select “utf8_unicode_ci” for the adjacent dropdown, then click the “Create” button.
13. Click the “Import” button at the top of the page. Ensure a message like “Importing into the database ‘codeastory_database’” appears.
14. Click “Choose File” under the “File to import:” section. Select the SQL file from Step 9.
15. Click the “Go” button on the bottom right of the page.
16. Ensure the database was successfully imported by checking for a message like “Import has been successfully finished, x queries executed. (codeastory_database.sql)” at the top of the page.
17. Create a new teacher account for local development by following Section 8.2 (except use your local database instead of the production database).
18. Open http://localhost/ in a browser and login with your teacher credentials.
19. Setup is complete! It is recommended that you click the “Stop” button for “Apache” and “MySQL” in XAMPP when you are not actively developing.

Code

5.1 File Structure

Figure 6. Visualized File Structure



```
Code-a-Story/
├── assets/
│   ├── custom_block_icons/
│   └── fontawesome-5.15.3/
├── css/
├── js/
│   ├── blockly/
│   ├── interpreter/
│   └── jquery-3.6.0.min.js
├── pages/
├── php/
└── index.php
```

Code a Story is broken up into several directories to separate the responsibilities of each component (Fig. 6).

The *assets* directory contains arbitrary files like images and icons. The *custom_block_icons* subdirectory includes the icons that are used for the code blocks when switching to "Image" mode in the block coding interface (Fig. 1.3). The *css* directory contains all of the styling files. The *js* directory contains all of the JavaScript that makes Code a Story dynamic. The *php* directory contains the PHP scripts used to communicate with the database and set/use session variables. The *pages* directory contains files where all of the aforementioned components are brought together into one file to create a web page. Most of the code in the *pages* directory is HTML; however, the file type is PHP so that the PHP scripts from the *php* directory can be included in the HTML code.

THE UNIVERSITY OF TENNESSEE KNOXVILLE

The *index.php* file is the page that the user lands on when navigating to codeastory.utk.edu; this page is blank since we immediately redirect to the login page.

The other subdirectories and files, including *fontawesome-5.15.3*, *blockly*, *interpreter*, and *jquery-3.6.0.min.js*, are Code a Story's dependencies (Section 4.1).

5.2 Security

It is important to be aware of the following security considerations.

The *sqlConnect.php* script is used to create a connection to the SQL database. This file includes sensitive information (i.e., database name, user, password, and host) and should not be uploaded to public repositories with such sensitive information.

When forming SQL statements in a PHP script, the *prepare* statement is used instead of directly executing SQL statements. Not only does this include performance improvements (reducing parse time and bandwidth to the server), but it also prevents SQL injection where hackers can run their own malicious code in Code a Story's SQL statements. Learn more at https://portswigger.net/web-security/sql-injection.

The *verifyAuthorization.php* script is included at the top of each page (except *login.php*) to ensure the user is authorized to be on that page based on a variety of factors. Reference the comments/code in *verifyAuthorization.php* to learn about such factors.

Database

6.1 Access

Code a Story utilizes a MySQL database. The code connects to the database via the *sqlConnect.php* script (Section 5.2). You can also access an interactive interface called phpMyAdmin by navigating to https://phpmyadmin.utk.edu/, selecting "Public LAMP MySQL Server (mariadbx)" as the server choice, and entering the *$DB_USER* and *$DB_PASSWORD* values from *sqlConnect.php* for the username and password fields, respectively.

The interactive database is especially useful for structuring the database (i.e., creating tables and defining their columns). Refrain from manually deleting rows from the database as this may result in dangling references (e.g., if a row in Table A references a row in Table B and only the row in Table B is deleted, then the row in Table A's reference to Table B is now broken). Instead, rely on Code a Story's existing delete features and PHP scripts to properly delete rows, when possible.

Accessing the database on your local development environment will be different from the aforementioned process. As described in Section 4.3, the credentials in *sqlConnect.php* should be different from that of the production environment's credentials. For example, *$DB_USER* could simply be "root" (i.e., the default superuser account) and *$DB_PASSWORD* could be an empty string – the specific values may depend on how you set up your local development environment. To access your local phpMyAdmin interface, navigate to http://localhost/phpmyadmin/.

