Final Project

The final project is your opportunity to show us all you've learned throughout the course and build something cool in the process. We encourage you to use it as a chance to create something relevant to your interests, such as to support a group or organization you work with or create a tool you would find useful.

Updates

We will update this page with additional information throughout the quarter. Here is a summary of notable changes:

- **Mar 12:** Added instructions for deploying.
- **Mar 10:** Added final submission instructions and template, and a couple more external libraries/tips.
- **Feb 16:** Added milestone template and a couple external libraries
- **Feb 13:** Added starter zip
- **Jan 25:** Initial page: due date for proposal and general guidelines

Due dates

All due dates have a 48-hour grace period.

- Proposal due Tue Feb 8 11:59pm PT
- Midpoint check-in due Thu Feb 24 11:59pm PT
- Final submission due Tue Mar 15 11:59pm PT

Overview and requirements

For the project, you will build a complete web application, using the technologies and skills covered throughout the course. It is very open-ended; we define only a number of high-level requirements that your project needs to hit to receive full credit.

Logistics

Individual: The project must be completed individually; two or more students cannot work jointly to create a single app. However, if you would like to integrate your project with another app or API, including another project, this may be possible, as long as the scope of the project is clearly defined and meets the requirements. For example, two students could each create a frontend page and backend API that all connect to each other. In contrast, one student cannot exclusively build the frontend while another exclusively works on the backend.

Third-party code: External libraries may be useful in performing many tasks, such as creating advanced data visualizations or connecting to external services. You are permitted to use third-party libraries or incorporate external code into your project (licensing and copyright permitting), but that code must not fundamentally change the organization and structure of your application. For example, using a library to create a graph, map, etc. on the page is fine, but you may not use jQuery, Bootstrap, React, or another frontend framework that completely changes how you interact with the DOM and events. You must clearly identify the libraries and external code you use. Such code will not be graded and will not count toward the functionality requirements.

Scope: As you will only have a few weeks to work on the project, it is a good idea to plan for a relatively small project to make sure you will be able to complete something that meets the requirements. You can always set stretch goals to work on if you have time, but in our experience, the end of the quarter tends to creep up on us before we know it.

Collaboration policy: To accommodate the above logistics, we have a section of the collaboration policy specifically for the project. Please review it and reach out to us if you are unsure of how it applies to your situation.

Technical requirements

To receive full credit, your project must meet all of the following requirements:

Functionality

- Frontend HTML and CSS
  - Overall page structure written in HTML (not dynamically generated)
  - Page layout defined in CSS

    - Uses CSS classes and ids
    - Uses Flexbox, font and text styling, and the box model (margin, border, padding)
- Frontend JavaScript
    - Add multiple (2+, i.e. 2 or more) event handlers
    - Create, modify, or remove DOM elements
    - Define multiple (2+) classes and modules
    - Use `fetch` to access your backend
- Backend JavaScript
    - Written in Node and Express
    - Defines multiple (2+) routes, including one GET and one non-GET route
    - Uses a route parameter
    - Reads a JSON request body and returns results in JSON
    - Uses a MongoDB database to store persistent data

Your page only needs to work on modern browsers.

Style

These requirements are the same as the criteria we use to grade assignments:

- Use semantic HTML elements (e.g. rather than making everything a div)
- Write maintainable CSS that takes advantage of inheritance and combining selectors
- Write robust JavaScript that is decomposed and uses modern features and best practices while avoiding obsolete ones
- Use `async`/`await` to handle API and database access

Extra features

While not required, you will receive extra credit and kudos for incorporating some of the full-stack topics we cover toward the end of the quarter. Depending on your project, these may be particularly important to make your app most useful and impactful. These topics include

- Making your app mobile-friendly
- Making sure your app is accessible to people using assistive technology
- Integrating your app with an authentication service (e.g. "Sign in with Google"). (You should not try to implement your own password/authentication system.)

Not Graded

We will not grade your project on aesthetics, originality, or "usefulness." We look forward to seeing your many great ideas, but do not worry about having to make a project that has never been done before or "looks interesting."

Example projects

Here are a few projects that have been done in the past. You may rehash one of these if you wish. Note that some of these projects will be more complex than others.

- A mood tracker (enter how you're feeling each day on a scale of 1-10, shows a visualization of your mood over time)
- Daily journal app (organize notes by day)
- An image gallery where you can add links, titles, and descriptions for images and display them in one place
- A page to visualize air pollution in your local area and let you leave comments about air quality near you
- An online HTML/CSS/JavaScript editor similar to Codepen where you can create, preview, and share projects with others
- A note sharing application where you can write notes and send them to others via a shareable link
- A calendar application for scheduling and tracking events
- A social app for finding friends with common interests, implementing the same "swipe" interface as Tinder
- A personal website where you can display a portfolio, gallery, resume, etc. and where visitors can leave you messages (even better if you make yourself an interface to edit the page easily)

Proposal

The first step of the project is to submit a short proposal. This allows the teaching staff to give you feedback before you get too deep into writing code, as well as confirm the logistics (e.g. collaboration and libraries) relevant to your project.

Download the template for the proposal and fill it in. Your description can be fairly short: perhaps a couple paragraphs. You don't need to "pitch" your idea to us; just tell us what you're thinking of doing.

Once you're done, submit it as `proposal.txt` on Paperless.

Your project needn't be set in stone once you submit the proposal. You may later discover additional libraries or APIs that would be useful, for example. If you are thinking of completely changing your project, though, please notify the teaching staff as soon as possible so we can give you feedback.

Getting started

Here is a starter zip file like the ones given with the assignments. It doesn't contain all that much, except the `npm` dependencies we've been using throughout the course. You don't have to use it, but you will need to set up your project so that we can run `npm install` and `npm start` on your submission.

