Component Specifications

Software Components:

Flask Components:

1. routes.py
    a. Purpose: Defines the various webapp routes that the user will be able to access on their local web server (or when hosted online, the routes for our website). Additionally it connects the HTML front end of the webapp to the rest of the functions in the python backend. routes.py renders the web pages using the html files found in templates.
    b. @app.route('/build')
        i. This is the route where the user will input any data they want to process using our package.
    c. @app.route('/results')
        i. This is the route where the user will see the results of the most recent model simulation.
2. forms.py
    a. Purpose: Hold the formats and fields for the various forms that the user will use to interact with the package.
3. run.py
    a. Purpose: Runs the webapp to begin hosting on their local server.
4. templates
    a. Purpose: A folder that stores the HTML files for rendering the webapp pages.
5. static
    a. Purpose: A folder that stores the various text and image files that the webapp operates with.

Antimony Builder: A set of python functions that take the text data inputted by the user on the Flask webpage and build a text file of antimony

1. init()
    a. Purpose: Save the user's inputs into a dictionary. The dictionary values are the names of the variables and include the variable names for the reactants and products, the initial conditions of the species, the species that are fixed, the kinetic constants, and the boolean determining reversibility. The dictionary keys are the associated labels ('Reactants', 'Products', 'FixedReactants', 'FixedProducts', 'ReactantIC', ProductIC', 'rxnConstants', 'Reversibility')
    b. Inputs: The inputs are string values that come from the text input boxes of the Flask webpage
    c. Outputs: a dictionary of the variables and their values
2. reactionAntimony()
    a. Purpose: To create strings that represent the submitted reaction in antimony form
    b. Inputs:
        i. the dictionary created in save2dict.py
        ii. currentK (int) which represents the amount of k values that have previously been defined

   c. Outputs:
      i. A txt file named Antimony1.txt that contains the strings of antimony of the reactions.
      ii. A currentK value depending on how many k values were defined in that specific reaction.
      iii. A list of strings called kList containing names for the reaction constants
3. conditionsAntimony()
   a. Purpose: To create strings that represent the submitted initial conditions of the defined reaction in antimony form
   b. Inputs:
      i. the dictionary created in save2dict.py
      ii. kList, the list of reaction constant names made in reactionAntimony.py
   c. Outputs:
      i. A txt file named Antimony2.txt that contains the strings of antimony of the conditions of the reaction.

Simulation Runner: A set of python functions that runs a simulation on a given model and saves the plot as an image to the current directory.

1. runSim()
   a. Purpose: to run a simulation on a given antimony file and return a plot as a jpg
   b. Inputs:
      i. No direct inputs to the function but antimony1.txt and antimony2.txt must be in the same directory as runSim.py and be filled by the antimony builder python functions
   c. Outputs:
      i. A jpg image of the steady state graph of the given reaction.
2. resetFiles()
   a. Purpose: To reset the antimony1.txt and antimony2.txt files
   b. Inputs:
      i. No direct inputs to the function
   c. Outputs:
      i. Blank antimony1.txt and blank antimony2.txt files
3. uploadAntimony()
   a. Purpose: To produce the steady state output of an antimony text file on the user's computer.
   b. Inputs: An antimony text file
   c. Outputs: A jpg image of the steady state graph of the given reaction
4. loadSBML()
   a. Purpose: To produce the steady state output of a computational systems biology model from the SimBio BioModels website and plot it.
   b. Inputs: A url to the download link of the model
   c. Outputs: A jpg image of the steady state graph of the given reaction

Interactions to Accomplish Use Cases:

a) The user would build a reaction, input kinetic variables, and then output a steady state plot

To accomplish this use case, the user will input their reactions one by one into the webpage. They submit the reactions iteratively so the user will input all of their variables in comma separated lists. These variables are the reactants, products, initial conditions of reactants and products, which species are fixed, what are the reaction constants, and then they will click a checkbox that indicates whether their reaction is reversible. Once they have typed in their variables and constants they will click 'Add Reaction'. This 'Add Reaction' then calls save2dict(), reactionAntimony(), and conditionAntimony. These functions take the user input from the Flask Webpage and create a dictionary with the values and then create antimony lines which are appended to two different text files. If the user wants to add another reaction they will then repeat the process of typing in the variables and adding reactions until they have completed their model. Then the user will click 'Run Simulation'. This button will call the runSim() function which will append the two text files of antimony (one for the reaction antimony and one containing the initial conditions) and input the combined txt file into a tellurium model and run the simulation. A JPG image of the graph will be exported to the Flask Webpage where the user is brought to another page where they will see the steady state graph.

b) The user would submit an antimony model and then view the resulting steady state plot

To accomplish this use case the user will click 'Choose File' which will open up their file browser. The user can choose what txt file they want and then click open. To submit this file the user must then click 'Finish Model Upload'. This then runs the uploadAntimony() function which takes in the txt file and creates the tellurium simulation. A JPG image is exported to the Flask Webpage and is shown to the user on a different page of the website.

c) The user would submit an SMBL file and then view the resulting steady state plot

To accomplish this use case the user will first go to bioModels, find the model they want to input, right click to download, then copy paste the link into the section labeled SBML File Link. The user then clicks the 'Submit SBML File' which would run the loadSBML() function. The SBML model is loaded into tellurium and the simulation is run. A JPG image is exported to the Flask Webpage and is shown to the user on a different page of the website.