6.2 Tables

The tables utilize IDs for several columns (i.e., "ClassID," "TeacherID," "StudentID," "StoryID," and "UploaderID"). These IDs enable other tables to reference the specific row of values that the ID originates from. For example, in the *classes* table, a class may be generated with a *ClassID* column value of 123456 and *Name* of "My Class." In the *students* table, five rows (i.e., five students) may have a *ClassID* of 123456. By correlating the row in *classes* and the five rows in *students* with the *ClassID* of 123456, we are able to derive that those five students belong to "My Class."

Below are descriptions of each tables' responsibilities:

The *assignments* table stores whether or not a story is assigned to students. When the *Assigned* column value is "1," then the story will appear on the students' dashboard for them to complete.

The *classes* table stores information about a class (i.e., class name and teacher).

The *cutscenes* table stores the source paths of cutscene images and defines which story and level those images belong to (i.e., *StoryID* and *LvlNum*, respectively). It also defines the order of the images in the cutscene (i.e., *CutscnNum*). Note: Some rows may have a *LvlNum* of "0" or one past the last level, denoting cutscenes to show before the first level is started or after the last level is completed, respectively.

The *levels* table stores the source paths of character, boundary, goal, and background images for each level of a story. It also stores the instructions for that level. These instructions can include, but are not limited to, code hints (e.g., types of code blocks to use), pages to read before completing the level, and a question(s) related to the story to test literacy skills. Note: The data stored in this table are consistent across different difficulty levels of the same story because the difficulty does not influence the story's plot, only the maze complexity (i.e., the *mazes* table).

The *mazes* table stores the positional coordinates of the character, boundaries, goal, and decoy goals in each level's maze (Fig. 1.1). The origin is the top-left grid unit, and the x- and y-values increase going right and down, respectively. The coordinate is formatted as "x,y". If there are multiple coordinates (e.g., *BoundCoords*), the coordinates are formatted as "x1,y1/x2,y2/etc." The *DecoyCoords* have *DecoyImgs* associated with each coordinate, where the *DecoyImgs* numeric value(s) corresponds to a level number(s) that the decoy goal image(s) will be from. For example, if *DecoyImgs* is "2,1" and *DecoyCoords* is "3,5/2,8", then two decoys will be placed in the maze located at coordinates (3,5) and (2,8) with the *GoalImg* of levels 2 and 1 from the *levels* table, respectively. The *mazes* table also stores the maximum number of code blocks that can be used for the level (i.e., *BlockCap*), serving as an extra coding challenge. *BlockCap* can also be "Infinity" for no maximum limit. Note: The "Run" block is counted as 1 block being used. The aforementioned data are associated with a difficulty level (i.e., *Difficulty*). Each story (i.e., *StoryID*) can have up to three difficulty levels (i.e., "0" for easy, "1" for medium, and "2" for hard).

The *progress* table tracks each student's current level (i.e., *CurrLevel*) on a story (i.e., *StoryID*) for all stories that have been assigned at least once. This enables students to continue across several coding sessions. Additionally, this powers the progress metrics in the teacher interface (Fig. 3.4).

The *stories* table stores information about a story (i.e., title, author, and source path of cover image). It also stores *Name* (e.g., "My Maze" in Fig. 3.3) to differentiate between multiple generations of the same story. For example, this name is useful for differentiating two generations of *The Very Hungry Caterpillar*, one as "easy" difficulty with cutscenes and the other as "hard" difficulty without cutscenes. The table also includes an *UploaderID* that defines who uploaded or copied a story; the column value either corresponds to a teacher's *ID* in the *teachers* table or is "0" meaning it was uploaded by the Code a Story team. The *stories* table also includes a *Published* column that can be one of two values, "1" meaning it is available to all teachers as an option in the generator interface (Fig. 4) or "2" meaning it is a story copied from an option in the generator interface and is assignable in the class interface (Fig. 3.3). Note: Stories are copied (i.e., *Published* is "2") from an original version (i.e., *Published* is "1") because of a feature idea where teachers could modify their individual copy (e.g., changing an image, rewriting the instructions, etc.) without changing the original version. This feature was never developed because it was out of project scope.

