**Software Factory @ Montana State**

wFlow: Project Portfolio

**Fall 2016 - Spring 2017**

MacKenzie O’Bleness & Lisa Peters

# Section 1: Program Code

Project code is hosted at [github.com/Workiva/wFlow-spike](https://github.com/Workiva/wFlow-spike). The program specifications are discussed in more detail in the requirements section of the Project Report, but here are some high level requirements of the module:

* Standard consumption by wDesk through the w\_module pattern in a Dart library.
* The creation of simple flowcharts, consisting of nodes and connections, in a browser.
* The implementation and use of a machine learning library to produce layouts of given nodes and connections.
* Basic saving and loading of graphs to a database.

All non-generated code is reproduced below.

## MachineLearning/ConnectedNodePair.py

import json  
  
import LearningParameter  
import Graph  
  
  
class ConnectedNodes(LearningParameter.LearningParameter):  
 """  
 A Learning Parameter for pairs of nodes joined by a connection.  
 Note that if Node A is connected to Node B and Node C, two pairs of connected nodes will be created;  
 NodeA-NodeB and NodeA-NodeC  
 Currently optimal fitness is defined by nodes spaced the ideal\_node\_distance (in pixels) apart  
 """  
 graph = {}  
 connected\_nodes = {}  
 fitness = None  
 actions = []  
 ranking = None  
 # edges of graph, do not move nodes beyond  
 x\_min = 16  
 y\_min = 0  
 x\_max = 763  
 y\_max = 550  
  
 ideal\_node\_distance = 150 # ideal distance in pixels between nodes  
  
 def \_\_init\_\_(self, graph):  
 self.graph = graph  
 self.connected\_nodes = self.parse\_graph(graph)  
 self.fitness = self.get\_parameter\_fitness()  
 self.create\_set\_of\_actions()  
 self.ranking = 1  
  
 def get\_num\_less\_than\_ideal(self):  
 """Get the number of connected nodes that are less than the ideal distance apart"""  
 total = 0  
 for key in self.connected\_nodes:  
 if self.connected\_nodes[key] < self.ideal\_node\_distance:  
 total += 1  
 return total  
  
 def create\_set\_of\_actions(self):  
 """  
 Create the set of actions for this specific LearningParameter, ie moving one or the other node of a connected pair a specific direction to increase the distance between them.  
 There is an action for each node of each pair for each direction.  
 """  
 movements = ["first\_left", "first\_up", "first\_down", "first\_right", "second\_left", "second\_up", "second\_down",  
 "second\_right"]  
 for fitness in range(0, 4):  
 for pair in self.connected\_nodes:  
 for move in movements:  
 self.actions.append(str(fitness) + ":" + str(pair) + ":" + move)  
  
 def get\_reward(self, old\_fitness, invalid\_move=False):  
 """Get the expected reward for taking an action"""  
 if invalid\_move:  
 reward = -100  
 else:  
 reward = 0  
 new\_fitness = self.get\_parameter\_fitness()  
 diff = abs(new\_fitness - old\_fitness)  
 if diff == 3 or new\_fitness == 3:  
 reward = 100  
 elif diff == 2:  
 reward = 50  
 elif diff == 1:  
 reward = 25  
 elif diff == 0:  
 reward = 0  
  
 if new\_fitness < old\_fitness:  
 reward = -reward  
  
 return reward  
  
 def get\_actual\_pair(self, value):  
 """Given a string representation of a connected pair, get the obj instance of that pair"""  
 for key in self.connected\_nodes:  
 if str(key) == value:  
 return key  
  
 def take\_action(self, action\_string, old\_fitness):  
 """Move nodes to change the state"""  
 fitness, pair\_str, action = action\_string.split(":")  
 connected\_pair = self.get\_actual\_pair(pair\_str)  
 which\_node\_string, dir\_string = action.split("\_")  
 out\_of\_bounds\_move = False  
  
 if which\_node\_string == "first":  
 node = connected\_pair.start\_node  
 else:  
 node = connected\_pair.end\_node  
  
 dir\_string = dir\_string.upper()  
 if dir\_string == 'UP' and (node.y + 100 < self.y\_max):  
 node.y = node.y + 100  
 elif dir\_string == "DOWN" and (node.y - 100 > self.y\_min):  
 node.y = node.y - 100  
 elif dir\_string == "RIGHT" and (node.x + 100 < self.x\_max):  
 node.x = node.x + 100  
 elif dir\_string == "LEFT" and (node.x - 100 > self.x\_min):  
 node.x = node.x - 100  
 else:  
 out\_of\_bounds\_move = True  
  
