Team 8 - CodeCap
Final Project Design
2/10/2020

Team Members:

Seth Peterson, Moe Zeid, Blake Jordan, Austin Wildman, Subhankar Dey

Synopsis:

A Chrome extension that allows users to screenshot a section of the screen and copies the recognized text to the user's clipboard.

Description:

The main use case for this Chrome extension will be coding tutorial videos. Take this scenario for example: a student is watching a YouTube video about a coding topic. The student wants to copy the code directly to their clipboard, but cannot. Therefore they spend time switching between an IDE and the video manually typing the code. The CodeCap extension will solve this problem. Outside of YouTube, this service can be used in any number of contexts where code is displayed in a non-text format.

The end result of the project will be a Chrome extension that, when activated, will prompt the user to select a section of their screen. The extension will then convert the image to text using Base 64 encoding. Then the text will be sent to an AWS Lambda function that executes Tesseract-OCR. The result of the Lambda function call, assuming no errors, will be copied to the user's clipboard.

Milestones:

Fall Semester:

1. Determine a language best fit for the requirements of the project and set up a basic development environment **(September 25th)**
2. Put together a high level diagram of how the application receives a request and sends a response **(October 2-5)**
3. Put together plans for the individual components of the application i.e the front end, the image capture, the image compression & send, character recognition, and more **(October 2-5)**

4. Decide on a database service and if applicable, determine a budget for using that service **(Mid October)**
5. Solidify plans and begin working on the individual components **(Early to mid November)**
6. Implement the Javascript frontend **(Mid to late November)**
7. Implement the image capturing and sending capability **(Mid to late November)**
8. Implement the image to text recognition capability **(Late November to Early December)**

Spring Semester:

1. Deploy character-recognizing python script as an API on AWS **(early January)**
2. Hook up javascript front end to python backend **(late January)**
3. User-configurable screenshot capability finished **(late February)**
4. Test all special or unusual OCR cases **(early March)**
5. Minimum viable product reached **(mid March)**
6. Publish extension to chrome store **(late March)**
7. Finish additional features & fix bugs **(mid April)**

Budget:

The only cost for this project will be utilizing two AWS services: Lambda and S3. Lambda will be utilized to execute Tesseract-OCR in a cloud computing fashion. S3 will be used to host the source code for the Lambda function. Since AWS offers free utilization up to a certain threshold, we will not need to purchase anything.

Work Plan:

Extension setup & accessibility: Austin Wildman, Subhankar Dey

Image capturing: Moe Zeid

Image uploading: Austin Wildman

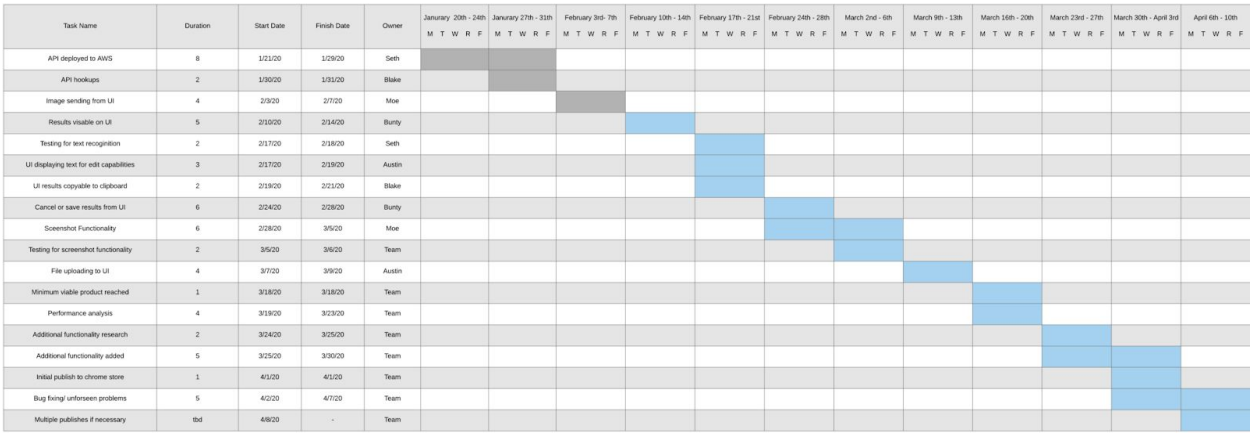
Lambda function setup: Blake Jordan, Whoever needed

