In this project, you will learn to track faces in a video and identify facial expressions using Affectiva. As a fun visualization, you will tag each face with an appropriate emoji next to it. You will then turn this into a game where the player needs to mimic a random emoji displayed by the computer!

Ready?

Getting Started

Clone the **AIND-CV-Mimic** repo on GitHub, and follow the instructions below.

We'll be using **Affectiva**'s Emotion-as-a-Service API for this project. Visit their **Developer Portal** and try out some of the sample apps. Affectiva makes it really easy to extract detailed information about faces in an image or video stream. To get a sense for what information you can obtain, check out the **Metrics**page.

Project files

To start working on the project, open the following files in your favorite text editor:

- **mimic.js**: Javascript file with code that connects to the Affectiva API and processes results.
- **index.html**: Dynamic webpage that displays the video feed and results.
- **mimic.css**: Stylesheet file that defines the layout and presentation for HTML elements.

You only need to implement the TODOs in mimic.js to complete the project. But feel free to modify the HTML and/or CSS file to change the look and feel of your game!

There are two additional files provided for serving your project as a local web application - you do not need to make any changes to them:

- **serve.py**: A lightweight Python webserver required to serve the webpage over HTTPS, so that we can access the webcam feed.
- **generate-pemfile.sh**: A shell script you'll need to run once to generate an SSL certificate for the webserver.

Serving locally over HTTPS

In order to access the webcam stream, modern browsers require you to serve your web app over HTTPS. To run locally, you will need to general an SSL certificate (this is a one-time step):

- Open a terminal or command-prompt, and ensure you are inside the `AIND-CV-Mimic/` directory.
- Run the following shell script: `generate-pemfile.sh`

This creates an SSL certificate file named `my-ssl-cert.pem` that is used to serve over https.

Now you can launch the server using:

```
python serve.py
```

Note: The `serve.py` script uses Python 3.

Alternately, you can put your HTML, JS and CSS files on an online platform (such as **JSFiddle**) and develop your project there.

Running and implementing the game

Open a web browser and go to: **https://localhost:4443/**

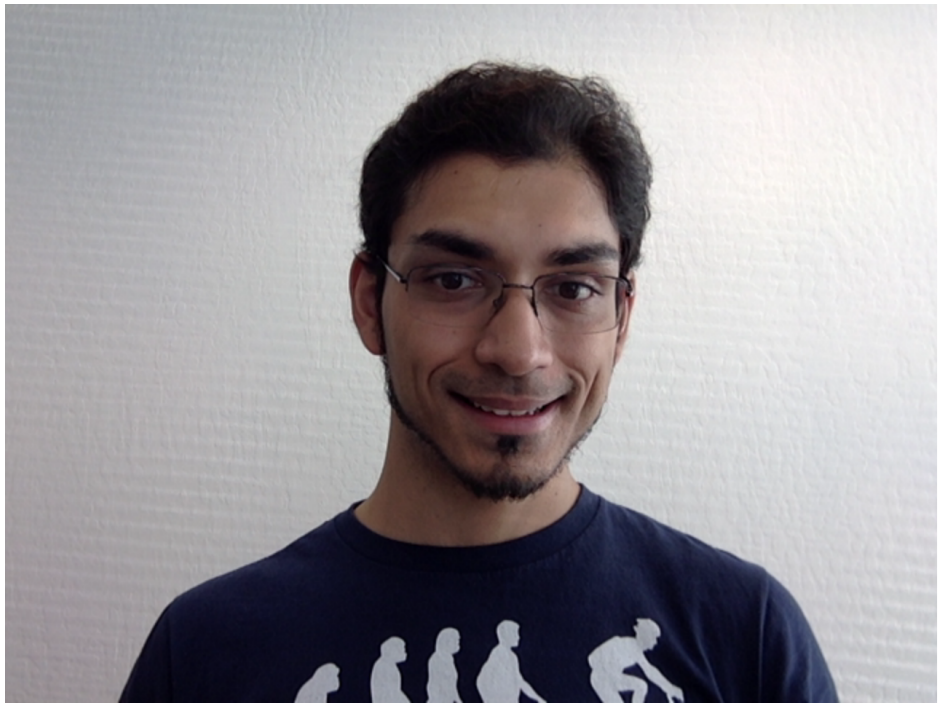
- Hit the Start button to initiate face tracking. You may have to give permission for the app to access your webcam.
- Hit the Stop button to stop tracking and Reset to reset the detector (in case it becomes stuck or unstable).
- Modify the Javascript code to implement TODOs as indicated in inline comments. Then refresh the page in your browser (*you may need to do a "hard-refresh" for the changes to show up, e.g. `Cmd+Shift+R` on a Mac), or use an auto-reload solution.*
- When you're done, you can shutdown the server by pressing `Ctrl+C` at the terminal.