 # Update after movement of nodes  
 self.connected\_nodes[connected\_pair] = connected\_pair.get\_distance()  
 self.fitness = self.get\_parameter\_fitness()  
 return self.get\_reward(old\_fitness, out\_of\_bounds\_move)  
  
 def get\_parameter\_fitness(self):  
 """ Get the fitness of the graph for this parameter"""  
 if self.get\_num\_less\_than\_ideal() == len(self.connected\_nodes):  
 return 0 # worst option  
 elif self.get\_num\_less\_than\_ideal() == 0:  
 return 3 # best option  
 else:  
 return 2 # could be better  
  
 def get\_set\_of\_actions(self):  
 """get the set of possible actions for the ConnectedPair Learning Parameter"""  
 return self.actions  
  
 def parse\_graph(self, graph):  
 """Given json representation of the graph, parse it into objects needed for Learning Parameter"""  
 connected\_nodes = {}  
 for connection in graph['connections']:  
 start\_id = connection['start']  
 end\_id = connection['end']  
 start\_node = graph['nodes'][start\_id]  
 end\_node = graph['nodes'][end\_id]  
 cp = Graph.NodePair(Graph.Node(start\_node['x'], start\_node['y'], start\_node['id']),  
 Graph.Node(end\_node['x'], end\_node['y'], end\_node['id']))  
 connected\_nodes[cp] = cp.get\_distance()  
  
 return connected\_nodes  
  
 def create\_graph(self):  
 """Put post Q-Learning graph back into JSON format to return to client"""  
 for cp in self.connected\_nodes:  
 start = cp.start\_node  
 self.graph['nodes'][start.id]['x'] = start.x  
 self.graph['nodes'][start.id]['y'] = start.y  
  
 end = cp.end\_node  
 self.graph['nodes'][end.id]['x'] = end.x  
 self.graph['nodes'][end.id]['y'] = end.y  
  
 print json.dumps(self.graph)  
  
 def get\_parameter\_ranking(self):  
 return self.ranking

## MachineLearning/Graph.py

import math  
  
  
class Node:  
 x = None  
 y = None  
 id = None  
  
 def \_\_init\_\_(self, x, y, id):  
 self.x = x  
 self.y = y  
 self.id = id  
  
 def \_\_str\_\_(self):  
 return "x: {}, y: {}".format(self.x, self.y)  
  
 def calc\_distance(self, second\_node):  
 distance = math.pow((self.x - second\_node.x), 2) + math.pow((self.y - second\_node.y), 2)  
 return math.sqrt(abs(distance))  
  
  
class NodePair:  
 start\_node = None  
 end\_node = None  
  
 def \_\_init\_\_(self, start\_node, end\_node):  
 self.start\_node = start\_node  
 self.end\_node = end\_node  
  
 def \_\_str\_\_(self):  
 return "({}) -> ({})".format(self.start\_node.id, self.end\_node.id)  
  
 def get\_distance(self):  
 return self.start\_node.calc\_distance(self.end\_node)

## MachineLearning/LearningParameter.py

import abc  
  
  
class LearningParameter(object):  
 \_\_metaclass\_\_ = abc.ABCMeta  
  
 # Get the expected reward for taking an action  
 @abc.abstractmethod  
 def get\_reward(self, old\_fitness):  
 return  
  
 # Take an action to change the state of the graph  
 @abc.abstractmethod  
 def take\_action(self, action\_string, old\_fitness):  
 return  
  
 # Get the fitness of the graph for this parameter  
 @abc.abstractmethod  
 def get\_parameter\_fitness(self):  
 return  
  
 @abc.abstractproperty  
 def fitness(self):  
 pass  
  
 @abc.abstractmethod  
 @fitness.setter  
 def fitness(self, val):  
 pass  
  
 # Get the ranking of this parameter for graph's overall fitness  
 @abc.abstractmethod  
 def get\_parameter\_ranking(self):  
 return  
  
