ABX Ninja
GUI Implementation Plan

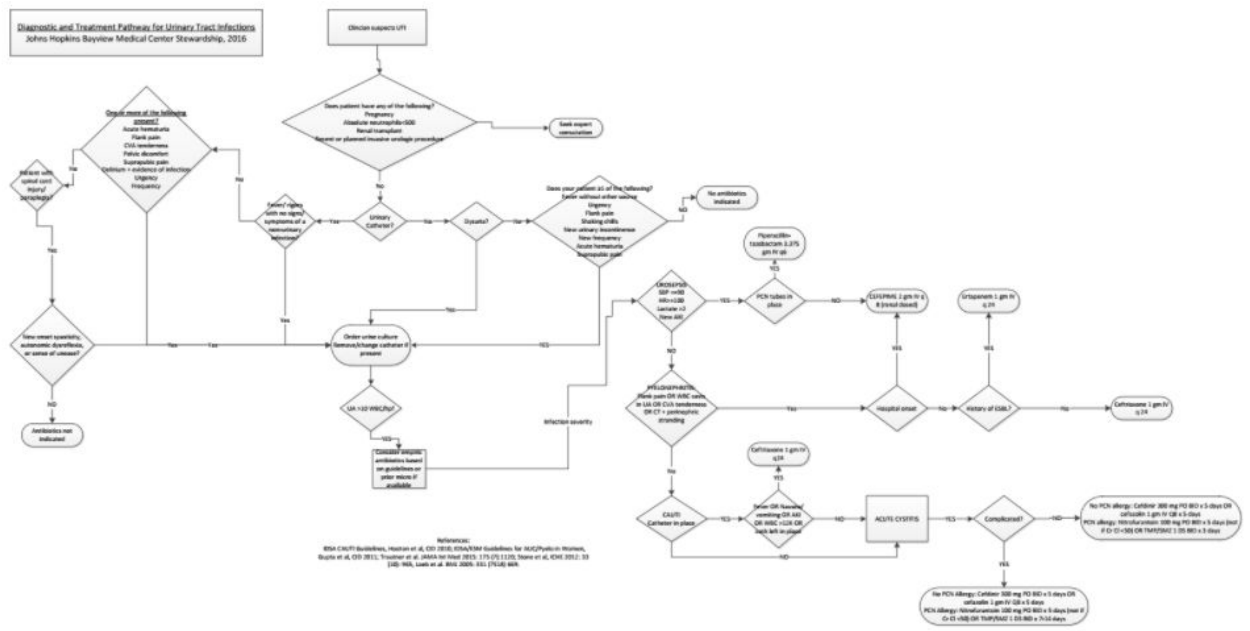
Authors
Katie Hochberg
Allie Sanzi

Mentors
Dr. Jennifer Townsend
Michael Cohen
Andrew Hinton

Vision Statement

Our goal is to create a web application that can be used by clinicians, medical residents, and medical students in order to provide antibiotic recommendations for patients affected by common infections. Centers for Disease Control and Prevention estimates that up to 50% of all the antibiotics prescribed for people are not needed or are not optimally effective as prescribed. This overuse of antibiotics is a dangerous issue facing healthcare in the United States and other countries around the world. To improve the accuracy of antibiotic prescription, Antibiotic Ninja will assist healthcare providers by making an appropriate antibiotic recommendation for patients based on their history, vital signs, and lab results.

Introduction
The assessments and recommendations made by ABX Ninja are generated from decision trees developed by our mentor, Dr. Jenny Townsend. The recommendations at each point are derived from Johns Hopkins antibiotics guidelines and Infectious Diseases Society of America (IDSA) guidelines. Each infection supported by ABX Ninja will have its own decision tree. The decision tree shown below is for Urinary Tract Infections. Each decision tree has nodes with a question that asks about the patient's status, which will be answered by the healthcare provider until they arrive at a recommendation.