Note: Your browser may notify you that your connection is not secure - that is because the SSL certificate you just created is not signed by an SSL Certificate Authority. This is okay, because we are using it only as a workaround to access the webcam. You can suppress the warning or choose "Proceed Anyway" to open the page.

As you work on your code, you may have to refer to resources in Affectiva's **JS SDK documentation**.

Tasks

The starter code sends frames from your webcam to Affectiva's cloud-based API and fetches the results. You can see several metrics being reported, including emotions, expressions and the dominant emoji!



EMOTION TRACKING RESULTS
Timestamp: 38.18
Number of faces found: 1
Appearance: {"gender":"Male","glasses":"Yes","age":"25 - 34","ethnicity":"Unknown"}
Emotions:
{"joy":100,"sadness":0,"disgust":0,"contempt":0,"anger":0,"fear":0,"surprise":2,"valence":100,"engagement":100}
Expressions:
{"smile":100,"innerBrowRaise":0,"browRaise":0,"browFurrow":0,"noseWrinkle":0,"upperLipRaise":0,"lipCornerDepressor":0,"chinRaise":0,"lipPucker":0,"lipPress":0,"lipSuck":0,"mouthOpen":100,"smirk":0,"eyeClosure":0,"attention":96,"lidTighten":0,"jawDrop":0,"dimpler":0,"eyeWiden":5,"cheekRaise":0,"lipStretch":62}
Emoji: 😃

DETECTOR LOG MSGS
Clicked the start button
Webcam access allowed
The detector reports initialized

1. Display Feature Points

Your first task is to display the feature points on top of the webcam image that are returned along with the metrics. To do this, open up mimic.js, and scroll down to the `drawFeaturePoints()` function near the bottom:

```
function drawFeaturePoints(canvas, img, face) {
    ...
}
```

It accepts three parameters: `canvas` (HTML DOM element to draw on), `img` (image frame that was processed), and `face` (an object with all the detected feature points and metrics for a face). The most important object to consider here is `face` - you can even print it to the Javascript console using `console.log()` to see what it contains (tip: the console in your web browser maybe under Developer Tools; in Chrome you can open it using `Cmd+Option+J`).

Implement the `drawFeaturePoints()` function as per instructions and save the file. To run your code, stop the face tracking app using the Stop button (if you haven't done so already), refresh the page and hit Start again.

Your output should look something like this:

EMOTION TRACKING RESULTS
Timestamp: 37.61
Number of faces found: 1
Appearance: {"gender":"Male","glasses":"Yes","age":"18 - 24","ethnicity":"Unknown"}
Emotions:
{"joy":100,"sadness":0,"disgust":0,"contempt":0,"anger":0,"fear":0,"surprise":3,"valence":99,"engagement":100}
Expressions:
{"smile":100,"innerBrowRaise":0,"browRaise":1,"browFurrow":0,"noseWrinkle":0,"upperLipRaise":0,"lipCornerDepressor":0,"chinRaise":0,"lipPucker":0,"lipPress":0,"lipSuck":0,"mouthOpen":94,"smirk":0,"eyeClosure":0,"attention":99,"lidTighten":0,"jawDrop":0,"dimpler":0,"eyeWiden":35,"cheekRaise":0,"lipStretch":9}
Emoji: 😃

DETECTOR LOG MSGS
Clicked the start button
Webcam access allowed
The detector reports initialized

