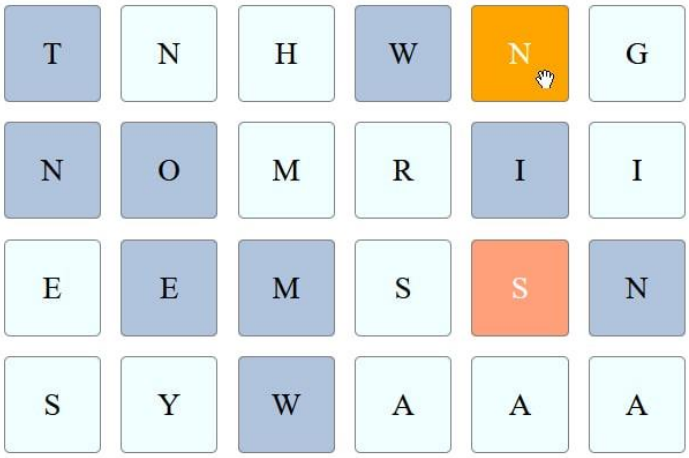
Web Applications I – Exam # 5 (deadline 2022-01-19 at 23:59)

"Crucipuzzle"

⚠ PRELIMINARY VERSION – THE FINAL VERSION WILL BE UPDATED ACCORDING TO THE QUESTIONS ON SLACK #exam5-crucipuzzle ⚠

Design and implement a web application to generate and solve a word searching puzzle, known as "CruciPuzzle". The game consists of a rectangular grid of alphabetic letters, where the player must discover embedded words in any of the 8 directions (2 horizontal, 2 vertical and 4 diagonals).



Example grid[1], with the words TOM, WE, NEW and WIN already selected (and highlighted), and the user is selecting the word SIN (the S has been selected as the starting point, and the N is about to be selected as an ending point).

The application must implement the following specifications.

A player may start a new game, by selecting the difficulty from 5 different difficulty levels; each level consists of a different grid size (difficulty 1 is 4 x 6, difficulty 5 is 20 x 30). The grid is filled with upper case letters, chosen randomly by the server, and the probability of choosing a letter is proportional to their frequency in the English language[2].

The letters must be graphically displayed to the user as a rectangular grid, so that it is possible to easily identify row, columns and diagonals. Each game has a fixed duration of 60 seconds.

The user may identify one or more words in the grid, starting from any letter and going towards any of the 8 possible directions. To select a word, the user will click on the *first* and *last* letters of the word. If the click positions are compatible (on the same row/column/diagonal), and if the letters connecting the

[1] you are NOT required to replicate this design, you are totally free to develop you own (and better-looking) graphical design

[2] You may find the pre-computed frequencies from Wikipedia https://en.wikipedia.org/wiki/Letter_frequency or other websites (e.g., http://pi.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html)

starting and ending positions compose a valid word in English (that will be checked against a dictionary[3] of valid words), then the letters are visually "marked", and the user will get as many points as the length of the word. In selecting a new word, the user may re-use some letters of previous words.

At the end of each 60-second round, the total score is computed by multiplying the total number of letters of the found words, times the difficulty level (1…5), and it is shown to the user. The users should be able to interrupt the game at any time before the 60 seconds deadline. In this case, the same actions will be taken.

The game may be played in an *anonymous* way (no login required, and the scores are not saved) or in a *personalized* way (the user must login before starting the game, and all scores of all played games are recorded in their personal history).

The application also shows a "hall of fame", in a page accessible without authentication, with the name and score of the 5 top-players.

Project requirements

- The application architecture and source code must be developed by adopting the best practices in software development, in particular those relevant to single-page applications (SPA) using React and HTTP APIs.
- The project must be implemented as a React application, that interacts with an HTTP API implemented in Node+Express. The database must be stored in a SQLite file.
- The communication between client and server must follow the "React Development Proxy" pattern, and React must run in "development" mode.
- The root directory of the project must contain a `README.md` fine, and have two subdirectories (`client` and `server`). The project must be started by running the two commands: "`cd server; nodemon server.js`" and "`cd client; npm start`". A template for the project directories is already available in the exam repository. You may assume that `nodemon` is globally installed.
- The whole project must be submitted on GitHub, on the same repository created by GitHub Classroom.
- The project **must not include** the `node_modules` directories. They will be re-created by running the "`npm install`" command, right after "`git clone`".
- The project may use popular and commonly adopted libraries (for example `day.js`, `react-bootstrap`, etc.), if applicable and useful. Such libraries must be correctly declared in the `package.json` file, so that the `npm install` command might install them.
- User authentication (login) and API access must be implemented with `passport.js` and session cookies. No further protection mechanism is required. The user registration procedure is not requested.
- The project database must be implemented by the student, and must be pre-loaded with *at least three* users, who played at least 5 times.

[3] The word list must be stored in the server, and should NOT be transferred to the client. Such word lists may be found on various websites, such as http://www.mieliestronk.com/wordlist.html, https://github.com/dwyl/english-words, http://www.gwicks.net/dictionaries.htm, …

Contents of the README.md file

The README.md file must contain the following information (a template is available in the project repository). Generally, each information should take no more than 1-2 lines.

1. A list of 'routes' for the React application, with a short description of the purpose of each route
2. A list of the HTTP APIs offered by the server, with a short description of the parameters and o the exchanged objects
3. A list of the database tables, with their purpose
4. A list of the main React components
5. A screenshot of **the page for playing the game, with some words already crossed out**. This screenshot must be embedded in the README by linking an image committed in the repository.
6. Username and password of at least 3 users.

Submission procedure (important!)

To correctly submit the project, you must:

- **Be enrolled** in the exam call.
- **Accept the invitation** on GitHub Classroom, and correctly **associate** your GitHub username with your student ID.
- **Push the project** in the **`main` branch** of the repository created for you by GitHub Classroom. The last commit (the one you wish to be evaluated) must be **tagged** with the tag `final`.

Note: to tag a commit, you may use (fron the terminal) the following commands:

```
# ensure the latest version is committed
git commit -m "...comment..."
git push

# add the 'final' tag and push it
git tag final
git push origin --tags
```

Alternatively, you may insert the tag from GitHub's web interface (follow the link 'Create a new release').

To test your submission, these are the exact commands that the teachers will use to download the project. You may wish to test them on a clean directory:

```
git clone ...yourCloneURL...
cd ...yourProjectDir...
git pull origin main  # just in case the default branch is not main
git checkout -b evaluation final # check out the version tagged with
'final' and create a new branch 'evaluation'
(cd client ; npm install)
(cd server ; npm install)
```

Ensure that all the needed packages are downloaded by the `npm install` commands. Be careful: if some packages are installed globally, on your PC, they might not be listed as dependencies. Always check it in a clean installation.

The project will be tested under Linux: be aware that Linux is case-sensitive for file names, while Windows and macOS are not. Double-check the case of `import` and `require()` statements.