Cloud computing: Seth Peterson

Text recognition: Seth Peterson, Moe Zeid

Actions (save, copy/paste, etc): Subhankar Dey

Gantt Charts:



Task Name	Duration	Start Date	Finish Date	Owner	Janurary 20th - 24th	Janurary 27th - 31th	February 3rd- 7th	February 10th - 14th	February 17th - 21st	February 24th - 28th	March 2nd - 6th	March 9th - 13th	March 16th - 20th	March 23rd - 27th	March 30th - April 3rd	April 6th - 10th
					M T W R F	M T W R F	M T W R F	M T W R F	M T W R F	M T W R F	M T W R F	M T W R F	M T W R F	M T W R F	M T W R F	M T W R F
API deployed to AWS	8	1/21/20	1/29/20	Seth												
API hookups	2	1/30/20	1/31/20	Blake												
Image sending from UI	4	2/3/20	2/7/20	Moe												
Results visable on UI	5	2/10/20	2/14/20	Bunty												
Testing for text recoginition	2	2/17/20	2/18/20	Seth												
UI displaying text for edit capabilities	3	2/17/20	2/19/20	Austin												
UI results copyable to clipboard	2	2/19/20	2/21/20	Blake												
Cancel or save results from UI	6	2/24/20	2/28/20	Bunty												
Sceenshot Functionality	6	2/28/20	3/5/20	Moe												
Testing for screenshot functionality	2	3/5/20	3/6/20	Team												
File uploading to UI	4	3/7/20	3/9/20	Austin												
Minimum viable product reached	1	3/18/20	3/18/20	Team												
Performance analysis	4	3/19/20	3/23/20	Team												
Additional functionality research	2	3/24/20	3/25/20	Team												
Additional functionality added	5	3/25/20	3/30/20	Team												
Initial publish to chrome store	1	4/1/20	4/1/20	Team												
Bug fixing/ unforseen problems	5	4/2/20	4/7/20	Team												
Multiple publishes if necessary	tbd	4/8/20	-	Team												

Preliminary Project Design:

The software will be split into two main components: the Chrome extension and the text-recognition Lambda function.

The Chrome extension will handle user interaction, which gives the user the ability to select a section of the screen, which will be analyzed to recognize the “encoded” text. This Chrome extension will be written in JavaScript. The extension will execute the Lambda function and display/use the results that are returned. Before the Lambda function can be executed, the screenshot must be encoded to string format using Base64 encoding. The encoded string will be included in the JSON request to the Lambda function. If an error is returned (in the case of no text recognized), the extension will notify the user. If the Lambda call was successful, the recognized text will be pasted to the user’s clipboard and the user will be notified.

The Lambda function will be written in Python 3.7.5 and use no dependencies besides the Tesseract-OCR binary. The function will take the image as an input as a Base 64 encoded string. The Base 64 string will be decoded back to PNG format. The Tesseract binary will then be executed in a command-line fashion and will output the two following =files: a plain-text file and a tab-separated-value file. The plain-text file contains the recognized text and the tab-separated-value file contains metrics for each “block” of recognized such as confidence values, horizontal indents, and text values. These files will be used in conjunction to reconstruct the horizontal whitespacing of the screenshot. This feature will be very important for horizontal whitespace sensitive languages such as Python.

Design Constraints:

Things constraining this project include the accuracy of Tesseract-OCR, the quality of the input image, and the ability of the encoded string to preserve the image without corruption. In order to improve the character recognition, it is important to provide as much data as possible for pattern matching. Additionally, there are constraints in the design of the application because it must follow some design model in order to work as a Google Chrome extension.