2. Show Dominant Emoji

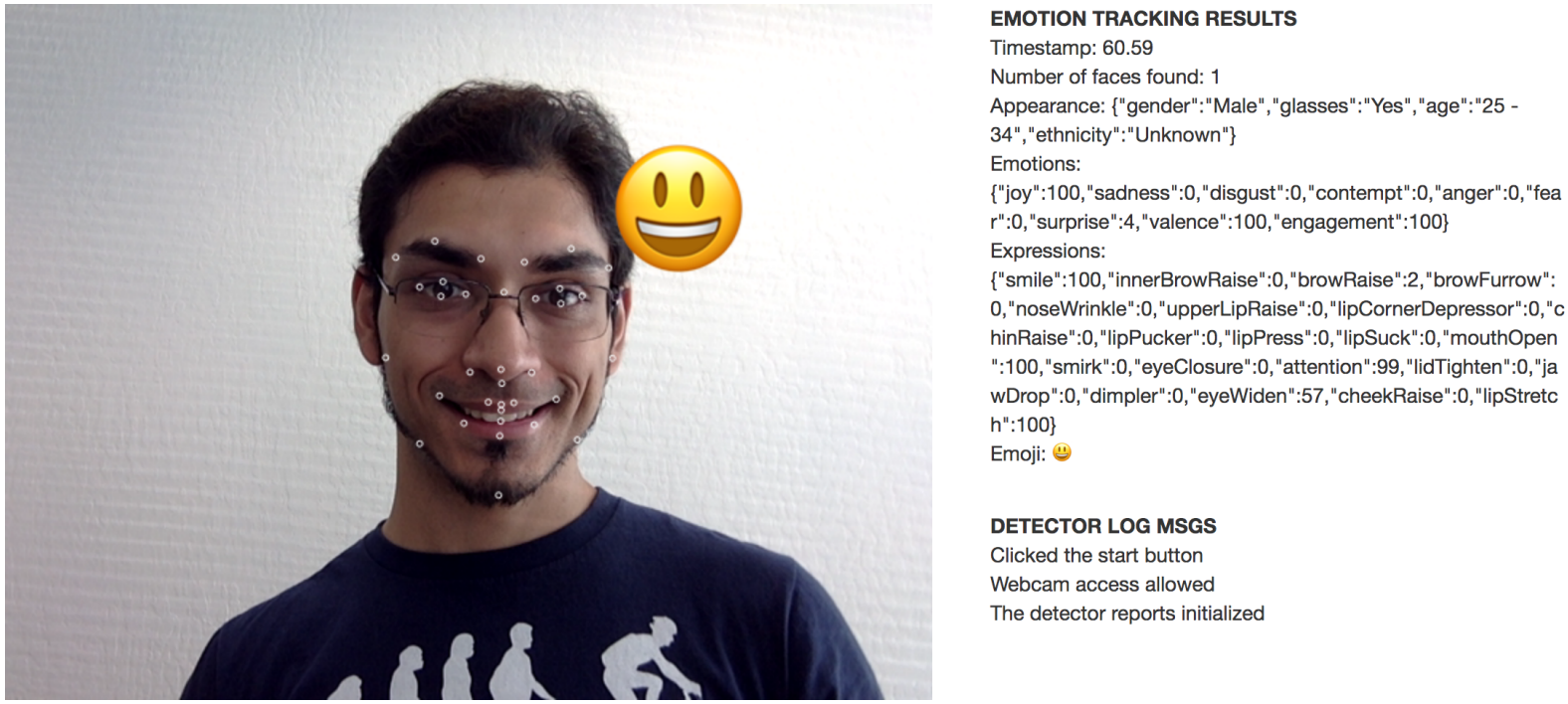
In addition to feature points and metrics that capture facial expressions and emotions, the Affectiva API also reports back what emoji best represents the current emotional state of a face. This is referred to as the *dominant*

emoji. In mimic.js, scroll down to the `drawEmoji()` function:

```
function drawEmoji(canvas, img, face) {
    ...
}
```

It has the same interface as `drawFeaturePoints()`, accepting the same `canvas`, `img` and `face` objects. Implement it as per given instructions. You can access the dominant emoji as: `face.emojis.dominantEmoji`

Your output should now look like:



Note that the emoji should be located close to the face, so you may want to use one of the facial feature points from the step above as the anchor point (or a combination of them). You can also try varying the size of the emoji

based on how big the detected face is (you can compute this, for instance, by looking at the distance between two opposing feature points).

3. Implement Mimic Me!

Now it's your turn to implement the game mechanics and make it as fun as possible! Scroll down to the bottom of mimic.js for more instructions. Feel free to modify the HTML and/or CSS files to change the look and feel of the game as well.

In this game, the computer should display an emoji at random, and the goal of the human player would be to mimic that emoji as best as they can. The computer should continually monitor the player's face, and as soon as they are able to mimic the face (or optionally after some timeout), the game should move on to the next random emoji.

Affectiva's SDK can recognize 13 different emojis. The unicode values for these emojis are provided as a list in mimic.js:

```
// Unicode values for all emojis Affectiva can detect
var emojis = [ 128528, 9786, ..., 128561 ];
```

Each value corresponds to an HTML entity, e.g. `&#128527;` for a smirk: 😏

To display a desired emoji as the next target for the player, you can use `setTargetEmoji()`, passing in the respective value. The dominant emoji returned by Affectiva is supplied in a Javascript string. In order to reliably compare it with the desired emoji code, you will need to convert it to unicode using `toUnicode()`. Both of these are provided as utility functions.

Some things to keep in mind:

- You can keep track of how many emojis the player is able to mimic in a certain amount of time (say, 1 minute) and give them a score based on that, or show a set number of emojis in a sequence and see how many they are able to mimic within a timeout for each.
- Remember to give the API some time to warm up and find the face before you start showing the emojis.
- Provide some audio/visual feedback whenever the player is able to mimic an emoji successfully.
- You may want to limit the possible emojis to a small set that you know that the system can recognize reliably (or ones that you can actually express!).

Extensions

Sky's the limit on where you can take this project! Feel free to share with your friends and family. You can host it online to make it available to everyone.

Some ideas for extensions:

- Make it a 2 player game, like Guitar Hero, where you compete with someone to mimic as many emojis as you can out of a streaming sequence of them.
- Pair a stream of emojis with a script and have the player read the script, interspersed with emotional expressions that are checked by the computer. Great for some acting practice!