As a convention, you should update the `package.json` to include your SUNet (email address, not number) in the package name, your name in the author line, and the name of your project in the description.

Milestone

For the milestone, we aren't looking for anything specific; we just want to see what progress you have made and make sure you're thinking about the project. As a rough guideline, we're hoping that you'll have started planning out the layout of one of your pages, as well as identified a few API routes you'll be using.

Download the milestone template and fill it in. Put it in the root of your project (where `package.json` and `server.js` are, and submit the project to Paperless. Note that, to avoid submitting the dependency files, you should **delete the `node_modules` folder before submitting**. Running `npm install` will recreate that folder.

External libraries and tips

Here are a couple examples of external libraries, as well as some general tips/useful tools, that you may find helpful. We'll add more here if we run into things we think may be broadly useful.

- **Data visualization: Chart.js:** This seems like a relatively straightforward way to create some graphs and charts, without too much boilerplate code. See their getting started guide for an example (but please replace their use of `var` :)).
  - The easiest way we've found to include Chart.js is to go to Chart.js's CDNJS page and use the first URL (with `min.js` at the end but without `esm`). Click the "Copy script tag" button and put that tag into your html file.
- **Authentication: Sign in with Google:** This is a simple way to get user accounts in your web app without having to deal with passwords and a number of other security precautions yourself.
  - We've written up some instructions for setting up Google authentication for your app. They're a bit more complicated than we'd have liked, but we've provided the code for the actual security/authentication parts, so hopefully it is manageable. Please post on the forum if you're unsure how some of the steps apply to your project.
- **Storing data across pages: sessionStorage:** If your project uses multiple pages, and you need to store some information (e.g. a logged in user or an API key) on one page and retrieve it in another, this is a clean way to do it.
  - The data you put in `sessionStorage` is accessible from every page on `localhost:1930`, in the same browser tab, until the tab is closed. (There is also localStorage for longer-term storage, but it often requires a bit more cleanup.).
  - You can only store strings, so if you want to store objects, you'll need to `JSON.stringify` and `JSON.parse` them.
  - The examples on that page should be fairly readable. For a list of the methods you can call on `sessionStorage`, see Storage.
- **Working with file uploads: data URLs:** If your project works with files the user uploads, such as images, audio, or video, we recommend handling them in the form of data URLs. These are normal strings (that start with `data:`) that are also valid URLs.
  - You can use the code from `import_export.js` in the assignment 2 starter. Change `readAsText` on line 28 to `readAsDataURL`. This will call your "import" callback function with a data URL.
  - You can send this string to your backend, store it in MongoDB, etc. as normal.

  - The really nice thing about data URLs is that, since they are valid URLs, they can be used as the `src` and `href` attributes of links, images, etc. For example, you could create a new `<img>` element and set its `src` to be a data URL you get from an API call.
  - If you want to create a link to download the file, you can use the `download` attribute of the `<a>` element, e.g. `<a href="YOUR_DATA_URL" download="myfile.png">`.
- You are welcome to use and adapt code from your assignments if you wish.
- If you are looking at using another external library in your frontend, we recommend including it with a `<script>` tag (few libraries out there currently support `import` directly from the browser), since it is rather difficult to make something you install with `npm install` available to the browser using our setup. If you need help integrating an external library, please reach out to us.

Final submission

Notes for the final submission:

- **Our test environment**
  - We will run your submission by doing `npm install` and `npm start`.
  - Please avoid having external (non-Node) backend dependencies such as other programs that need to be installed/running on the machine, etc. If your project depends on the output of another program, please provide some example output that we can use to test your project.
  - Similarly, if your project uses an external API that requires an API key, please include the key in your code if you can (e.g. if you can request the key with a throwaway email address, that may be helpful). If that's not possible, please include some example output and point your project to it so we don't need to request our own key.
  - There is a section in the readme (below) to give us additional info about running your project. We greatly appreciate every effort to make the process as straightforward as possible for us!
- **Initializing the database:** If your project counts on there being some data in the MongoDB database at the start, please include a `mongodb_init.js` script in your porject, as follows:
  1. Create a file `mongodb_init.js` with the following two lines:

     ```
     db = new Mongo().getDB(YOUR_DATABASE_NAME);
     db.dropDatabase():
     ```

     where `YOUR_DATABASE_NAME` is the database name you use in your backend API. (The second line resets this database to ensure we will start from a clean state.)
  2. Add `mongosh` commands after these two lines to insert the initial data. For example,

     ```
     db.users.insertMany([
       { id: "mchang", ... },
       ...more documents...
     ])
     ```

     - To export documents from a collection, you can (from your terminal) run a command like

       ```
       mongosh YOUR_DATABASE_NAME --quiet --eval "db.COLLECTION_NAME.find()" > output.js
       ```

       . This will create a file `output.js` which contains the array of documents in `COLLECTION_NAME`, which you can pass to the `insertMany` in your script.
  3. We should be able to import the data using this script by running (from the terminal) `mongosh mongodb_init.js`. Please test it yourself to confirm that the data loads. (**Reminder:** The script will wipe out the database as its first step, so please make sure you don't have any data you want to keep in there before running the script.)

Now download the README.txt template and fill it in to let us know about your final submission. Then submit it, along with the full project directory (the folder that contains `package.json`) to Paperless. Note that, to avoid submitting the dependency files, you should **delete the node_modules folder before submitting**.

Deploying your final product

We hope that you will want to share your work with the world when you're done. See our instructions on deploying your project on Heroku.

To be clear, you are **not** required to deploy your project to receive credit for it. We will evaluate the work you have submitted on Paperless. If you would like to include a URL to your working project in your README, though, we'd love to see it!