Technical Constraints

One of the technical constraints in this project is the programming language chosen for the specific components of the Google Chrome extension. The language that the text-recognition API will be constrained to Python 3. Another technical constraint is the web-browser compatibility. The extension will be tailored for Google Chrome, therefore it will not be compatible with Firefox, Microsoft Edge, Safari, etc. However, this constraint is minimal compared to what it could have been if the project was chosen to be developed as an operating system-level application. Since it is being developed for use in Google Chrome, the decision was made to write the extension front-end in Javascript. Being a web extension creates a flexibility to use the functionality anywhere where Chrome is supported. As it was alluded above, the project is constrained by the ability of Tesseract-OCR. Ultimately, the extension will only be able to work as well and as accurately as Tesseract-OCR does. One way to minimize such a large dependency on Tesseract-OCR is to set a standard of photo quality in order to have a baseline to work from. If lower quality images are allowed to come through the Lambda, there must be a level of certainty of the accuracy of the optical character recognition that can be fed back to the user in some way. There must also be a standard set in how much deprecation the image can take after being encoded in Base 64 and decoded back to an actual image for Pytesseract to interpret. This does not have to be a percentage shared to the user directly, but if the level of certain is below a certain threshold, the user should receive something like a notification that the recognized text may not be entirely accurate. Another technical constraint exists in the dependency to host the Lambda function using Amazon AWS. Lambda functions execute in an environment on Amazon's in-house operating system, Amazon Linux 1. After beginning work on the Lambda function, several dependency issues were discovered relating to PyTesseract and Pillow (imaging library). These issues were taken into consideration, and the

dependencies were removed. This decision will make executing the Tesseract-OCR binary more complex, but easily do-able without the 3rd party libraries.

Business Constraints

Business constraints across this project are typical for most projects. One of the main constraints is the schedule. The absolute main one is the working prototype due at the end of the Spring semester. Others are ones set by the team. The team plans to complete a strong portion of the front-end and backend by the end of the fall semester. So that time window may work to limit or constrain some planned aspects of the project, but trying to have those components done by then allows time to troubleshoot and polish the project for the Spring. Another business constraint is the fact that the developers working on this project are all enrolled full-time at a university, therefore not every aspect of the project can be built and executed perfectly. The purpose of the Google Chrome Extension is specifically to capture text from images that is meant to be code. That in itself represents a business constraint. Because of the very specific and predictable ways that code is formatted in, the design of the text capture and the way it goes about interpreting it is constrained to what format code may generally take.

