A5: Beyond Web & Mobile

Due Tuesday, March 16th, 11:59pm PST

In this assignment, you'll explore how the interface development principles you have explored in web and mobile interfaces apply to interfaces for other devices by implementing a time zone converter. Many websites already provide this functionality, but this task is intended to give an overview into how implementation works on a few frameworks for these devices. This assignment is focused on the implementation and interaction with these frameworks, and we have provided functionality to help with the mathematical task of converting time zones.

This assignment may be completed in pairs (exactly two students enrolled in the class) or individually. You may choose your own partner or use Slack to help find one. Because the assignment features build on top of one another, it is recommended that you pair program rather than distribute the assignment features. There is no grade benefit or penalty to completing the assignment individually.

By the end of the day on Wednesday, March 10th, please navigate to the "People" tab on Canvas to sign up for an A5 group. Do correspond with your partner while signing up for groups. If you're the first member to join a group, please sign up for the next group that's empty. Keep a note of the Assignment Group Number that you've signed up for and add it to your readme file.

You will not be able to swap to another group after the 10th. If something happens after you sign up and you decide you're best off working alone (team conflicts, incompatible schedules, etc.), you can disband your group and let the course staff know. However, you will not be able to team up with someone else, even if they do not have a partner. You will need to complete the assignment alone.

Starter code

A starter repository is on GitHub Classroom.

Repository structure

Unlike previous assignments, you might not end up developing your code in your starter repository, but a few files are included to help you get started and to scaffold submission of your assignment. The included files are:

- A `questions.docx` file with some short essay questions for you to explain and reflect on how the framework you used supports interface implementation.
- A `readme.txt` file to explain your implementation and anything else we should know.
- Folders with starter code for each of Fitbit, Alexa, and A-Frame. While most files are unique setup for the framework, each contains a `timezone.js` file with helper functions for using the spacetime and spacetime-informal libraries.

You will also need to create a screen or video recording of your app, `demo.mp4` (or some other reasonable file format such as `.mov` or `.avi`).

Working in pairs

Each framework implements sharing codebases slightly differently. If you are developing command line, one of you can set up your GitHub repository to be shared and add the other as a collaborator. However, some frameworks will rely on their own web editors, but should enable adding other accounts as collaborators.

Given the varying ways in which editors support collaboration and the overall goal of this assignment, we strongly suggest that you complete this assignment via pair programming, alternating roles as drivers and navigators.

Framework options

In this assignment, you may choose one of three development frameworks to implement your time zone converter: Fitbit (watch), Alexa (voice assistant), or A-Frame (AR/VR). You do not need to implement your converter on the actual devices being designed. You can complete this assignment using either the simulator the framework provides or your browser (depending on the framework). However, you may run your solution on the device for extra credit. You may also implement more than one development framework for extra credit.

This assignment intends to demonstrate how the skills you have built up for implementing interfaces can translate to new platforms and devices. Moreso than previous assignments, you will need to leverage the documentation of the frameworks in order to complete this assignment. We are not covering the technical details of any of these frameworks in lecture, as this assignment is your opportunity to apply what you've learned about interface implementation to a new framework. This might seem daunting, but it's possible to complete the bulk of this assignment with only a few lines of code.

The frameworks have roughly equal difficulty. Alexa is a bit harder to get started with, Fitbit has some quirks around using the spacetime libraries, and A-Frame is a bit harder to support interactivity with. But your experience will inevitably vary. Below we have added a few suggestions for setting up and getting started with each framework.

Fitbit setup and advice

Fitbit requires creation of a Fitbit account (no device is required to create one), registering it with Fitbit Studio, and downloading the simulator. You can develop your app entirely within Fitbit Studio, but doing so makes it harder for us to share starter code, so we recommend using the command line interface. To do so, we will use npx, a tool for running binary files from npm packages. Once you have `cd`ed into your directory and opened the

simulator, run `npx fitbit-build` and `npx fitbit` to load the Fitbit shell. Once the shell is loaded, run `build-and-install` (or simply `bi`) each time you make changes to your code. You may need to change what device it is running on, such as to the Versa 3 or the Ionic (which is not the same as the Ionic framework used in A4), to resolve compiler errors.

A Fitbit app includes many files by default, most of which you will not need to modify. There are two main files to pay attention to: `resources/index.view`, which defines the main view of your app, and `app/index.js`, which defines the code behind your app. You may additionally want to edit `resources/styles.css`, which contains all of your CSS styles. You might reference the `companion` folder, which contains the spacetime libraries and `timezone.js` helper, but you shouldn't need to edit these files for this assignment.

