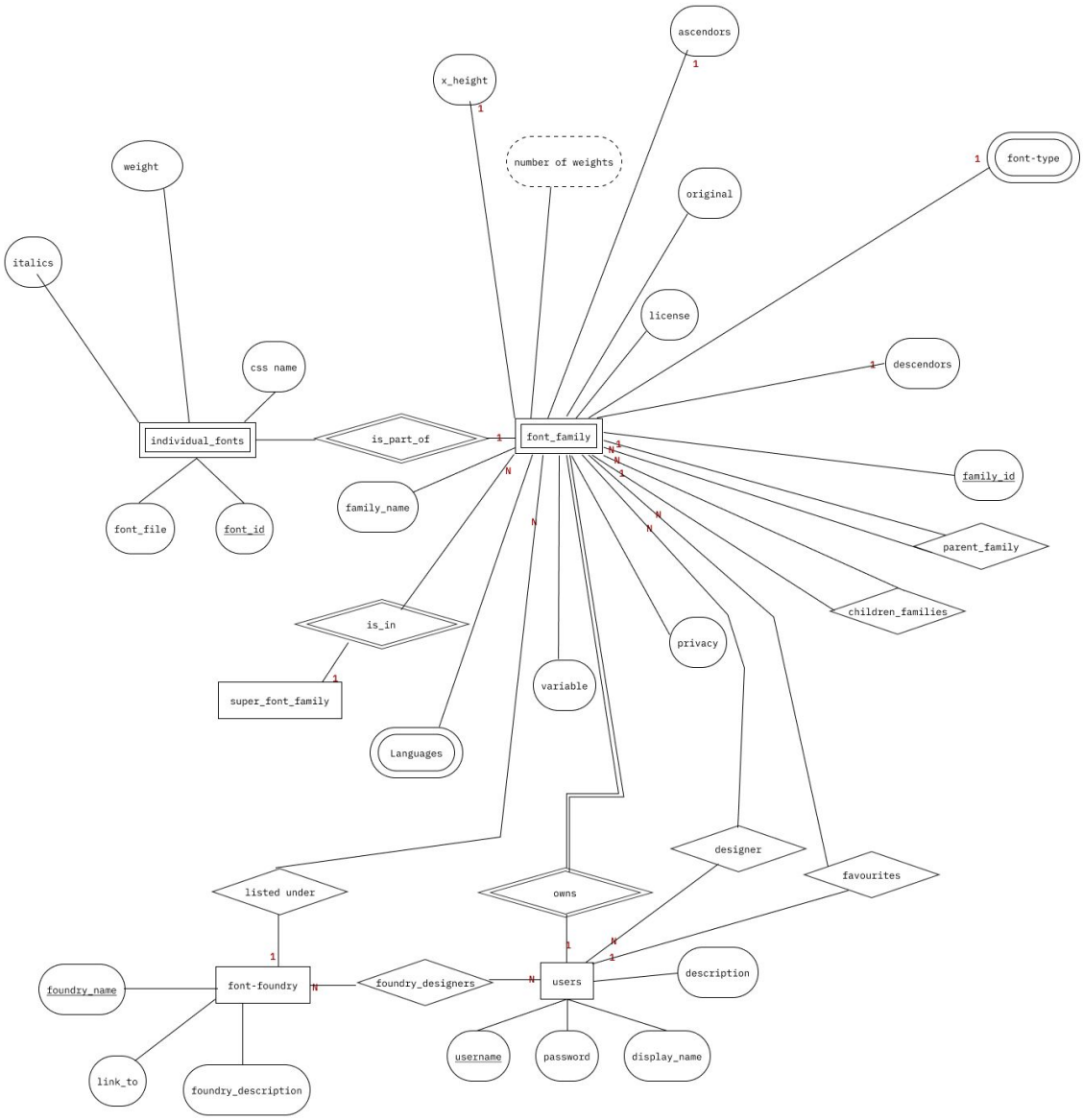
PA2 Report

Ethan Ma #301330262 | Josh Fernandez 301246300 | Aliasger Rasheed 301351886

IMPORTANT: When running the file, please run it under http://localhost/josh_fernandez/iat352_beak-and-spur/build/index.php

Our team created our application in the build folder. The web application will not look or function as intended if the index is not accessed within the build file.