The *students* table stores information about a student (i.e., first/last name and their class). Students' passwords are set as whatever password the student enters on their first login. The passwords are unencrypted so that teachers can look them up in case students mistype while setting their password or forget their password.

The *teachers* table stores information about a teacher (i.e., email, first/last name, and school). Teachers' passwords are encrypted using bcrypt, specifically PHP's implementation called crypt_blowfish, with a cost of 10. When the *TempPwd* column value is "1", the teacher will be prompted to set a new password upon logging in. Once the new password is set, the column value will be set to "0". Note: The *IsAdmin* column was intended for an incomplete feature where users with *IsAdmin* as "1" would have the ability to add, view, and edit (i.e., change email, reset password, and delete) teachers' accounts via the dashboard interface. Since these tasks can be manually completed through phpMyAdmin, this feature was low priority and never completed.

Production

7.1 Access

Code a Story is hosted on The University of Tennessee Knoxville's virtual server space, managed by the Office of Innovative Technologies (OIT). Learn more at https://oit.utk.edu/accounts/web-accounts/.

To access the production files, follow the steps below:

1. Create an OIT Linux account at https://setuplinux.utk.edu/.
2. Gain access to the "codeastory" group by contacting OIT.
3. Install FileZilla Client from https://filezilla-project.org/. This application offers an easy-to-use, drag-and-drop interface for transferring files from your local machine to the remote server. Note: There are alternatives to FileZilla Client; learn more at https://utk.teamdynamix.com/TDClient/2277/OIT-Portal/KB/ArticleDet?ID=113598.
4. Open FileZilla and set up an interactive login with the following configuration:
   Protocol: SFTP – Secure File Transfer Protocol
   Host: linux2.oit.utk.edu
   Port: 22
   Logon Type: Interactive
   Username: <Your NetID>
   For a visual guide, follow the steps under the "FileZilla" section at https://utk.teamdynamix.com/TDClient/2277/OIT-Portal/KB/ArticleDet?ID=113598.
5. After configuring interactive login, click the "Connect" button. Enter your NetID password when prompted, then complete the secondary prompt for two-factor authentication (2FA).
6. Ensure you are connected by checking if content is populating in the "Remote site" section in FileZilla (i.e., the right side).
7. Once connected, enter "/home/codeastory/public_html" in the "Remote site:" path field.
8. If Code a Story's files appear (Fig. 6), you have successfully accessed production!

7.2 Deployment

To deploy your changes to the production environment, access production (Section 7.1), then drag and drop your modified local files to the appropriate remote directories, overwriting the corresponding remote files. Your changes will immediately be reflected at https://codeastory.utk.edu/, so ensure that you have thoroughly tested your changes. Note: You may need to hard refresh your browser to see the changes.

Refrain from re-uploading the dependencies (Section 4.1) since these can take a while to transfer due to their size. Refrain from re-uploading *sqlConnect.php* from your local development environment since your local credentials may be different from the production credentials.

Do not forget to update the production database structure (Section 6.1) if you made changes to your local database structure.

Administration

8.1 Adding a New Story

Adding a new story to the generator (Fig. 4) for teachers to use requires multiple steps, involving image editing, maze design, and database manipulation. The following subsections will walk through the steps necessary to add a new story.

To clarify, “story components” refers to assets (i.e., images) and information (i.e., author, title, instructions) related to the story while “mazes” refers to the story components’ placement on the maze interface (Fig. 1.1) and coding elements (i.e., maximum code block limit).

8.1.1 Story Components

Retrieve the title and author of the story you would like to add. Additionally, save an image of the story’s cover, or any relevant thumbnail image. If a cover image is not provided, a default image will be used.

