ChatGPT

Take-Home Project: AI-Powered Alcohol Label Verification App

Overview

In this take-home assignment, you will develop a **full-stack web application** that simulates a simplified version of the Alcohol and Tobacco Tax and Trade Bureau (TTB) label approval process. TTB agents review alcohol beverage labels to ensure the information on the label matches the information submitted in the application form. Your task is to create an app that **accepts an alcohol label image and a form with product information, then uses AI to verify if the image's content matches the form's data**. This project will test your ability to follow instructions, your coding skills across the stack, and your self-direction and judgment in making reasonable implementation choices.

Key Objectives:
- Build a web UI for inputting form data (some key fields from a TTB label application) and uploading an image of an alcohol label.
- Implement a backend service that uses basic AI (e.g. OCR or image analysis) to extract information from the label image.
- Compare the extracted label information with the form inputs and determine if they match (just like a TTB agent would verify a label).
- Display clear results to the user indicating success (if everything matches) or failure (if discrepancies are found), along with details on what was verified.
- Ensure the app handles various scenarios (matching info, mismatched info, missing fields on the label, unreadable image, etc.) gracefully and clearly.

This project is intended to be completed in about **one day**, so scope your solution accordingly. We are more interested in your approach, code quality, and how you handle the requirements than in a huge, overly complex system. Keep it as simple as possible, but feel free to note how you would extend it if given more time.

Project Requirements

1. Form Input (Simplified TTB Application Form)

Create a simplified web form for the user to input key information about the alcohol product. These fields represent the information that a producer would submit to TTB for label approval (we have based them on the TTB guidelines [1]):

- **Brand Name** – The brand under which the product is sold (e.g. *Old Tom Distillery*).

- **Product Class/Type** – The general class or type of the beverage. For distilled spirits this could be the class/type designation (e.g. *Kentucky Straight Bourbon Whiskey* or *Vodka*), for beer it might be style (e.g. *IPA*), etc.
- **Alcohol Content** – The alcohol by volume (ABV) percentage (e.g. *45%*). You can allow the user to enter this as a percentage or a number.
- **Net Contents** – The volume of the product (e.g. *750 mL*, *12 fl oz*). *(Optional for minimum viable product, but include if you can)*.
- **Other Info (Optional)** – You may include additional fields as you see fit, such as **Manufacturer/ Bottler Name & Address** or **Warnings**, but these can be considered bonus. The form linked in the prompt includes many fields; we only need a subset for this project, so use your judgment to keep it simple.

Ensure the form is clear and easy to use. For example, label each field and use appropriate input types (text fields for names, maybe number or text for ABV, etc.). Include an option to upload an image file of the label (see next section).

2. Image Upload for Label

Provide a way for the user to **upload an image of the alcohol label**. This will simulate the actual label artwork that TTB would review. The image could be a photograph or a generated picture of a bottle label. For the purposes of this assignment, you can assume the image will contain the necessary text (brand name, etc.) clearly enough to be recognized.

- You can make this an `<input type="file">` in an HTML form, or a drag-and-drop area, etc., as long as the user can select an image from their computer.
- The app should support common image formats (JPEG, PNG, etc.).

3. Backend: AI Processing (OCR/Image Analysis)

On the back end, implement the **"AI" functionality to extract text or relevant information from the label image**. In a real system this could involve complex image recognition, but for this project basic OCR (Optical Character Recognition) is sufficient. You have flexibility in how to implement this, for example:

- Use an OCR library or service to pull out text from the image (e.g. Python's `pytesseract`, Google Vision API, AWS Textract, OpenCV, or any tool you prefer). This will allow you to identify the text printed on the label, such as the brand name, alcohol percentage, etc.
- (Optional) If you want to get more creative, you could also do things like image classification or template matching, but **OCR is the most straightforward approach** here.