1. Database design - explain how you designed your database (what are the design decisions) + ER diagram



The database was designed in a way to accommodate easy access to each font-family, which made querying for font-attributes simple. Each font comes with certain characteristics that would be needed for filtering: such as font foundries, x height, y height. The ER diagram is primed to easily access font-attributes, it was a large priority when designing it.

Conceptual Schema

Entity sets and relationship sets

Name	type	cardinality	between
ascendors	Strong Entity Set		
descendors	Strong Entity Set		
font_foundries	Strong Entity Set		
font_type	Strong Entity Set		
members	Strong Entity Set		
super_font_families	Strong Entity Set		
weights	Strong Entity Set		
x_heights	Strong Entity Set		
font_families	Weak Entity Set		
individual_fonts	Weak Entity Set		
is_ascendor	Relationship	1 → N	ascendor → font_family
is_descendor	Relationship	1 → N	descendor → font_family
is_in_super_font_far	Relationship	1 → N	super_font_family → font_family
is_type	Relationship	1 → N	font_family → font_type
is_weight	Relationship	1 → N	weight → individual_fonts
is_x_height	Relationship	1 → N	x_height → font_family
listed_under	Relationship	1 → N	font_foundry → font_family
works_for	Relationship	N → M	members → font_foundry
is_part_of	Weak Relationship	1 → N	font_family → individual_fonts
owned_by	Weak Relationship	1 → N	members → font_family

+ New

Members

attributes	attribute type	description
description	simple	
username	primary key	
display_name	simple	
password	simple	
font_family	foreign key	

+ New

COUNT 5

font_family

Attributes	Attribute Type	description
privacy	simple	
css_name	simple	
children_families	foreign key, multivalue	foreign keys of children families
languages	multivalue, simple	
family_id	primary key	
no_of_weights	derived attribute	
family_id	foreign key	foreign key of parent family
font_id	foreign key	primary key of individual fonts
super_family	foreign key	primary key of super font family
font_type	multivalue, foreign key	
descendors	enum, simple	
ascendors	enum, simple	
x_height	enum, simple	

individual_fonts

Attributes	Attribute Type	description
font_file	simple	
family_id	foreign key	
italics	foreign key	
weight	foreign key	

+ New

COUNT 4

font_foundry

Attributes	Attribute Type	description
foundry_name	primary key	
link_to_foundry	simple	website link
foundry_description	simple	
super_family	foreign key	

+ New

COUNT 4

font_type

Attributes	Attribute Type	descripti...
font_type	primary key	

+ New

COUNT 2

Strong entity
– all attributes of strong entity will be attributes of relation schema.

M-to-M relationship
– separate table need to be created with the primary keys of all participating strong entity sets.

1-to-M relationship
– primary key of one side entity is included as foreign key in many side entity set.

reduction of relational schema + Add a view

Type	ER Component	Column
Strong entity set	font_foundries	font_foundries(foundry_name, link_foundry, foundry_desc)
Strong entity set	members	members(username, mem_desc, password, display_name)
Strong entity set	font_families	font_families(family_id, privacy, languages, family_id[children], family_id[parent], super_family)
Strong entity set	font_types	font_types(type)
Strong entity set	super_families	super_families(super_family, super_desc)
Weak entity set	individual_fonts	individual_fonts(family_id, weights, italics, font_file)
One-to-many relationship	is_ascendor	font_families(family_id, privacy, languages, family_id[children], family_id[parent], super_family, ascendor)
One-to-many relationship	is_descendor	font_families(family_id, privacy, languages, family_id[children], family_id[parent], super_family, ascendor, descendor)
One-to-many relationship	is_height	font_families(family_id, privacy, languages, family_id[children], family_id[parent], super_family, ascendor, descendor, x_height)
many-many relationship	ownership	ownerships(username, font_family_id)
many-many relationship	favourites	favourites(username, family_id)
many-many relationship	foundry_employees	foundry_employees(foundry_name, username)
many-many relationship	family_types	family_types(family_id, type)
weak relationship	fonts_in_family	no table