Determine how many levels you would like this story to have based on the story’s plot. For example, *The Very Hungry Caterpillar* has a level for each day in the story. For each level, you will need to save four images that will be used in the maze. These four images will represent the character, boundary, goal, and background. For example, in Fig. 1.1, the caterpillar is the character, the leaves are the boundary, the apple is the goal (related to the first day in the story), and the background is the background from the story. These images can be reused across multiple levels. For example, the aforementioned caterpillar image is reused across all of the levels in *The Very Hungry Caterpillar* story. Ensure that your chosen images have transparent backgrounds (PNG file type); this may require some image editing.

Add all images to the *assets* directory in a new subdirectory named after your story.

You should also write instructions for each level. These instructions can include, but are not limited to, code hints (e.g., types of code blocks to use), pages to read before completing the level, and a question(s) related to the story to test literacy skills.

8.1.2 Adding Story Components to the Database

Adding the story components from Section 8.1.1 to the database will require manipulating the *stories* and *levels* database tables.

In the *stories* table, create a new row by clicking the "Insert" button at the top of the page. Enter your story's title and author in their respective fields. If you have a cover image, replace the existing contents of the *Cover* field with the source path to this image. Set *UploaderID* to 0. Ignore the *ID*, *Name*, and *Published* fields. Click the "Go" button in the bottom right below the *UploaderID* row and ensure "1 row inserted" appears at the top of the page. Retrieve the *ID* of the newly created row.

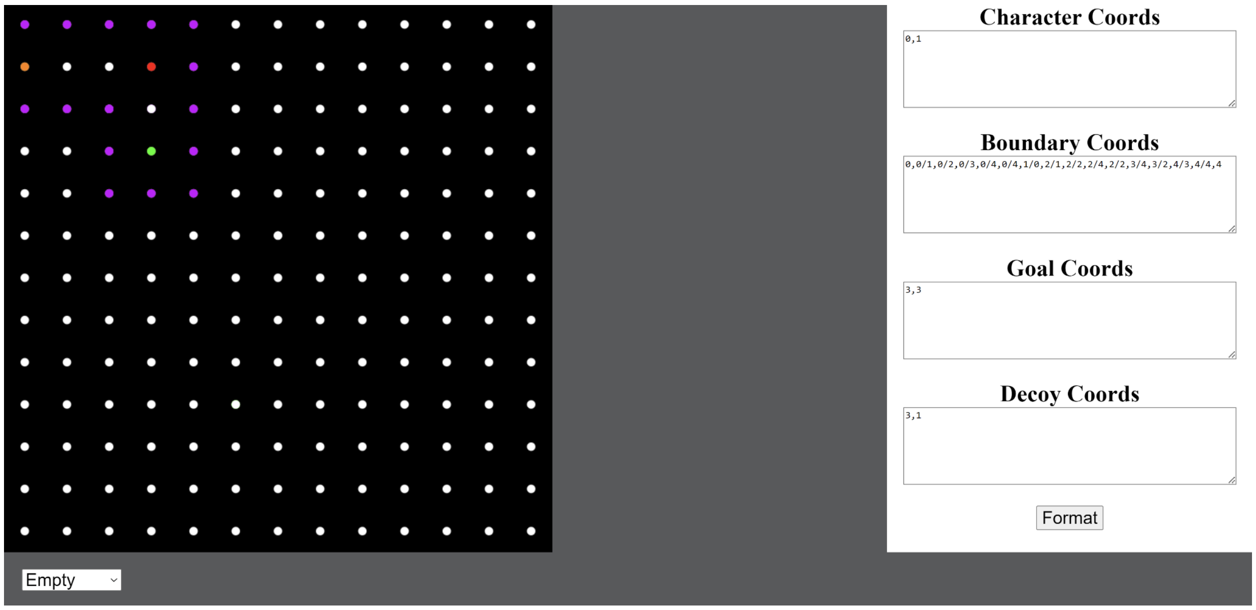
In the *levels* table, create a new row following the aforementioned steps for each level. In each row, enter that level's character image, boundary image, goal image, and background image source paths as well as the instructions in their respective fields. Enter the level number, starting at "1", in the *LvlNum* field. Enter the *ID* retrieved from the newly created *stories* table row as the *StoryID* for all of the rows.

