# Final Planning

- Summary

  Our final product will consist of a well designed desktop application that will allow a user to input a JSON file containing signal data (array of floats), transform the data using the forward or inverse WHT, then return the data compressed or extracted according to the user's input. This application will give the user access to graphing and input widgets that interface with the back-end and allow for some customization of screen elements to fit their needs. The back-end will pre-compute the matrices needed for the WHT and apply the WHT to input data. There will also need to be an API that links the graphing library with the transform functions.

- Team Transformers' Roles:

  Derik: Leader of the pack: Free hands to help wherever needed, documentation, facilitate needs between roles

  Finn: Frontend Code: UI/visualization using Maui Blazor

  Chet: Front/Middle: Graphing API with Transform Library

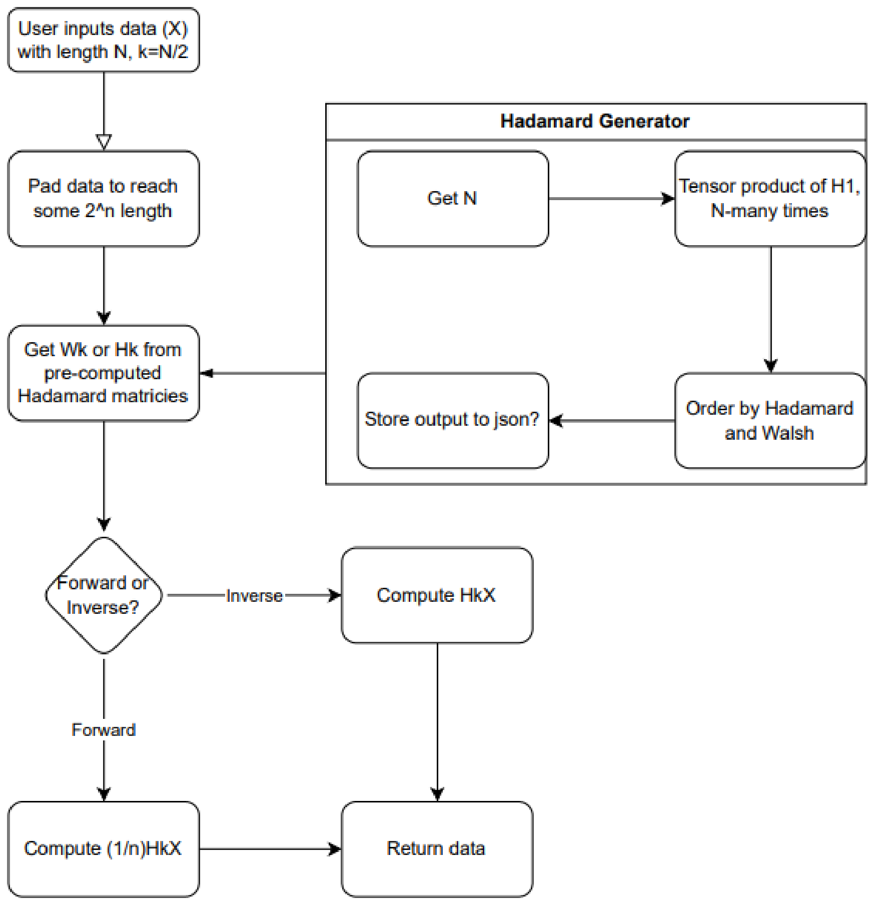
  Kain: Backend Code: Matrix Generation and Transform Library

  Calvin: Backend Code: Matrix Generation and Transform Library

- Description and Functional Requirements

  Like mentioned above the MVP will be a desktop app that allows users to upload a JSON file that contains their data which will be sent to the backend through an API and transformed and then sent back and displayed in a graph. The tools we will be using are C#, MAUI Blazor and a graphing API (still undecided).