+ New

COUNT 14

final relational schema + Add a view

Type	ER Component	Column
Strong entity set	font_foundries	font_foundries(foundry_name, link_foundry, foundry_desc)
Strong entity set	members	members(username, mem_description, password, display_name, profile_img)
Strong entity set	font_families	font_families(family_id, family_name, username [foreign key of uploader], designer [foreign key of designer], privacy [boolean], font_type[SET], variable[boolean], languages [SET], family_id[parent], super_family [foreign key of super family], original [boolean], foundry_name[foreign key of foundry font is listed under], ascendor [ENUM], descendor [ENUM], x_height[ENUM], licence[type of font licensnce. default is SIL Open Font License]
Strong entity set	super_families	super_families(super_family, super_desc)
Weak entity set	individual_fonts	individual_fonts(font_id[primary key], family_id[foreign family key], weights[ENUM], italics[ENUM], width[ENUM], font_file, css_name)
many-many relationship	ownerships	ownerships(username, font_family_id)
many-many relationship	favourites	favourites(username, family_id)
many-many relationship	foundry_designers	foundry_employees(username, foundry_name)

2. Database connectivity code

The code in-charge of database connectivity connects to the `josh_fernandez` database. The code is responsible for initializing and validating user input or form fields successfully, writing appropriate SELECT, INSERT, and UPDATE queries for them, and showing results to the user if necessary. Connecting to the database is important for registering and validating members and showing fonts and font families dynamically.

The code follows the same step-by-step procedure as Helmine's code except different code snippets are extracted as functions for ease-of-use, improved readability, and lesser chance of errors.

Opening and closing the database are performed by the functions inside the `php-backend/helpers/db-connection-methods.php` script. All database connectivity code can be found inside the `php-backend` folder as `php-backend/process-<name-of-task>-form.php`. Similarly, all form field initialization and printing of results is in the `<name-of-task>-complete.php` in the root folder.

Helper functions and methods are located in the `php-backend/helpers` folder and they help in making the code more readable and less prone to errors so that code is not being repeated as much. These include:

- `db-connection-methods.php`: for opening and closing the database
- `file-upload-methods.php`: for uploading a file, like a user's profile image when they want to update their profile
- `form-analysis-methods.php`: for initializing, validating, and preparing form fields properly
- `query-append-methods.php`: for appending attributes to a string, ready to be added to a SQL query
- `query-perform-methods.php`: where the different queries live

In the `process-<name-of-task>-form.php`, we use helper methods to validate field values, construct queries, and perform queries. This level of abstraction helps with readability and understanding the code.

In addition to updating a user's profile, the update profile page pre-populates the logged-in user's information in the form fields. The database connectivity code for it can be found inside `php-backend/prepare-update-form.php`.

3. Secure authentication handling

Upon registration, a new member's password is encrypted and stored in the members database using PHP's `password_hash` function. Use of this function can be found in `register-complete.php`, just before sending it to `php-backend/process-reg-form.php` where it will be stored as the new member's password.

When logging in, PHP's `password_verify` function is used to compare between the password stored in the database and what the user typed. As with `password_hash`, use

of this function can be found in `login-complete.php`, after retrieving the member's password hash value.