8.1.3 Adding Cutscenes [Optional]

To add cutscenes to your story, save the cutscene images to your story's subdirectory in the *assets* directory. For each cutscene image, add a new row to the *cutscenes* table and enter the image's source path in the *Img* field. Enter the level number in the *LvlNum* field where the cutscene image should appear directly before. As mentioned in Section 6.2 while describing the *cutscenes* table, some rows may have a *LvlNum* of "0" or one past the last level, denoting cutscenes to show before the first level is started or after the last level is completed, respectively. For each group of cutscenes with the same *LvlNum* value, *CutscnNum* defines the order that the cutscene images are displayed, starting at "1". Enter the *ID* retrieved from the newly created stories table row as the *StoryID* for all of the rows.

8.1.4 *Creating Mazes*

Figure 7. Maze Creation Interface


Character Coords

```
0,1
```

Boundary Coords

```
0,0/1,0/2,0/3,0/4,0/4,1/0,2/1,2/2,2/4,2/2,3/4,3/2,4/3,4/4,4
```

Goal Coords

```
3,3
```

Decoy Coords

```
3,1
```

To create a maze, navigate to https://codeastory.utk.edu/pages/manualGeneration.php. The interface in Fig. 7 will appear. Use the dropdown in the bottom left to choose an element, then click on a grid unit to place that element. Do this for each of the elements. The orange dot is the character, purple dots are the boundaries, green dot is the goal, and red dot(s) is the decoy goal(s). Once all elements have been placed, click the "Format" button on the right to generate the coordinates for the elements. Save the coordinates and repeat for each level of your story. Refer to Section 6.2's description of the *mazes* table for an explanation of the coordinates' format.

Additionally, you will need to generate different mazes for each level if you are adding multiple difficulties (i.e., easy, medium, or hard). For example, if you want to offer all three difficulties, you will need to create $3 \times (\#\ of\ levels)$ mazes. Note: Only one difficulty is required.

8.1.5 *Adding Mazes to the Database*

In the *mazes* table, create a new row for each maze. In each row, enter that maze's character, boundary, goal, and decoy coordinates from Section 8.1.4 in their respective fields. For each decoy coordinate, enter the level number of the goal that the decoy should be. Refer to Section 6.2's description of the *mazes* table for an explanation of the *DecoyImgs* and *DecoyCoords* correlation.

Choose the maximum number of code blocks to allow students to use on this level by entering either "Infinity" or a finite number in the *BlockCap* field. Enter the level number, starting at "1", in the *LvlNum* field. Enter the *ID* retrieved from the newly created stories table row as the *StoryID* for all of the rows. Enter the difficulty (i.e., "0" for easy, "1" for medium, and "2" for hard) that the row/level belongs to in the *Difficulty* field.

If you do not wish to create and add new mazes, an alternative to Section 8.1.4 and the aforementioned steps in Section 8.1.5 is to reuse existing mazes from other stories. Simply copy the rows from the other story's *mazes* table entries and change each row's *StoryID* to the *ID* retrieved from your newly created *stories* table row. Ensure that you have the correct number rows (i.e., levels) for your story, as determined by the number of levels created in Section 8.1.1 and 8.1.2.

