Browse Matches
Final Report and Technical Specifications

A Senior Project in Computer Science at Yale University
for Elle Buellesbach
Advisor: Benedict Brown

May 1st 2019
Abstract:

In reconstructing artifacts which have one primary planar surface, such as frescos, digital tools can aid workers in determining which fragments match together. Existing programs allow for the digitalization of fragments, as well as the creation of proposed matches based on algorithms using computerized visual analysis. However, often these proposed matches are incorrect or multiple proposed matches are made for the same fragment. Therefore, a layer of human approval is necessary. An existing program, which runs locally using Qt 4 displays images of possible matches and allows the user to annotate and approve them. However, this program is both outdated and has limited functionality due to its lack of portability.

This project is a web interface to allow for the annotation and approval of proposed matches between pre-scanned and processed fragments. It connects a Vue.js frontend and a Node.js server. It also connects to a remote mysql database, allowing multiple users to comment on and approve fragments and to view each other's responses. The web interface also involves functionality to query the database, and is designed such that the web interface is flexible enough to work with another dataset with only minor query changes. Additionally, the interface is screen size responsive and key-stroke navigable improving portability.

To Download and Run Code:

All code is located at: https://github.com/ebuellesbach/BrowseMatches.

The files consist of a server built with Node.js and frontend built with Vue.js. To recompile the Vue frontend and to run the node server locally:

```
cd backend
npm run make
```
**User Interface:**

Welcome Page (Figure 1):
- The home page of the application is accessible without login
- A click on ‘BrowseMatches’ in the navigation bar, will return to this page.

Login (Figure 2):
- Navigate the login page by clicking ‘login’ on the navigation bar
- Login is done with a user and email combination.

Search (Figure 3):
- Navigate the search page by clicking ‘search’ on the navigation bar
- User is automatically redirected here after login
- The blue hyperlink ‘Search’ will redirect to display search results
- Search criteria:
  - If all are blank, it returns all fragments in sort order of probability descending
  - The Fragment IDs can be searched for an exact match or if ‘LIKE’ is clicked, any results with the search query as a substring will be returned
Results Display

- Each match corresponds to one tile on the display page.
- The number of tiles per page (and thus the number of pages) dynamically responds to
  screen size
- The ‘Larger’ and ‘Smaller’ buttons can be used to increase and decrease the size of the
  tile (between predefined constraints)
- On each tile, are three boxes (green, yellow, and red) which correspond to three
  possible user evaluations for each match (‘yes’, ‘maybe’, ‘no’). The box displays the total
  vote count for all users. The star reflects the vote choice of the logged-in user. In the
  figure above, the first match as one yes vote and one maybe vote. The current user
  voted maybe.
- Clicking on a tile will open more information (see Match Page below)
- Key-stroke commands:
  - All arrow keys can be used to navigate the grid. If on the last tile of a page, the
    arrow key to the right will go to the next page. If on the first tile of a page, the
    left arrow key will go to the prior page. The selected page will appear
    surrounded by a grey box
  - + (plus sign) will increase the tile size
  - - (minus sign) will decrease the tile size
  - 1, 2, and 3 can be used to vote on the selected tile
  - Space bar will open the modal with more information for the selected tile
Match Page (Figure 5):

- When a tile is selected, a modal appears with more information.
- The layout of this modal changes depending on screen width.
- A larger image of the match is displayed, along with the match ID as found in the database.
- The IDs for the target fragment and the source fragment are listed. Their IDs (in green) are links which will lead back to the Display Page, however, with only matches involving that selected fragment.
- The vote boxes are similar to the tile page, key stroke controls also work for voting here.
- Below the votes, is a box with the current user’s comment. By pressing the comment button on the right hand size, this will become a text area to change the comment. By pressing the button again this comment will be saved.
- Below the image, is more data about the fragment as stored in the database.
- Finally, a list of comments for all other users concerning this fragment is displayed.
Design Details:

High level file organization:

/BrowseMatches
  /backend
    - Contains a Node.js server
    - This server serves the compiled Vue frontend, as well as manages database queries for the frontend
  /frontend
    - Contains files that compile into the Vue frontend
  /originalcode
    - Contains files from the original Browse Matches
    - Contains the code which creates the SQL database (useful to understand the database organization)

The backend:

config.js
  - Contains the configuration details for connecting to the mysql database

server.js
  - Organized into 5 sections:
    - Database Connection:
      - Establishes a connection to the database
    - App Configuration:
      - Sets the configuration details for the app including host and port
      - Connects the static files of the Vue frontend to the server
    - Authentication:
      - POST /login: queries the database for a user and email, and returns successfully if a match exists.
      - A password function was not implemented so as to maintain the existing database structure. To maintain security, there is no registration functionality from the web page, instead a new user can be added by directly accessing the database.
    - Frontend Routing:
      - Servers the compiled Vue frontend
    - Database Routes:
      - function convertData:
- Takes data about the matches from the SQL query and converts it into the format used by the Vue Frontend
- Format of data returned -- a list where each entry is a dictionary:

```
{
  id: (The ID of the match)
  imageUrl: (full URL, as served by a separate image generating server)
  tblData: (Raw data from SQL table)
  user: {
    vote: (The current user’s evaluation: ["yes"/"no"/"maybe")
    comment: (The current user’s comment)
  }
  otherComments: [(A list of all other comments)]
  stats: { (The tally of all user’s evaluations)
    yes:
    no:
    maybe:
  }
  confirmed: (If this piece has been fully confirmed [true/false])
}
```

- GET /data
  - Returns a list of match data (paginated) based on a given query
  - Query parameters:
    - Uid: (The current user’s ID - REQUIRED)
    - perPage: (# of matches per page of data - REQUIRED)
    - page: (Current page of data requested - REQUIRED)
    - tgt: (Filter search results based on given string for target fragment ID)
    - src: (Filter search results based on given string for source fragment ID)
    - tgtLike: (Search for results based on substrings containing tgt)
    - srcLike: (Search for results based on substrings containing src)
    - evalu: (Filter for matches with selected evaluation)
    - txt: (Filter for matches with comments including substrings of txt)
    - sortType: (Sort matches based on ["probability"/"error"])
    - sortOrder: (Sort matches ["ascending"/"descending"])

- GET /data/count
  - Returns the total number of entries in the database for a given query
  (based on same parameters as above but without pagination)

- GET /data/:matchId
  - Returns the data for a given match ID

- PUT /data/:matchId
  - Updates the database for a given matchID
  - Parameters:
    - data:
      - txt: (New comment)
- **evalu**: (New evaluation)
- **uid**: (User making the change)

The frontend:

- `src/App.vue`
  - The main Vue file, which includes the router-view
- `src/main.js`
  - Mounts the Vue instance with the router including a listener for changes in the window height and width
- `src/router.js`
  - Creates a Vue router, with protected pages where the beforeEach function checks for authentication
- `src/components/`
  - **Login**: A separate component to handle login, linked through router
  - **Welcome**: The welcome page, linked through router
  - **Overview**: A separate component to handle the search, linked through router.

Generates a search that will go to the Display

- The components that make up the display page are nested as follows:
  - `DisplayWrapper → Display → Page → MatchTile`
  - The `DisplayWrapper` is layer for data control. Display contains the page navigation bar and size buttons. Page organizes the grid of MatchTiles
Moving Forward:

There are several possible avenues to continue work on this project. Some possible suggestions include:

- Improvements to the SQL database so the convertData function does not need to loop through the data to compile all user information.

- Linking in functionality to mark a match as confirmed: allowing this to update the database and remove other possible matches for that fragment. This functionality exists in the older code, but it was not ported to this server.

- General, streamline of the Vue instance. There are areas for elimination of repetition and better data management.