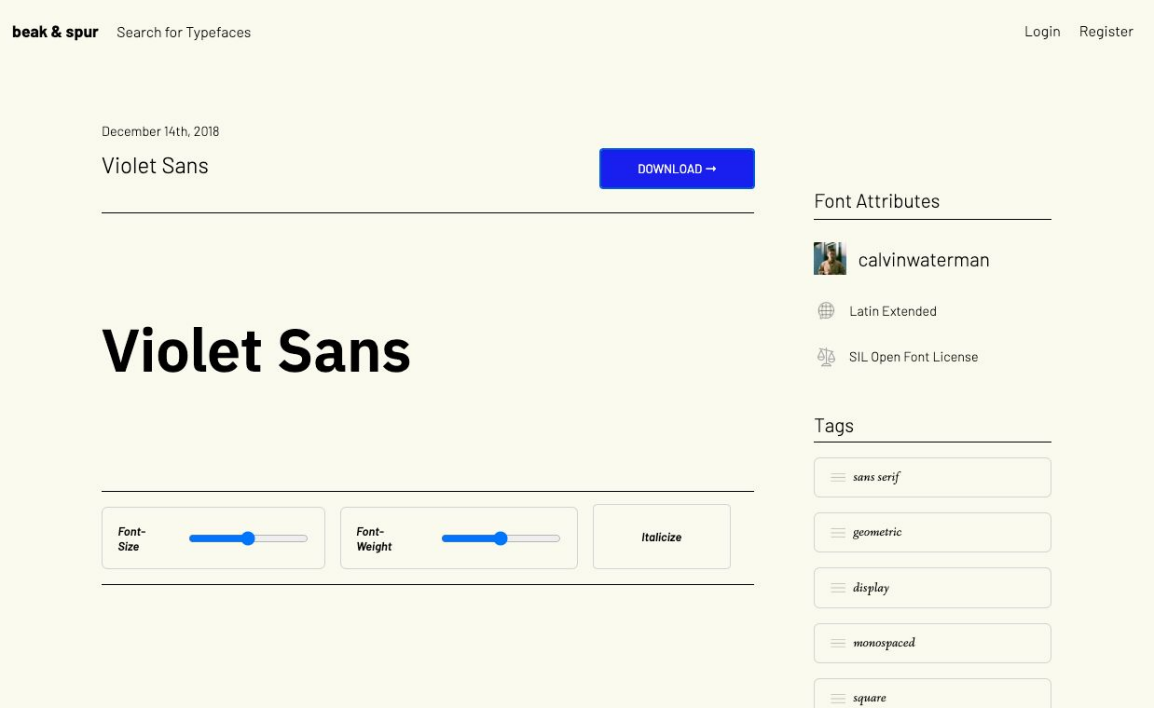
We also use a session variable called `session_user` to show members-only pages, such as the update profile page. Starting a session can be found in `php-backend/set-header.php` while ending a session (and letting the member log-out) can be found in `php-backend/log-out.php`.

Properly setting pages for members and visitors and further verification of form fields will be improved during PA3.

4. **Visitors - explain what functionality your web app has for visitors and how this has been implemented**

Visitors have a few options when navigating Beak and Spur.

1. Visitors can browse fonts on our web-application. Through the mainpage `index.php`, visitors can view a variety of different fonts for their choosing. If visitors are intrigued by the font-family and want to dive into characteristics of the font, they can view them in the `font-family-page.php`. This gives users characteristics such as font-designer, supported languages, and licence type. However, because the users aren't registered with the website, they are not able



to download the fonts. In our next milestone (PA3), we are planning on loading in ttf.

files from our database and displaying it on this page so users can interact and type with the fonts.

2. If visitors are unhappy with the selection displayed on the `index.php` page, users can search for specific fonts in the `filters.php page`. Within the page, visitors can choose between primary and secondary filters to search fonts of that characteristic.
   a. If a visitor needs a font with three types of font characteristics of their choosing, they can click the filters. We chose to accommodate many filters to anticipate scaling of our website. Therefore, our database isn't populated with many fonts, because there are no users on our site.

   b. However, if visitors come to our website with a specific font in mind, they can use our search function to scavenge the database for the font. This search bar runs a query within our database and attempts to reference everything that the visitor types.

5. Member registration and login

A visitor can register to Beak & Spur through the `register.php` page. The form asks the visitor for a username, their email address, and their password. Upon clicking the "Register" button, the username and email address are being checked through PHP's `preg_match` functions used in `validateUsername` and `validateEmail` located in `php-backend/helpers/form-analysis-methods.php`. Further, the password is being hashed. (See secure authentication handling.) These fields are being added to the `members` table using an INSERT query.

A visitor can also login to the site through the `login.php` page, which asks them for their username and password. These fields are being checked through a SELECT query. Upon successful login, a new session is initialized. Upon logging out, the session ends.