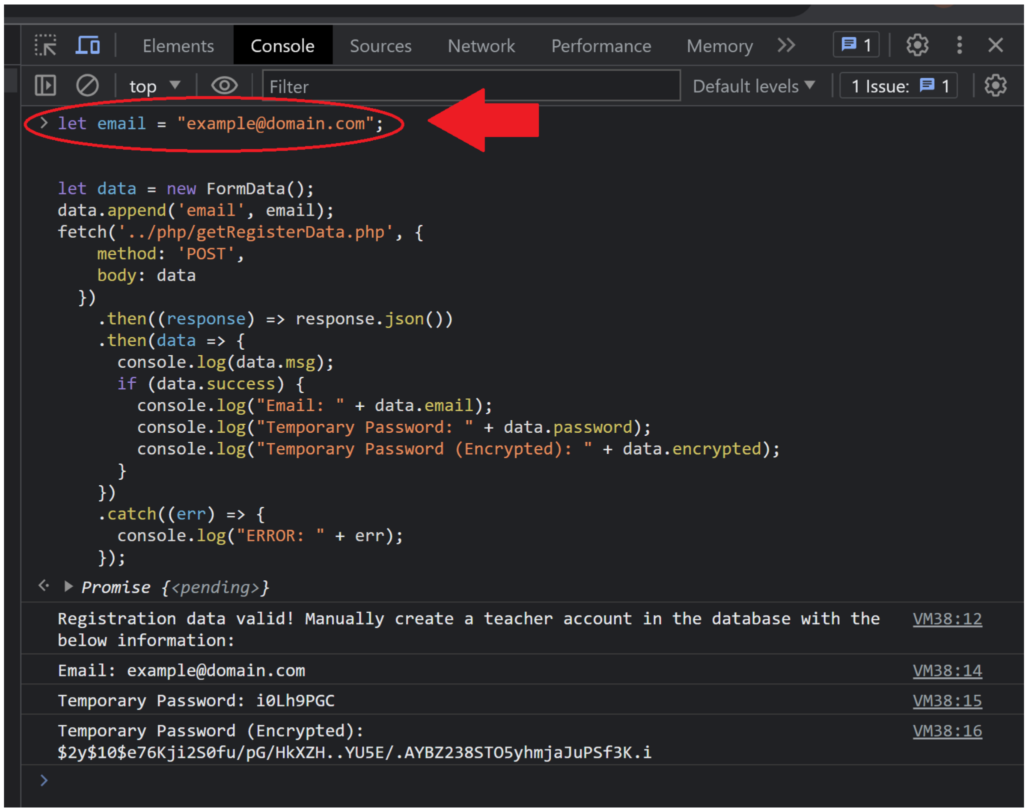
8.1.6 Publishing

With the story components and mazes added to the database, the story is now ready for teachers to use. In the *stories* table, change the *Published* column value for your story's row to "1". After doing so, the story will automatically appear in the generator interface (Fig. 4) for teachers to access.

If you encounter any issues, please reference the existing stories' implementation to identify any potential inconsistencies.

8.2 Creating a Teacher Account

Figure 8. Teacher Registration Snippet in Console



```
> let email = "example@domain.com";

  let data = new FormData();
  data.append('email', email);
  fetch('../php/getRegisterData.php', {
      method: 'POST',
      body: data
    })
      .then((response) => response.json())
      .then(data => {
        console.log(data.msg);
        if (data.success) {
          console.log("Email: " + data.email);
          console.log("Temporary Password: " + data.password);
          console.log("Temporary Password (Encrypted): " + data.encrypted);
        }
      })
      .catch((err) => {
        console.log("ERROR: " + err);
      });
< ▶ Promise {<pending>}
  Registration data valid! Manually create a teacher account in the database with the   VM38:12
  below information:
  Email: example@domain.com                                                             VM38:14
  Temporary Password: i0Lh9PGC                                                          VM38:15
  Temporary Password (Encrypted):                                                       VM38:16
  $2y$10$e76Kji2S0fu/pG/HkXZH..YU5E/.AYBZ238STO5yhmjaJuPSf3K.i
>
```

Code a Story does not have an interface to manage teacher accounts, as mentioned in Section 6.2 while explaining the *teachers* table. Instead, you must manually create a new row in the *teachers* database table.