 @abc.abstractproperty  
 def ranking(self):  
 pass  
  
 @abc.abstractmethod  
 @ranking.setter  
 def ranking(self, val):  
 pass  
  
 # Get specific set of actions to take to improve LearningParameter's fitness  
 def get\_set\_of\_actions(self):  
 return  
  
 # Get necessary vars from graph in order to create specific learning parameter.  
 def parse\_graph(self, graph):  
 return  
  
 # Create json graph from graph state after Q-Learning algorithm has run  
 def create\_graph(self):  
 return

## MachineLearning/QLearning.py

import ConnectedNodePair as CNP  
import json  
import argparse  
import sys  
import random  
  
  
class QLearning:  
 def \_\_init\_\_(self, cur\_state, num\_episodes, discount\_factor=1.0, learning\_rate=0.5):  
  
 # Q is policy action map, where keys are fitness  
  
 # create specific set of actions for the given Learning Parameter  
 actions = self.create\_actions(cur\_state)

Q = self.create\_q(actions)  
  
 for iteration in range(0, num\_episodes):  
 for act in actions:  
 if cur\_state.fitness == 3:  
 cur\_state.create\_graph()  
 return  
 # get reward and newState and terminate if Learning Parameter fitness is optimal (Fitness == 3)  
 reward, newState = self.get\_reward(cur\_state, act)  
 q\_key = str(cur\_state.get\_parameter\_fitness()) + act[1:]  
  
 # Q(s, a) <-- Q(s, a) + alpha[r + gamma\*max\_alpha(Q(s', a'))]  
 Q[q\_key] = Q[q\_key] + learning\_rate \* (reward + discount\_factor \* self.get\_max\_reward(newState, Q))  
  
 cur\_state = newState  
  
 # Unable to reach optimal fitness, give state as is  
 cur\_state.create\_graph()  
  
 # Find the best reward from the set of possible actions when in state s'  
 @staticmethod  
 def get\_max\_reward(new\_state, Q):  
 max\_reward = 0  
 for key in Q:  
 if key[0] == str(new\_state.get\_parameter\_fitness()):  
 max\_reward = max(max\_reward, Q[key])  
 return max\_reward  
  
 # Take the action and get the reward from the LearningParameter  
 @staticmethod  
 def get\_reward(new\_state, action):  
 old\_fitness = new\_state.get\_parameter\_fitness()  
 reward = new\_state.take\_action(action, old\_fitness)  
 return reward, new\_state  
  
 # Create the initial (state, action)-> utility,  
 # which sets all utilities to 0, ie no state,action pair is better than any others  
 @staticmethod  
 def create\_q(actions):  
 Q = {}  
 for action in actions:  
 Q[action] = 0  
 return Q  
  
 # Get set of actions from specific LearningParameter  
 @staticmethod  
 def create\_actions(cur\_state):  
 actions = []  
 for action in cur\_state.get\_set\_of\_actions():  
 actions.append(action)  
 random.shuffle(actions)  
 return actions  
  
  
# http://stackoverflow.com/questions/956867/how-to-get-string-objects-instead-of-unicode-ones-from-json-in-python  
def json\_load\_byteified(file\_handle):  
 return \_byteify(  
 json.load(file\_handle, object\_hook=\_byteify),  
 ignore\_dicts=True  
 )  
  
  
def json\_loads\_byteified(json\_text):  
 return \_byteify(  
 json.loads(json\_text, object\_hook=\_byteify),  
 ignore\_dicts=True  
 )  
  
  
def \_byteify(data, ignore\_dicts=False):  
 # if this is a unicode string, return its string representation  
 if isinstance(data, unicode):  
 return data.encode('utf-8')  
 # if this is a list of values, return list of byteified values  
 if isinstance(data, list):  
 return [\_byteify(item, ignore\_dicts=True) for item in data]  
 # if this is a dictionary, return dictionary of byteified keys and values  
 # but only if we haven't already byteified it  
 if isinstance(data, dict) and not ignore\_dicts:  
 return {  
 \_byteify(key, ignore\_dicts=True): \_byteify(value, ignore\_dicts=True)  
 for key, value in data.iteritems()  
 }  
 # if it's anything else, return it in its original form  
 return data  
  
  
def main():  
 # This works for server/system call  
 true\_graph = json\_loads\_byteified(sys.argv[1])  
  