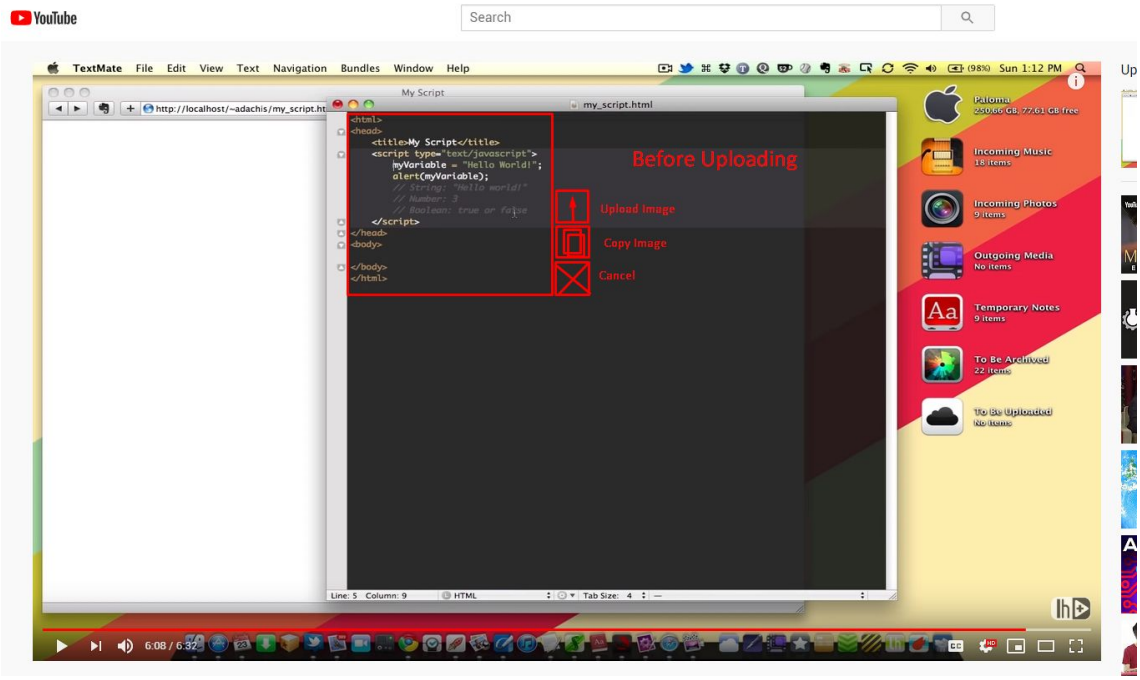


Illustration of users' view before uploading an image. Draggable screenshot window will pop up with the press of a button (e.g. print screen). User can cancel, upload the image, or copy the image to their clipboard

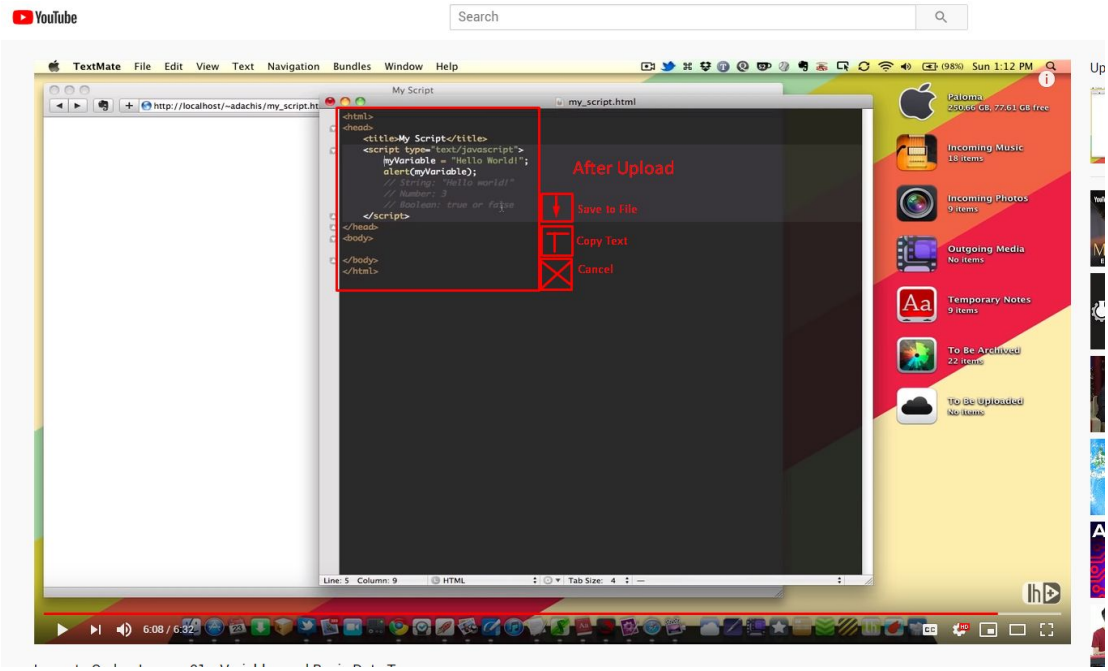
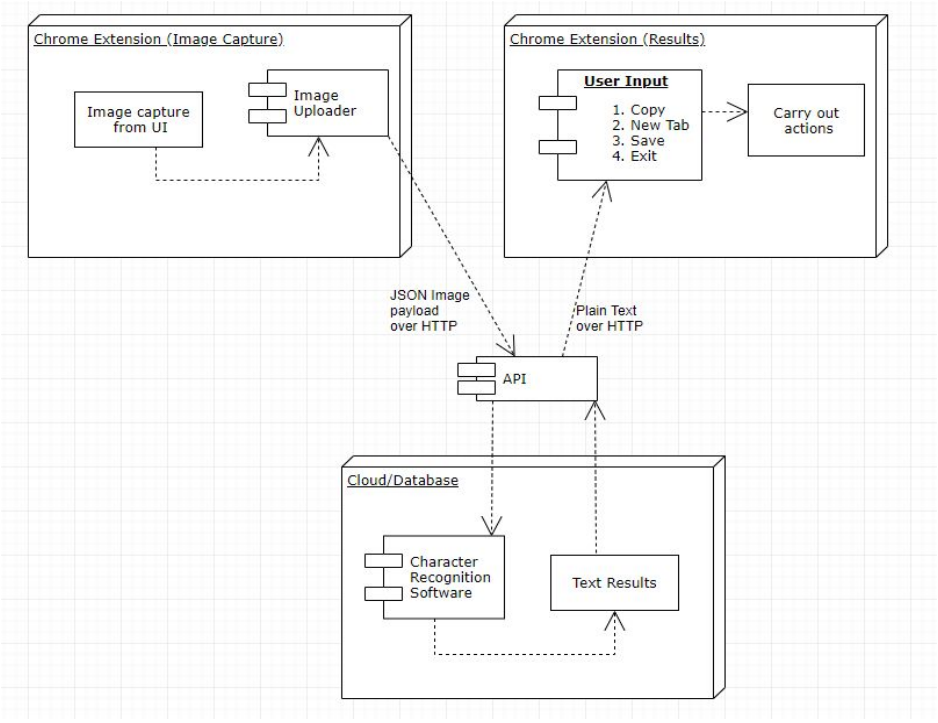