To create a new teacher account, follow the steps below:

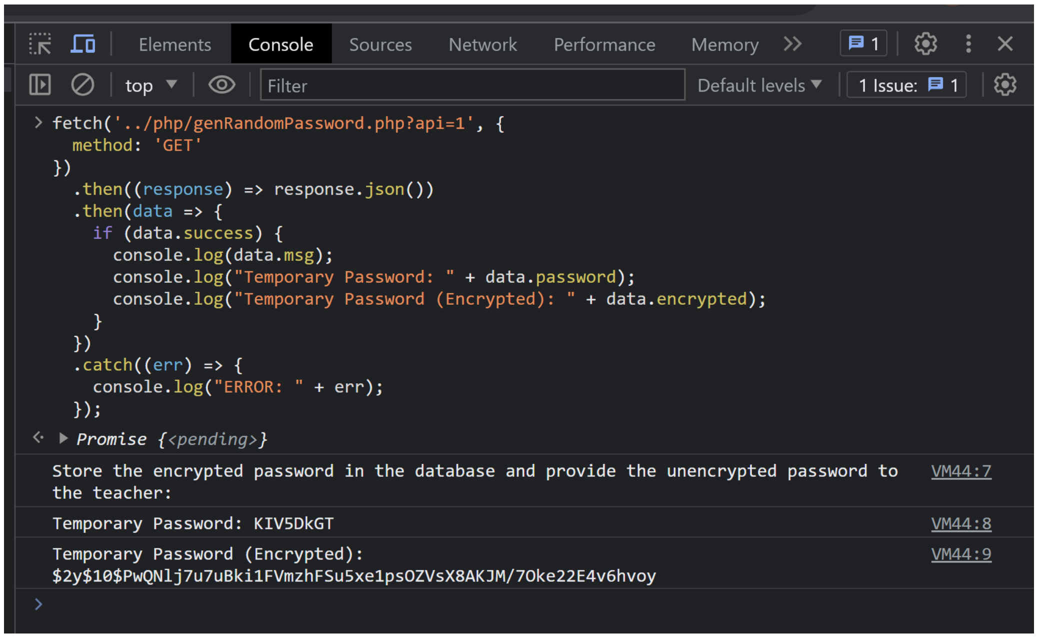
1. Retrieve the teacher's email, first/last name, and school.
2. Navigate to https://codeastory.utk.edu.
3. Open your browser's console. Note: For Chrome, Firefox, and Edge, you can use the "CTRL+SHIFT+J" shortcut for Windows and "COMMAND+SHIFT+J" for Mac.
4. Copy and paste the snippet code from Section 10.1 into the console, setting "email" to the teacher's email (Fig. 8). Press the Enter key.
5. The console will output an encrypted and unencrypted random temporary password if the teacher's email is valid (i.e., not already registered).
6. Access the production database (Section 6.1) and navigate to the *teachers* table.
7. Click the "Insert" tab at the top of the page.

8. Enter the teacher’s email, first/last name, and school into the corresponding fields. Enter the encrypted password from Step 5 into the “Password” field. Ignore the “ID,” “TempPwd,” and “IsAdmin” fields.
9. Click the “Go” button in the bottom right below the “IsAdmin” row.
10. If “1 row inserted” appears at the top of the page, you have successfully created a teacher account!
11. Send the teacher the unencrypted password from Step 5 to log in.

Note: Upon the teacher's first login with the temporary password from Step 5, they will be prompted to create a new password. This way, the Code a Story team does not know the teacher's password.

8.3 Resetting a Teacher’s Password

Figure 9. Teacher Password Reset Snippet in Console



```
> fetch('../php/genRandomPassword.php?api=1', {
    method: 'GET'
  })
    .then((response) => response.json())
    .then(data => {
      if (data.success) {
        console.log(data.msg);
        console.log("Temporary Password: " + data.password);
        console.log("Temporary Password (Encrypted): " + data.encrypted);
      }
    })
    .catch((err) => {
      console.log("ERROR: " + err);
    });
<· ▶ Promise {<pending>}
  Store the encrypted password in the database and provide the unencrypted password to    VM44:7
  the teacher:
  Temporary Password: KIV5DkGT                                                            VM44:8
  Temporary Password (Encrypted):                                                         VM44:9
  $2y$10$PwQNlj7u7uBki1FVmzhFSu5xe1psOZVsX8AKJM/7Oke22E4v6hvoy
>
```

Code a Story does not have an interface to manage teacher accounts, as mentioned in Section 6.2 while explaining the teachers table. Instead, you must manually reset the password in the *teachers* database table.