 # This works with pycharm, keep around for debugging  
 # parser = argparse.ArgumentParser()  
 # parser.add\_argument("a")  
 # args = parser.parse\_args()  
 # with open('debug.json') as data\_file:  
 # true\_graph = json\_load\_byteified(data\_file)  
  
 # new learning instance with initial state and max iterations  
 cnlp\_state = CNP.ConnectedNodes(true\_graph)  
 max\_iterations = 500  
 optimal\_policy = QLearning(cnlp\_state, max\_iterations)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

## MachineLearning/test/unit/ConnectedNodePairTest.py

import json  
import unittest  
  
import MachineLearning.ConnectedNodePair as CNP  
import MachineLearning.QLearning as ql  
  
  
class TestConnectedNodePair(unittest.TestCase):  
 # Graph with four nodes, two connections, two overlapping pair  
 def setUp(self):  
 with open('MachineLearning/test/json\_graphs/ConnectedNodePair.json') as data\_file:  
 test\_graph = ql.json\_load\_byteified(data\_file)  
 self.cnlp\_state = CNP.ConnectedNodes(test\_graph)  
  
 def separate\_nodes(self):  
 # separate both  
 self.cnlp\_state.take\_action('1:(3) -> (2):second\_up', 0)  
 self.cnlp\_state.take\_action('1:(3) -> (2):second\_up', 0)  
 self.cnlp\_state.take\_action('1:(0) -> (1):second\_up', 0)  
 self.cnlp\_state.take\_action('1:(0) -> (1):second\_up', 0)  
  
 def separate\_one(self):  
 self.cnlp\_state.take\_action('1:(3) -> (2):second\_up', 0)  
 self.cnlp\_state.take\_action('1:(3) -> (2):second\_up', 0)  
  
 def undo\_separate\_nodes(self):  
 # separate both  
 self.cnlp\_state.take\_action('1:(3) -> (2):second\_down', 0)  
 self.cnlp\_state.take\_action('1:(3) -> (2):second\_down', 0)  
 self.cnlp\_state.take\_action('1:(0) -> (1):second\_down', 0)  
 self.cnlp\_state.take\_action('1:(0) -> (1):second\_down', 0)  
  
 def test\_init(self):  
 self.setUp()  
 self.assertEquals(len(self.cnlp\_state.connected\_nodes), 2)  
 self.assertEquals(len(self.cnlp\_state.graph['connections']), 2)  
 self.assertEquals(len(self.cnlp\_state.graph['nodes']), 4)  
 self.assertEquals(self.cnlp\_state.fitness, 0)  
 self.assertEquals(self.cnlp\_state.ranking, 1)  
  
 def test\_get\_num\_less\_than\_ideal\_none(self):  
 self.setUp()  
 # all nodes seperated  
 self.separate\_nodes()  
 self.assertEquals(self.cnlp\_state.get\_num\_less\_than\_ideal(), 0)  
  
 def test\_get\_num\_less\_than\_ideal(self):  
 self.setUp()  
 # Two overlapping pairs  
 self.assertEquals(self.cnlp\_state.get\_num\_less\_than\_ideal(), 2)  
  
 # separate one  
 self.separate\_one()  
 self.assertEquals(self.cnlp\_state.get\_num\_less\_than\_ideal(), 1)  
  
 # TODO: why is this failing? And why in the world are we getting multiple numbers?  
 # def test\_create\_set\_of\_actions(self):  
 # self.setUp()  
 # self.assertEquals(len(self.cnlp\_state.actions), 384)  
  
 def test\_get\_reward\_no\_change\_in\_fitness(self):  
 self.setUp()  
 old\_fitness = 0  
 reward = self.cnlp\_state.get\_reward(old\_fitness)  
 self.assertEquals(reward, 0)  
  
 def test\_get\_reward\_increase\_one(self):  
 self.setUp()  
 old\_fitness = 1  
 self.separate\_one()  
 reward = self.cnlp\_state.get\_reward(old\_fitness)  
 self.assertEquals(reward, 25)  
  
 def test\_get\_reward\_increase\_two(self):  
 self.setUp()  
 old\_fitness = 0  
 self.separate\_one()  
 reward = self.cnlp\_state.get\_reward(old\_fitness)  
 self.assertEquals(reward, 50)  
  