Processing Steps: The backend should take the uploaded image, run the OCR/text extraction, and then compare the extracted text to the form inputs. For example:

- Does the text on the label contain the **Brand Name** exactly as provided in the form? 2
- Does it contain the stated **Product Class/Type** (or something very close/identical)?
- Does it mention the **Alcohol Content** (within the text, look for a number and "%" that matches the form input)? 3
- If you included Net Contents in the form, check if the volume (e.g. "750 mL" or "12 OZ") appears on the label.

- **Health Warning Statement:** For alcoholic beverages, a government warning is mandatory by law. A real TTB check would verify that the label has the exact text of the warning statement (the standardized text about pregnancy and driving) [4]. For simplicity, you can at least check that the phrase "GOVERNMENT WARNING" appears on the label image text (and possibly that some portion of the warning text is present). This can be a bonus feature if you have time.

You might need to do some basic text normalization (e.g., ignore case differences, or common OCR errors) when comparing. Use your judgment on how strict to be. Ideally, the app should flag obvious mismatches (like completely different brand names or numbers), but it could allow minor differences in formatting (like "Alc 5% by Vol" in image vs "5%" in form). Clearly document any assumptions in how you perform matching (for example, "the match is case-insensitive" or "the alcohol percentage must match exactly as a number").

4. Verification & Results Display

After processing, the app should present a **results page or message** that clearly indicates whether the label **"matches"** the form or not. This should include:

- A **success message** if all the key fields match (e.g. " The label matches the form data. All required information is consistent.").
- If there are mismatches or missing info, a **failure message** (e.g. " The label does not match the form.") along with specifics on what didn't match. For example:
- "Brand name on label ('Old Tom Distillery') does not match the form input ('Tom's Distillery')."
- "Alcohol content on label (8.0%) differs from form (5.0%)."
- "Government warning text is missing from the label."
- If the image couldn't be processed (e.g. OCR found no text or the image was too low-quality), handle this as an error case and inform the user (e.g. "⚠ Could not read text from the label image. Please try a clearer image.").

Make sure to handle both **success and failure states** gracefully. Even if multiple fields are wrong, the app should report all the discrepancies, not just stop at the first. This shows thoroughness in verification, similar to how TTB would review and point out all issues.

Additionally, consider the user experience: - If everything matches, seeing a green check or success note is great.
- If not, the user should know exactly why – listing each field check outcome helps demonstrate what the app did.

5. User Interface & UX

While this is not primarily a design test, the UI should be reasonably pleasant and clear. A simple, clean design is fine: - Use logical layout for the form (group related fields, use proper labels).
- After submission, show results clearly; you might display them on the same page below the form or navigate to a results page. Up to you.
- If possible, allow the user to easily try another image or edit the form if things didn't match (so they don't have to refill everything from scratch on each attempt).

Including visual cues (like ✓ / ✗ or colored text) for match/mismatch can enhance clarity. **Bonus:** If you're comfortable with front-end work, you could even highlight the portions of the image where the text was found, but this is not required.

Sample UI Idea: You could have a single page app where the form is on the left and an image preview on the right; when the user submits, below that a panel appears with a checklist of each item (Brand, ABV, etc.) marked "Matched" or "Not Found/Mismatch". Use your creativity but keep it user-friendly.