The code behind Fitbit apps are separated between an `app` folder, which contains the main code behind the app, and a `companion` folder, which contains code to run on a paired device (like a phone). Because the watch has limited resources, computationally-expensive operations need to take place on the paired device. In our case, the watch has too little memory even to load the spacetime library! As a result, we need to send messages between the app and the companion to get and convert times. We have provided helper functions for sending a request to the companion to use the spacetime library and to listen for a response and parse it. Fitbit providers helper guides for using SVGs and editing the interface with JavaScript, but many of these interactions should be familiar from A2.

Alexa setup and advice

There are multiple ways of developing for Alexa, and many possible dialogs which can satisfy the assignment tasks. Getting started with Alexa requires a mix of designing the dialog and programming how the assistant should respond. We have created some slides to give some direction and a few examples of skill interactions for asking for the current local time or time in another city. Amazon also provides documentation and tutorials to help get started. We recommend using Alexa Conversations in this course, but you are welcome to use the base Alexa Skills Kit if you wish. You may find that your Alexa Conversations take a little while to compile (1-2 minutes. This is normal, as your skill is training a machine learning model to understand the interactions you define.

To get started, you must first create an Amazon developer account and a skill off of a provided template (the slides offer suggestions for what to select). The starter code we have provided includes three files: `index.js`, `timezone.js`, and `package.json`. `timezone.js` should be added to your skill; it provides utility functions for using spacetime. The `index.js` we provide should replace the `index.js` provided by the template skill; it adds some functions for looking up the current time and the current city. Rather than replacing the template skill's `package.json` with the file we provided, we recommend copying the dependencies on spacetime and spacetime-informal with their corresponding version numbers.

You can develop your app entirely from the developer console. You do not need to utilize the command line, but are welcome to try Alexa's command line interface if you wish.

Alexa has the ability to complete the tasks required for this assignment as a built-in skill (e.g., Alexa will respond to the question "What time is it in Johannesburg?" and similar). You must develop a custom skill which replicates this behavior; attempting to pass off the built-in skill as your own is not allowed per the academic integrity policies.

A-Frame setup and advice

There are multiple ways of setting up an A-Frame project. An easy way is to run it in your browser using `live-server`. The starter code we provide builds on the A-Frame boilerplate, adding helper functions in `timezone.js` and an example entity `clock.js`. The spacetime and spacetime-informal libraries are imported from Content Delivery Networks (CDNs), in the provided `index.html`. You should not need to edit `timezone.js`; editing `clock.js` and `index.html` (and adding new entities if you wish) should be sufficient for completing this assignment. You may wish to add a CSS stylesheet and link to it in your `index.html`, but this is likely not required.

Your solution can use only mouse and keyboard interactions; you do not need to support a controller to complete the assignment. Some helpful resources include the A-Frame website and Mozilla documents. For a more complex interface, you may find A-Frame GUI components helpful (but you can complete the assignment without using these). For the bonus feature, the A-Frame suggests using Cordova to run your A-frame program on your device natively. This may also require implementing some controller-specific interactions.

Requirements

Your time zone converter is required to support the following functionality:

- Display (or speak) the current time in the local time zone.
- Display (or speak) the current time in other time zones.

This assignment is foremostly about implementing the functionality and reflecting on how the framework you selected supports interface implementation. You will not be graded on the design of your app.

Displaying & converting time

Because the goal of this assignment is to try out a new framework and reflect on how it supports interface implementation, it is okay for your solution to be a **proof of concept**. You do not need to rigorously test every hour/minute, every time zone, etc. If it's easier, your solution can ignore the existence of the International Date Line. You can display the time in another time zone and not display the date. But, if you find it easier, you can choose to display the date. It is similarly fine if your display does not update in real time, you could make all calculations based on the time you booted up the app. But similarly, if you find it easier, you can update in real time. You do not need to support conversion to all time zones. Feel free to pick ~3 which are personally meaningful to you. But again, you can support all time zones if you'd like. The `timezone.js` file provides functionality intended to help create and convert times.

Similarly, this assignment is not being evaluated on the user experience of the app you create. Your interface can be inefficient or unintuitive. It's okay if a person would be susceptible to make input mistakes with the app, such as buttons that would be hard to press on a small screen or exact voice commands which would be hard to remember.