Motivation
Currently, in order to input these decision trees, developers have to manually translate the decisions trees into the database using the following four tables:

1. <u>Tree</u> - currently has only three entries which correspond to the three infections we support (Soft Skin and Tissue, Urinary Tract, and Respiratory).
2. <u>Node</u> - contains the information for each decision point in the tree. Each node has a threshold score that must be met to take the "yes" path. If that threshold is not met, then the "no" path will be taken. The threshold score is calculated using a formula encoded in each node that uses the information entered by the healthcare provider.
3. <u>Factors</u> - holds the information for each of the inputs entered by the provider. These factors have a corresponding type to determine which input screen they are displayed on (clinical history, symptoms, vital signs and labs, or image findings), and an input type (multiple choice or text box).
4. <u>Result Node</u> - holds the assessments and recommendations for each outcome of each infection. These result nodes are stored separately from regular nodes because they can be modified for each institution. Storing these separately reduces the amount of space required by the database to store decision tree information.

Here is an example tree and the corresponding tables:

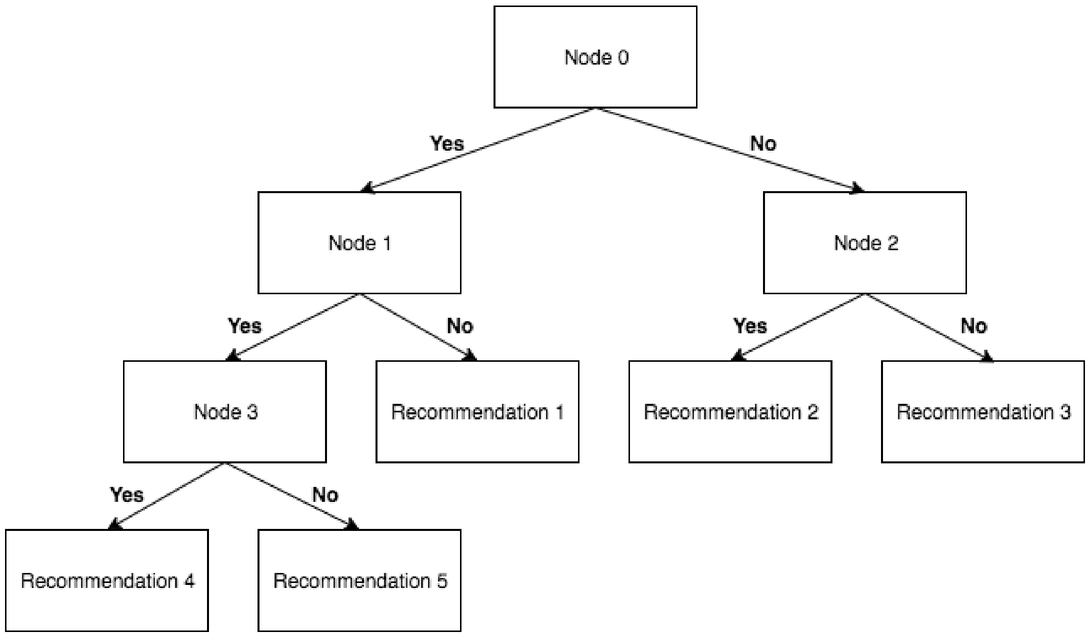


Table 1: Nodes

id	tree	name	display	score	threshold	yes	no
0	1	node0	Node 0	(factors.factor1 > 12 ? 1 : 0) + (factors.factor2 ? 2 : 0) + (factors.factor3 < 5 ? 1 : 0)	2	1	2
1	1	node1	Node 1	(factors.factor4 ? 1 : 0) + (factors.factor5 ? 1 : 0)	1	3	4
2	1	node2	Node 2	(factors.factor6 ? 4 : 0) + (factors.factor7 < 4.5 ? 1 : 0) + (factors.factor8 > 30 ? 2 : 0) + (factors.factor9 <= 100 ? 2 : 0)	5	5	6
3	1	node3	Node 3	(factors.factor10 ? 1 : 0)	1	7	8

Table 2: Results

id	tree	recommendation	assessment	institution
4	1	Recommendation 1	Assessment 1	1
5	1	Recommendation 2	Assessment 2	1
6	1	Recommendation 3	Assessment 3	1
7	1	Recommendation 4	Assessment 4	1
8	1	Recommendation 5	Assessment 5	1