6. Technical Constraints and Flexibility

- **Technology Stack:** You are free to use any programming languages or frameworks you prefer. For example, you could build the frontend with HTML/CSS/JavaScript (or a framework like React/Vue) and the backend with Node.js, Python (Flask/FastAPI/Django), etc. A full-stack framework like Next.js or a simple server-rendered approach is also fine. Use what you're comfortable with, as long as the end result is a web-accessible app.
- **AI Libraries/Services:** You likewise can choose any libraries or APIs for OCR and image processing. Keep in mind the time constraint – it's perfectly acceptable (even encouraged) to use off-the-shelf solutions (open-source libraries or cloud APIs) rather than writing your own OCR from scratch. For instance, **pytesseract** (Tesseract OCR) is an easy way to get text from images in Python, and there are JavaScript OCR libraries as well. If using an external API, ensure it's free or has a free tier and that no sensitive keys are exposed in your code.
- **Deployment:** We'd like to see the app deployed live. You can use a service like **Vercel**, **Netlify**, **Heroku**, **Render**, or any hosting of your choice. The app doesn't need to handle heavy traffic; a free tier or personal deployment is fine. If deployment is problematic, at least provide clear instructions to run the app locally.
- **Data Persistence:** There is no requirement to use a database or save data between sessions. You can keep it simple – the form submission triggers the check and results are shown immediately. If you prefer to demonstrate some state handling, that's up to you, but not required.
- **Testing:** Given the short timeline, formal unit tests are optional, but you are welcome to include some if it helps demonstrate your skills. At minimum, please do some manual testing with different images to ensure your solution works for the basic scenarios.

7. Sample Label Examples

To help you visualize the task, here are a couple of **sample alcohol label images** with the kind of information your app should handle:

Example Distilled Spirits Label. This sample label contains key mandatory information: **Brand Name** ("OLD TOM DISTILLERY" at the top), **Class/Type** ("Kentucky Straight Bourbon Whiskey"), **Alcohol Content** ("45% Alc./Vol. (90 Proof)"), **Net Contents** ("750 mL"), the bottler's statement, and the **Government Warning** text at the bottom. Your form inputs for this image might be: Brand Name = *Old Tom Distillery*, Class = *Kentucky Straight Bourbon Whiskey*, Alcohol Content = *45%*, Net Contents = *750 mL*. The app should confirm all these are present in the image text. (For example, it would find "Old Tom Distillery" in the image OCR result, match "45%", etc., and verify the warning statement is included.) [5] [3]

Image: A fictitious bourbon whiskey label with all required text fields, created for this assignment.

(We encourage you to create or source your own test label images as well. You can use AI image generation tools or simple graphic design to produce labels with varying information. This will help you demonstrate your app working on different inputs. For instance, you might generate a beer label image and see if your app can verify the beer name and ABV, etc.)

Bonus Objectives (Optional Enhancements)

If you complete the core functionality, you can earn **bonus points** by extending the project with any of these extras (these are **completely optional** – do not attempt them until the required parts are done):

- **Detailed Compliance Checks:** Expand the verification to cover more of TTB's rules. For example, ensure the **government health warning statement** not only exists but is exactly as required (including the wording and capitalization of "Surgeon General") [4] . You could store the expected warning text and compare it to the OCR output to flag any discrepancies. Another example: verify that the alcohol content on a wine is within allowed descriptors (e.g., "Table Wine" if no ABV is given). These require deeper domain knowledge, so only attempt what you're comfortable with.
- **Multiple Product Types:** Allow the form to handle different beverage types (e.g. Beer, Wine, Distilled Spirits) with slightly different required fields or checks. For instance, wine labels might need a sulfite declaration, beer labels might list ingredients, etc. This would show adaptability. (You could include a dropdown to select the category, and then adjust which fields or checks are applied.)
- **Image Highlighting:** Make the result more interactive by highlighting on the label image where each piece of information was found. For example, draw a rectangle around the detected brand name or ABV on the image. This can be done by capturing OCR coordinates or using an image editing library to overlay highlights. It's a great visual touch if you can implement it.
- **Refinement of OCR Results:** Incorporate some logic to handle OCR errors or text mismatches more smartly. For instance, if OCR misreads "O" as "0" or misses a small character, your comparison could be made fuzzy or tolerant to minor errors. This could involve techniques like edit distance or regex matching (e.g., ignoring spaces or punctuation when comparing).
- **Polish and UX Improvements:** Go beyond the basic UI – for example, make it a single-page application with async form submission (AJAX) so the page doesn't fully reload. Add loading indicators while the image is being processed. Make the design look professional (could be a simple Bootstrap styling or any CSS framework). Though not the primary goal, a well-polished app will certainly be noticed.
- **Automated Tests:** If you enjoy testing, you could add a few automated tests for your text-matching logic or a couple of end-to-end tests using a tool like Selenium or Playwright. This is not expected for a one-day project, but it's an impressive bonus if done.