We expect that your converter will display or speak the current time as a static string or similar. The conversion functionalities must be *interactive*, meaning that a person must be able to select the time zone they wish to convert to and trigger the conversion to take place (e.g., by speaking a command, pressing a button, selecting an item from a menu). Again, you can choose whether this is fixed among a few available options (e.g., in a drop-down menu, from a few available buttons, among a list you've supported recognition of) or a free-form description (e.g., typed or spoken in plain text).

Readme

Please update your `readme.txt` with how long the assignment took, who or what online resources you consulted with, any bonus features you added, and anything else we should know. If the assignment was completed in a pair, the readme should list names/emails/ids of both students. Only one submission needs to be made, however.

Demo

Please record a demo video (`demo.mp4` or a different file extension) which shows the demo features of your app: displaying/speaking the current local time and converting the current local time to another time zone. You may record the device simulator; your demo does not need to be running on the actual device. We would prefer a single video walking through each of the different parts of your app, but you may instead find it useful to make separate videos for different parts. If you do record multiple videos, give each a descriptive file name (like `demo_display.mp4`). This video does not need to be polished. It is intended to help us understand the interface you have designed and developed and how it operates, but it's not intended to be a marketing pitch. We expect most videos can be 1 minute or less.

Questions

Please update the `questions.docx` document with answers to each of the questions about how the framework you have implemented supports interactivity. These questions are intended for you to reflect on how the frameworks support some of the more conceptual design and implementation principles we have discussed in this course. We expect answers to be a paragraph, max two paragraphs per question. A few of the questions posed have correct answers, but the questions are mostly intended to trigger consideration of how the framework you used supports implementation.

Submitting

To submit, zip your repository and upload it to Canvas. Ensure the `questions.docx`, `readme.txt`, and `demo.mp4` files are in the root of your repository. Submit your full codebase in a folder labeled for the framework you implement (e.g., `fitbit`, `alexa`, or `a-frame`). For Alexa, you can download your full solution by clicking the "download" button in the *code* tab of the Alexa Developer Console. You can delete code folders you did not implement prior to submitting. If a `node_modules` was created, please delete it prior to submission. Any late uploads are subject to the course's late policy. As a reminder, we will not be able to grant extensions for this assignment due to the short deadline for submitting grades to the registrar.

Because many people will complete this assignment in pairs, you are also required to complete a short evaluation on Canvas signifying how work was distributed. The partner evaluation is available at https://canvas.eee.uci.edu/courses/32801/quizzes/151377.

Grading

This assignment will be graded on a scale of 10 points, broken down as follows:

- The ability to display or speak the current local time (1 point)
- The ability to convert the current local time to another time zone (3 points)
- A demo video demonstrating these features (1 point)
- Answers to questions about how the framework supports implementation (5 points)

There are a few ways to receive extra credit on this assignment, but we will cap the maximum grade for this assignment at 12/10. You can receive extra credit for implementing more than one framework or getting your solution running natively on a device.

You may implement a second framework for 1 point of extra credit, or all three frameworks for 2 extra credit points. To receive this extra credit, each implementation must support all of the required functionality. If you decide to implement more than one framework, indicate this in your `readme.txt` and submit the corresponding folders and multiple demo files (named something like `demo_fitbit.mp4` to distinguish between frameworks). You can pick one framework to answer the essay questions in `questions.docx`.

You can receive 1 point of extra credit for running your solution natively on the device it is designed for. The framework's documentation provides instructions for running your solution natively. Because this feature is extra credit, the course staff may not be able to provide assistance. We recognize that this extra credit option requires owning a capable device, and do not endorse purchasing a new device solely to achieve the extra credit point. Each framework supports a few different devices. Fitbit's framework supports running on the Versa, Versa Lite, Versa 2, Versa 3, Sense, and Ionic models (again, not to be confused with the Ionic framework). Alexa can run on an Echo (Dot, Show, Studio, Plus, Flex, etc.) as well as the Alexa mobile app. A-Frame can run on dedicated VR devices (HTC Vive, Oculus Rift/Quest/Go, Google Daydream, Samsung GearVR, Vive Focus) as well as a mobile phone through Cordova. Please create a separate demo video for running natively (perhaps named `demo_device.mp4` or similar) and indicate this option in your `readme.txt`.

The course staff reserves the right to redistribute, penalize, or award bonus points to one or both students in situations where work was not distributed equitably among partnerships.

In prior courses, you've been asked to follow good principles for indentation, naming variables, commenting, etc. We expect you to do the same in this course, but aim to avoid being draconian in our enforcement of these principles. Egregiously poor formatting, completely uncommented code, etc. may incur a small penalty (e.g., -1 point), but we expect this to be rarely applied.