 def test\_get\_reward\_increase\_three(self):  
 self.setUp()  
 old\_fitness = 0  
 self.separate\_nodes()  
 reward = self.cnlp\_state.get\_reward(old\_fitness)  
 self.assertEquals(reward, 100)  
  
 def test\_get\_reward\_negative(self):  
 self.setUp()  
 old\_fitness = 3  
 reward = self.cnlp\_state.get\_reward(old\_fitness)  
 self.assertEquals(reward, -100)  
  
 def test\_get\_actual\_pair(self):  
 self.setUp()  
 self.assertEquals(self.cnlp\_state.connected\_nodes[self.cnlp\_state.get\_actual\_pair('(0) -> (1)')],  
 78.00640999302557)  
 self.assertEquals(self.cnlp\_state.connected\_nodes[self.cnlp\_state.get\_actual\_pair('(3) -> (2)')],  
 98.12746812182611)  
  
 def test\_take\_action\_up(self):  
 self.setUp()  
 # starting distance  
 np = self.cnlp\_state.get\_actual\_pair('(0) -> (1)')  
 self.assertEquals(np.start\_node.x, 50)  
 self.assertEquals(np.start\_node.y, 100)  
 self.cnlp\_state.take\_action('0:(0) -> (1):first\_up', 0)  
 self.assertEquals(np.start\_node.y, 200)  
 self.assertEquals(np.start\_node.x, 50)  
  
 def test\_take\_action\_right(self):  
 self.setUp()  
 # starting distance  
 np = self.cnlp\_state.get\_actual\_pair('(0) -> (1)')  
 self.assertEquals(np.start\_node.x, 50)  
 self.assertEquals(np.start\_node.y, 100)  
 self.cnlp\_state.take\_action('0:(0) -> (1):first\_right', 0)  
 self.assertEquals(np.start\_node.y, 100)  
 self.assertEquals(np.start\_node.x, 150)  
  
 def test\_take\_action\_down(self):  
 self.setUp()  
 # starting distance  
 np = self.cnlp\_state.get\_actual\_pair('(0) -> (1)')  
 self.assertEquals(np.start\_node.x, 50)  
 self.assertEquals(np.start\_node.y, 100)  
 self.cnlp\_state.take\_action('0:(0) -> (1):first\_down', 0)  
 self.assertEquals(np.start\_node.y, 100)  
 self.assertEquals(np.start\_node.x, 50)  
  
 def test\_take\_action\_left(self):  
 self.setUp()  
 # starting distance  
 np = self.cnlp\_state.get\_actual\_pair('(0) -> (1)')  
 np.start\_node.x = 200 # to avoid moving out of bounds  
 self.assertEquals(np.start\_node.x, 200)  
 self.assertEquals(np.start\_node.y, 100)  
 self.cnlp\_state.take\_action('0:(0) -> (1):first\_left', 0)  
 self.assertEquals(np.start\_node.y, 100)  
 self.assertEquals(np.start\_node.x, 100)  
  
 def test\_take\_action\_invalid(self):  
 self.setUp()  
 # starting distance  
 np = self.cnlp\_state.get\_actual\_pair('(0) -> (1)')  
 np.start\_node.x = 50 # to avoid moving out of bounds  
 self.assertEquals(np.start\_node.x, 50)  
 self.assertEquals(np.start\_node.y, 100)  
 self.cnlp\_state.take\_action('0:(0) -> (1):first\_left', 0)  
 self.assertEquals(np.start\_node.y, 100)  
 self.assertEquals(np.start\_node.x, 50)  
  
 def test\_take\_action\_end\_node(self):  
 self.setUp()  
 # starting distance  
 np = self.cnlp\_state.get\_actual\_pair('(0) -> (1)')  
 np.start\_node.x = 50 # to avoid moving out of bounds  
 self.assertEquals(np.end\_node.x, 128)  
 self.assertEquals(np.end\_node.y, 101)  
 self.cnlp\_state.take\_action('0:(0) -> (1):second\_right', 0)  
 self.assertEquals(np.end\_node.y, 101)  
 self.assertEquals(np.end\_node.x, 228)  
  