- A High-Level Systems Overview [e.g. what are the major components in your system]; Diagrams and Text descriptions would be appropriate


User inputs data (X) with length N, k=N/2
Pad data to reach some 2^n length
Get Wk or Hk from pre-computed Hadamard matricies
Hadamard Generator
Get N
Tensor product of H1, N-many times
Order by Hadamard and Walsh
Store output to json?
Forward or Inverse?
Inverse
Compute HkX
Forward
Compute (1/n)HkX
Return data

- List and Detailed Description of the MAJOR Milestones for your project (think top 3-4 things) -- How will you know when you have successfully reached/completed each milestone?

Walsh and Hadamard matrix generator

1. A C# service that generates and stores the Walsh and Hadamard-ordered matrices needed for the WHT algorithm.
    - Computes nth Hadamard matrix using the recursive algorithm
    - Creates a copy of the Hadamard matrices in Walsh/sequencey ordering
    - Stores the output in JSON format to save computation at runtime

Front End

2. A decently designed UI/UX that provides widgets for some customization of the workspace.

        - The app should successfully take in user input, call the API and display the data according to the chosen widgets.
        - File I/O
        - Responsive and modern design with bootstrap

    Graphing and Transform API

    3. A simple to use and understand structure that allows us to efficiently and effectively transfer data generated by the engine to the front end. C# is the most obvious candidate in the group to do this, as it will be much easier to implement it alongside maui blazor.
        - Takes input data and asynchronously returns the graph data to be displayed.

    Back End

    4. Takes an input of data, transformation type and compression factor at a minimum. Applies appropriate transform and returns a JSON of the transformed data to be used in the Graphing and Transform API.
        - Follows the WHT algorithm and returns transformed data with minimal latency.

- Assuming a 10-week development from January 22 to March 30 :
    - Week 0: Planning done, initial code experimentation over winter break. First Maui scaffolding in git repo.
    - Week 1-2:
        - Team 1: Finn, Chet
            - Get front-end hooked up with a graphing API. Setup connection with back end.
        - Team 2: Calvin, Kain, Derik
            - Start working with matrices. Get the back end hooked up with the graphing API. Start figuring out extra needs from these relationships. Goal is outlined code with cursory functionality within components, strong and useful connections between them.
    - Week 3-4:
        - Team 1: Finn, Chet
            - Get graphing API working and mock WHT Service for now. Setup widget elements, If using Plotly.NET then start creating graph templates (templates are a feature of Plotly). We will be able to use these for stretch goals and we will likely be waiting for Team 2 at this point. Generate data for testing WHT library if other goals complete.

        - Team 2: Calvin, Kain, Derik
            - Have the WHT done and unit tested. Format output for use in graphing API.
    - Week 5-6:
        - Team 1: Everyone
            - Finalize file I/O if not done already. Test WHT library against test data. Finalize MVP and begin stretch goals. If we are behind schedule, delay until done with MVP.
    - Week 7-8:
        - Team 1: Finn, Chet, Derik
            - Implement more widgets to allow for FFT, similar to WHT requirements. Mock FFT from the back-end in the graphing API until FFT is implemented in the back-end.
        - Team 2: Calvin, Kain
            - Find a Fast Fourier Transform library and hook it up to our Library. Should be accessible similarly to WHT.
    - Week 9-10:
        - If all is going well, then our primary goal is to create the Polyphonic tuner since we would just need one more, relatively easy to implement algorithm. If we don’t have enough time then we can skip the implementation but still show the flow from input through FFT and then we could just discuss the last steps in our presentation.
    - April 1 onwards:
        - We can use this time to solidify and practice our presentation. We will have a little bit of time still if we need to button a few things up but this is not expected. Work can continue towards stretch goals as time allows.

- A list of any additional/stretch goals that could added to project if time allows.

    2D WHT

    Fast Fourier Transform implementation and comparison with WHT

    Polyphonic Tuner

    Different File output types

    Multiple platforms

    Diversity of widgets