# M2: Design Prototype (Framework)

## Overview

As a team, you will complete and deliver a ***design prototype*** demonstrating the feasibility and core features of your project. The features demonstrated will encompass a ***vertical slice*** – i.e., an iteration of your project that has all core features integrated in at least one dimension, though it need not be polished. The prototype should not involve implementations of all aspects of every feature; instead, the design prototype is intended to provide a basis for architecture and practical testing (including usability and user experience). The goal of the design prototype deliverable is to identify exactly how the project will be structured and to provide a foundation for the second phase of the project. The prototype should represent approximately 40 hours of effort ***per person***.

## Specification

The deliverable should meet the following requirements.

### User Experience

The user feedback system, including UI and sensory response elements, should be in place by this milestone. All elements of user control should be present.

#### Interface Design

The application programming interface (API) should be usable. Options should be intuitive and consistent. Additionally, any persistent state of the framework must be easily accessible via the interface.

#### Navigation

The API should be functional and well documented. Bugs should be minimal, if present, and must not detract significantly from the overall experience. Use of API should not require significant time for acclimation; calls should be predictable and easy to use. Features should be discoverable and well-explained.

#### User Perception

The API must be intuitive to users. Teams should have solicited user testing from non-team members to ensure general usability. Users should report an enjoyable experience. The prototype should help users attain stated goals with minimal frustration.

#### Responsiveness

The API should be responsive and should not waste computing resources. For any operation that requires significant I/O operations, the call should yield the CPU while awaiting response – i.e., there should be no busy wait loops. Additionally, if any call may have a delayed return, there should be a non-blocking option within the API call. Success, failure, and exceptional occurrences must be communicated (by exception, error code, or other mechanism) to the calling procedure. Changes in state should be immediately accessible.

## Build Quality

This build should endeavor to avoid major bugs. All content, including art and code systems, should be present.

### Robustness

By this milestone, crashes should not occur within the context of regular (unexceptional) use. There should also be no major or noticeable glitches within regular use.

### Consistency

Except where explicitly otherwise by design, the API and state should act predictably, i.e., for the same call and state, it should yield the same result. When and if the framework behaves unpredictably, it must be for a clear and compelling reason.

### Aesthetic Rigor

No major cosmetic issues should be present. Any aesthetic issues that are present should not cause the application to be unusable. All assets (e.g., images and sounds) should be present and functional within the context of the project.

### Vertical Features

All major architectural elements should be complete with at least one demonstrative implementation.

#### *Front-End*

For each major use-case, at least one variant of the application call must be implemented within the API of the framework. This must connect to the *persistent state*.

#### *Persistent State*

For each major use case, there must be a demonstration of use of data structures and framework state in the application. This must connect to the *front-end* and the *back-end*.

#### *Back-End*

For each major use case, at least one variant of the data processing step must be implemented. This must connect to the *persistent state*.

# Submissions

Your submission will be a tgz (gzipped tar) archive including the following:

- Source code and assets for submission
- README file

## Source & Assets

Submissions should include all source code, including build configurations and scripts (e.g., setup.py and/or Cargo.toml) and run scripts as necessary. The project should be runnable using an executable command without parameters and should be prepared by build scripts. For server applications, a command to initialize the server state for first run may be included. Submissions should also include all pre-built assets necessary for application function (e.g., pre-built database collections necessary for initial function and/or visual/audio assets).

## README File

A README file, either in plaintext or markdown (UTF-8) should be included. The README should have, at a minimum, the following information:

- A description of the project
- One-line command for building and installation
- Link to the repository for the project (e.g., GitHub) [Note: if private, you must add instructors]
- List of packages included in framework
- Executable command(s)