To reset a teacher’s password, follow the steps below:

1. Navigate to https://codeastory.utk.edu.
2. Open your browser’s console. Note: For Chrome, Firefox, and Edge, you can use the “CTRL+SHIFT+J” shortcut for Windows and “COMMAND+SHIFT+J” for Mac.

3. Copy and paste the snippet code from Section 10.2 into the console (Fig. 9). Press the Enter key.
4. The console will output an encrypted and unencrypted random temporary password.
5. Access the production database (Section 6.1) and navigate to the teachers table.
6. Find the teacher's row and click the associated "Edit" button.
7. Replace the existing contents of the "Password" field with the encrypted password from Step 4.
8. Set the "TempPwd" field to "1".
9. Click the "Go" button in the bottom right below the "IsAdmin" row.
10. If "1 row affected" appears at the top of the page, you have successfully created a teacher account!
11. Send the teacher the unencrypted password from Step 4 to log in.

Note: Upon the teacher's first login with the temporary password from Step 4, they will be prompted to create a new password. This way, the Code a Story team does not know the teacher's password.

Future Considerations

9.1 Security Vulnerabilities

Due to a lack of cybersecurity knowledge during development, Code a Story has several security vulnerabilities to be aware of and potentially resolve in the future.

Code a Story is vulnerable to cross-site scripting (XSS) attacks primarily due to the use of `innerHTML` throughout the JavaScript code. A hacker can inject malicious code into Code a Story's URL, share it with others, and trigger an alert that looks like it is from Code a Story but is actually from the hacker. This is only one example of the potential consequences. Learn more at https://portswigger.net/web-security/cross-site-scripting.

As mentioned in Section 6.2, students' passwords are unencrypted so that teachers can look them up in case students mistype while setting their password or forget their password. Gaining access to a student's account only compromises the student's progression in a maze (i.e., doing their homework for them), so it is not high risk; however, if the student uses the same password for other applications/services, then exposing this password can compromise their other accounts, which can be very consequential. Instead, students' passwords should be encrypted and teachers should only have the ability to reset the students' passwords instead of viewing them.

There are likely other vulnerabilities that are currently unknown, so it is important to consult an individual with more cybersecurity experience if Code a Story scales.

9.2 Copyright Issues

Code a Story may encounter copyright issues due to the use of images from *The Very Hungry Caterpillar* and *Green Eggs and Ham*. It is important to address these issues before scaling beyond academic and research purposes.

Appendix

10.1 Teacher Registration Snippet

```
let email = "example@domain.com";

let data = new FormData();
data.append('email', email);
fetch('../php/getRegisterData.php', {
    method: 'POST',
    body: data
  })
    .then((response) => response.json())
    .then(data => {
      console.log(data.msg);
      if (data.success) {
        console.log("Email: " + data.email);
        console.log("Temporary Password: " + data.password);
        console.log("Temporary Password (Encrypted): " + data.encrypted);
      }
    })
    .catch((err) => {
      console.log("ERROR: " + err);
    });
```

10.2 Teacher Password Reset Snippet

```
fetch('../php/genRandomPassword.php?api=1', {
  method: 'GET'
})
  .then((response) => response.json())
  .then(data => {
    if (data.success) {
      console.log(data.msg);
      console.log("Temporary Password: " + data.password);
      console.log("Temporary Password (Encrypted): " + data.encrypted);
    }
  })
  .catch((err) => {
    console.log("ERROR: " + err);
  });
```

10.3 Acknowledgment

This documentation was written by Jovan Yoshioka in collaboration with Dr. Amir Sadovnik. Thank you to the entire Code a Story / Computer Science for Appalachia (CSA) team for their continued support throughout the development of this project.