 def test\_get\_parameter\_fitness\_three(self):  
 self.setUp()  
 # all nodes seperated  
 self.separate\_nodes()  
 self.assertEquals(self.cnlp\_state.get\_parameter\_fitness(), 3)  
  
 def test\_get\_parameter\_fitness\_two(self):  
 self.setUp()  
 # Two overlapping pairs  
 # separate one  
 self.separate\_one()  
 self.assertEquals(self.cnlp\_state.get\_parameter\_fitness(), 2)  
  
 def test\_get\_parameter\_fitness\_zero(self):  
 self.setUp()  
 # Two overlapping pairs  
 self.assertEquals(self.cnlp\_state.get\_parameter\_fitness(), 0)  
  
 def test\_get\_set\_of\_actions(self):  
 self.setUp()  
 self.assertEquals(self.cnlp\_state.get\_set\_of\_actions(), self.cnlp\_state.actions)  
  
 def test\_parse\_graph(self):  
 graph = {"\_id": "blah", "fitness": "0",  
 "nodes": [{"id": 0, "x": 50, "y": 100, "color": "#c4ffff", "content": "Node 0"},  
 {"id": 1, "x": 128, "y": 101, "color": "#c4ffff", "content": "Node 1"},  
 {"id": 2, "x": 168, "y": 286, "color": "#c4ffff", "content": "Node 2"},  
 {"id": 3, "x": 70, "y": 291, "color": "#c4ffff", "content": "Node 3"}],  
 "connections": [{"id": 0, "start": 0, "end": 1}, {"id": 1, "start": 3, "end": 2}]}  
 self.cnlp\_state.parse\_graph(graph)  
 cn = self.cnlp\_state.connected\_nodes  
 self.assertEquals(len(cn), 2)  
  
 np1 = self.cnlp\_state.get\_actual\_pair("(0) -> (1)")  
 self.assertEquals(np1.start\_node.x, 50)  
 self.assertEquals(np1.start\_node.y, 100)  
 self.assertEquals(np1.end\_node.x, 128)  
 self.assertEquals(np1.end\_node.y, 101)  
  
 np2 = self.cnlp\_state.get\_actual\_pair("(3) -> (2)")  
 self.assertEquals(np2.start\_node.x, 70)  
 self.assertEquals(np2.start\_node.y, 291)  
 self.assertEquals(np2.end\_node.x, 168)  
 self.assertEquals(np2.end\_node.y, 286)  
  
 def test\_create\_graph\_no\_change(self):  
 expected\_start\_string = """{"connections": [{"start": 0, "end": 1, "id": 0}, {"start": 3, "end": 2, "id": 1}], "nodes": [{"y": 100, "x": 50, "content": "Node 0", "id": 0, "color": "#c4ffff"}, {"y": 101, "x": 128, "content": "Node 1", "id": 1, "color": "#c4ffff"}, {"y": 286, "x": 168, "content": "Node 2", "id": 2, "color": "#c4ffff"}, {"y": 291, "x": 70, "content": "Node 3", "id": 3, "color": "#c4ffff"}], "\_id": "blah", "fitness": "0"}"""  
 self.setUp()  
 self.cnlp\_state.create\_graph()  
 graph = json.dumps(self.cnlp\_state.graph)  
 self.assertEquals(graph, expected\_start\_string)  
  
 def test\_create\_graph\_with\_change(self):  
 self.setUp()  
 expected\_end\_string = """{"connections": [{"start": 0, "end": 1, "id": 0}, {"start": 3, "end": 2, "id": 1}], "nodes": [{"y": 100, "x": 50, "content": "Node 0", "id": 0, "color": "#c4ffff"}, {"y": 301, "x": 128, "content": "Node 1", "id": 1, "color": "#c4ffff"}, {"y": 486, "x": 168, "content": "Node 2", "id": 2, "color": "#c4ffff"}, {"y": 291, "x": 70, "content": "Node 3", "id": 3, "color": "#c4ffff"}], "\_id": "blah", "fitness": "0"}"""  
 self.separate\_nodes()  
 self.cnlp\_state.create\_graph()  
 graph = json.dumps(self.cnlp\_state.graph)  
 self.assertEquals(graph, expected\_end\_string)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