Table 3: Factors

tree	node	name	type	display
1	node0	factor1	lab	Factor 1
1	node0	factor2	clinical	Factor 2
1	node0	factor3	lab	Factor 3
1	node1	factor4	clinical	Factor 4
1	node1	factor5	clinical	Factor 5
1	node2	factor6	clinical	Factor 6
1	node2	factor7	lab	Factor 7
1	node2	factor8	lab	Factor 8
1	node2	factor9	lab	Factor 9
1	node3	factor10	clinical	Factor 10

Decision Tree GUI Description

The goal of this interface is that administrators will be able to edit and create decision trees without developers performing manual conversion.

The GUI will look very similar to the decision trees shown above, however, only the name will be visible in the node. The rest of the information will be viewable by clicking on the node. This will simplify GUI so that it is intuitive to use.

In terms of functionality, administrators will be able to drag and drop new nodes into the tree and update the factors and results associated with those nodes. Nodes can be connected by dragging an arrow from one node to another and providing the relationship (yes/no). To update node information, the node must be selected by the administrator. For a non-result node, an administrator can update the name, the yes/no paths, the threshold score, and the associated factors. Factors will have a display name, an input type (yes/no choice or number), and an expression to determine what value of the factor will contribute to a node's overall score. For a result node, an administrator can update the assessment, directive, and recommended antibiotics. Antibiotics will have a drug name, dose, frequency, and duration. Nodes and result nodes will be indicated by different shapes or colors for clarity.

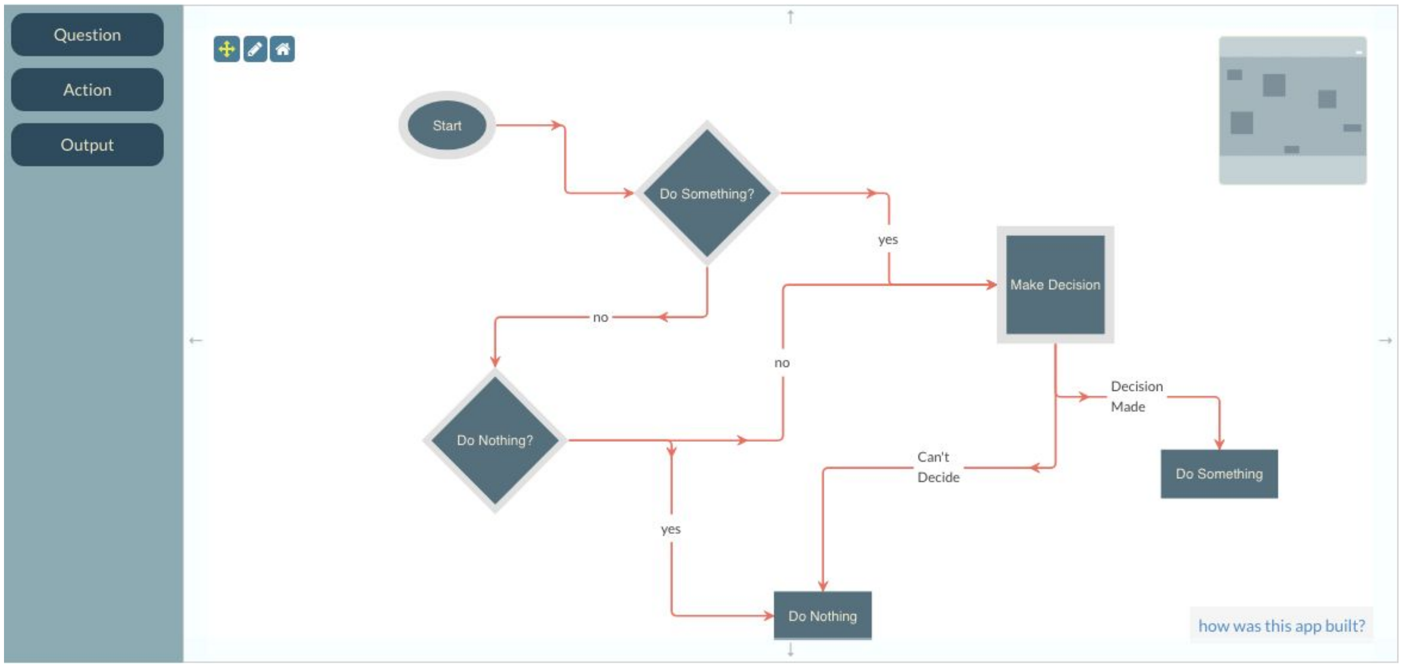
Initial Research
Through our initial research, we found that there are not many publicly available libraries that could be directly used to implement this feature. To implement this GUI from scratch, we found a few options that provide some insight:

GoJS:
http://gojs.net/latest/samples/index.html
This library has many different graphical representations. It uses a JSON object to store the different nodes and objects in the graph, which would be helpful to view how the objects are encoded and displayed. However, this library seems to have graphs that are populated from the backend and not editable from the frontend.

AngularJS Flowchart:
https://www.codeproject.com/Articles/709340/Implementing-a-Flowchart-with-SVG-and-AngularJS
This project has an implementation for a flowchart using AngularJS. It allows multiple inputs and outputs from each node, which is helpful for us to understand since our nodes will have multiple outputs. It also uses JSON objects to store different nodes and objects in the flowchart. However, our project uses Angular 2, which is not compatible with AngularJS, and this implementation does not allow users to create new nodes from the frontend.

Angular UI Tree:
https://angular-ui-tree.github.io/angular-ui-tree/#/basic-example
This is an actual tree implementation. Users can add items from the frontend, unlike the previous two implementations, which is beneficial. It stores the items in the tree as JSON objects, and the structure of these items are closely related to how the nodes from the decision tree will have to be stored. However, the GUI structure is not intuitive our application.

jsPlumb:
https://jsplumbtoolkit.com/docs/toolkit/demo-flowchart-builder.html
See below for a jsPlumb API interface. Users are able to select the type of node to create. In our case, the types of nodes are regular nodes and result nodes. In addition, the user can draw various paths from node to node that are labeled yes or no. Overall, this interface looks very similar to the decision trees for this project. jsPlumb has a community implementation which allows for limited use of their API, however, this would likely require significant work. To access the full version, which would be much easier to integrate with our current implementation, the ABX Ninja team would need to purchase a license.



Ideal Implementation
It would be ideal to purchase a license for the jsPlumb API. ABX Ninja developers could extend the API to have a sidebar pop up when a given node is clicked. This sidebar would display the factor and result information associated with the current node and allow users to edit the nodes separately from the tree. This would result in an uncluttered and intuitive GUI. Once the factor and result information is updated on the frontend, the developers could encode the data to fit the database schema and push the changes to the database.

Implementation Plan B
If the ABX Ninja team is unable to purchase a license for the jsPlumb API, the best option is to build out the GUI from scratch using the other libraries and projects listed above for inspiration. Nodes in the tree should be stored as JSON objects, and the GUI should use that JSON information to populate the tree correctly. Nodes should be styled and permissioned based on their type (node or result) such that users are able to differentiate between them and result nodes have no outgoing paths. Upon clicking a node, a sidebar should appear that contains all of the relevant factor and result information. This should be editable and will need to be pushed to the database when changes are made. This information could also be held in a JSON object that would then be used to create the rows in the database according to the current schema. All nodes should be editable and deletable from the graph. The following lists contains important things to remember when executing this implementation.

Important things to remember

- The start node in the tree must have an ID of 0 and must be continuous because the backend stores the nodes in an array and uses the node's ID as its index into the array.
- Factors can be used in score formulas for multiple nodes, however each node's factors will be determined separately, so checks will need to be done to ensure that factors are not displayed twice to user
- When deleting a node, it is important to update all nodes that it is connected to and remove all of its corresponding information. However, be aware that factors can be used for multiple nodes, so a factor should only be deleted if it is not used by any of the other nodes.