Feel free to mention any bonus features you implement (or attempted) in your documentation so we don't overlook them.

Deliverables

By the end of the project, you should deliver the following:

1. **Source Code Repository:** Upload your code to a GitHub repository (or similar platform) and send us the link. The repository should contain:
2. All source code for the project (frontend and backend).

3. A README file with clear instructions on how to run the application locally. Include any setup steps, installation commands, or requirements. If special environment variables or API keys are needed, explain how to set those up (and consider including sample dummy values or using a config file).
4. Brief documentation of your approach: for example, mention which OCR or AI tools you used, any key assumptions or decisions you made, and any known limitations. This can be in the README or a separate docs file. We're interested in why you chose certain solutions, so we can understand your thought process.

5. If you wrote tests or additional scripts, include instructions on how to run those as well.

6. **Deployed Application Link:** Provide a URL where we can access the running app (e.g. a Vercel or Heroku link). We should be able to load the URL in a browser, upload an image and fill the form, and see the results. Make sure this deployment is the same version as your code in the repo. If there are any login or credentials needed (not likely for this project), include those. *Note:* It's fine if the app is only up during the evaluation period; you can take it down later if it's on a personal account, but please ensure it's live when you submit.

7. **Sample Test Evidence (optional):** You can also send a few screenshots or brief video/gif of your app in action, especially if there are interesting cases to show. This is not required, but sometimes helpful to quickly demonstrate that it works as intended. For example, a screenshot of the form filled out, the label image, and the results page highlighting a mismatch.

Finally, when you have everything ready, **send us an email** (or whatever channel we've been using) with: - The link to your repository
- The link to your live deployed app
- Any additional notes or comments you want to add about your solution.

We will review your code and try out the application ourselves.

Evaluation Criteria

We will evaluate the take-home project on several factors: - **Correctness & Completeness:** Does the app fulfill the core requirements? Does it accurately detect matches vs mismatches between the form and image? Did you include all the required fields and checks?
- **Code Quality:** Is the code well-organized, readable, and maintainable? (E.g., clear module structure, understandable variable names, not overly complex, etc.) We will read your code, so use best practices as time allows.
- **Technical Choices:** Did you choose appropriate tools/libraries for the task? (For instance, using an OCR library makes sense here; doing something like using a heavy neural network where a simple OCR suffices might be overkill – we are looking at your judgment.) Using modern frameworks or sticking to vanilla code is your decision; just be able to justify why it made sense for you.
- **UI/UX and Polish:** We don't expect a design award-winning app in a day, but a user-friendly interface and nice touches (like clear messages and error handling) will be noted.
- **Followed Instructions:** This write-up contains a lot of details – we will check that you included what was asked (for example, if the instructions say to include a README and deployment, we expect those to be provided). This is an important part: it shows you can pay attention to requirements.
- **Creativity & Bonus Efforts:** Any extra mile features or creative approaches you took will certainly be

considered as a plus. We want to see that you can think for yourself and add sensible improvements without explicit instructions. Just make sure the core requirements are solid before spending time on extras.

We understand this is a time-constrained assignment, so we don't expect perfection. **Use your best judgment** on where to focus your efforts. It's better to have a working core application with clear code, than an overly ambitious project that is incomplete or buggy. If you had to make trade-offs or cut certain optional parts due to time, you can mention those in your notes.

Good luck, and we look forward to seeing your finished app! If you have any questions about the requirements, feel free to reach out for clarification (though we also value how you fill in the gaps on your own). Now, get coding and have fun with it!

---

1 2 3 4 5 ttb.gov
https://www.ttb.gov/media/66695/download?inline