Illustration of users' view after uploading image and receiving the resulting text. User can cancel, save the resulting text to a file, or copy it to their clipboard



System view of the application. Includes the chrome extension component (before and after image upload), and the api used to perform character recognition

Ethical Issues

- Academics

  Since CodeCap's fundamental purpose is to be able to copy code from tutorial videos or pictures, any student could potentially abuse it for academically dishonest reasons. While the ability to use unoriginal code in academic settings is already available (via copy/paste), CodeCap could make this process a lot easier in various settings. For example, if a student comes across a video or picture containing code that isn't their own, they could use CodeCap to quickly copy and paste it into their project, rather than typing it out manually. Skipping the process of manually typing the code could discourage students from thinking about how the code works, as well as how they could write a similar version on their own.

Intellectual Property Issues

- Technologies

  The implementation of the project will utilize several technologies: JavaScript, Python 3, Amazon Web Services, and Tesseract-OCR. JavaScript has no terms of use, but Python 3 has a PSF Licence Agreement for use. The terms of this agreement can be found at https://docs.python.org/3/license.html. Amazon Web Services also has a list of Service Terms. This list of terms can be found at https://aws.amazon.com/service-terms. Tesseract-OCR iis listed under a Apache License V2.0. The only limitations of this license is the lack of liability and warranty, which will not be an issue for our use case.

Change Log:

- Budget:

  The budget was updated to reflect the need for deploying the Tesseract-OCR functionality with AWS Lambda and S3.

- Preliminary Project Design:

The preliminary project design was updated to reflect the removal of PyTesseract, specify use of Tesseract-OCR, and specify the use of AWS Lambda and S3.

- Design Constraints:

  The design constraints section was updated to reflect the removal of PyTesseract, specify use of Tesseract-OCR, and specify the use of AWS Lambda and S3.

- Work Plan:

  The work plan was edited to include more topics, allowing for a more elaborate division of labor.

- Project Name:

  The project name was changed from Snap Shot to CodeCap in order to convey the main purpose of the project: capturing code.

- Gantt Charts:

  Updated the Gantt chart to include accurate todo items for the spring semester.

- Project Milestones:

  Made spring milestones more specific and consistent with the current work plan