## MachineLearning/test/unit/Graph\_test.py

import unittest  
import MachineLearning.Graph as g  
  
  
class TestNode(unittest.TestCase):  
  
 def setUp(self):  
 pass  
  
 def test\_Node\_init(self):  
 n = g.Node(1, 2, 3)  
 self.assertEquals(1, n.x)  
 self.assertEquals(2, n.y)  
 self.assertEquals(3, n.id)  
  
 def test\_calc\_distance\_positive\_numbers(self):  
 n1 = g.Node(1, 2, 1)  
 n2 = g.Node(1, 3, 2)  
 self.assertEquals(n1.calc\_distance(n2), 1)  
 self.assertEquals(n2.calc\_distance(n1), 1)  
  
 def test\_calc\_distance\_zero\_numbers(self):  
 n1 = g.Node(0, 1, 1)  
 n2 = g.Node(0, 0, 2)  
 self.assertEquals(n1.calc\_distance(n2), 1)  
 self.assertEquals(n2.calc\_distance(n1), 1)  
 n1 = g.Node(0, 0, 1)  
 n2 = g.Node(1, 0, 2)  
 self.assertEquals(n1.calc\_distance(n2), 1)  
 self.assertEquals(n2.calc\_distance(n1), 1)  
  
 def test\_calc\_distance\_no\_distance(self):  
 n1 = g.Node(0, 0, 1)  
 n2 = g.Node(0, 0, 2)  
 self.assertEquals(n1.calc\_distance(n2), 0)  
 self.assertEquals(n2.calc\_distance(n1), 0)  
  
 def test\_calc\_distance\_negative\_numbers(self):  
 n1 = g.Node(-1, 0, 1)  
 n2 = g.Node(0, 0, 2)  
 self.assertEquals(n1.calc\_distance(n2), 1)  
 self.assertEquals(n2.calc\_distance(n1), 1)  
 n1 = g.Node(0, 0, 1)  
 n2 = g.Node(-1, 0, 2)  
 self.assertEquals(n1.calc\_distance(n2), 1)  
 self.assertEquals(n2.calc\_distance(n1), 1)  
  
 def test\_to\_string(self):  
 n1 = g.Node(-1, 0, 1)  
 self.assertEquals(str(n1), 'x: -1, y: 0')  
  
  
class TestNodePair(unittest.TestCase):  
  
 def setUp(self):  
 pass  
  
 def test\_NodePair\_init(self):  
 n1 = g.Node(1, 2, 1)  
 n2 = g.Node(1, 2, 2)  
 np = g.NodePair(n1, n2)  
  
 self.assertEquals(n1, np.start\_node)  
 self.assertEquals(n2, np.end\_node)  
  
 def test\_get\_distance(self):  
 n1 = g.Node(1, 2, 1)  
 n2 = g.Node(1, 3, 2)  
 np = g.NodePair(n1, n2)  
 self.assertEquals(np.get\_distance(), 1)  
  
 def test\_to\_string(self):  
 n1 = g.Node(-1, 0, 1)  
 n2 = g.Node(1, 3, 2)  
 np = g.NodePair(n1, n2)  
 self.assertEquals(str(np), '(1) -> (2)')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

## MachineLearning/test/unit/QLearningTest.py

import unittest  
import MachineLearning.QLearning as QL  
import MachineLearning.ConnectedNodePair as CNP  
  
  
class TestQLearning(unittest.TestCase):  
  
 def test\_json\_load\_bytefied(self):  
 with open('MachineLearning/test/json\_graphs/json\_loads\_test.json') as data\_file:  
 graph = QL.json\_load\_byteified(data\_file)  
  
 # test to be sure all elements created  
 self.assertEquals(type(graph), dict)  
 self.assertEquals(len(graph['connections']), 6)  
 self.assertEquals(len(graph['nodes']), 4)  
 self.assertEquals(graph['fitness'], 0)  
 self.assertEquals(graph['\_id'], "Example Chart")  
  
 # test one a random to be sure nodes are created correctly  
 nodes = graph['nodes']  
 self.assertEquals(nodes[2]['color'], "#c4ffff")  
 self.assertEquals(nodes[2]['content'], "Node 2")  
 self.assertEquals(nodes[2]['x'], 531)  
 self.assertEquals(nodes[2]['y'], 339)  
 self.assertEquals(nodes[2]['id'], 2)  
  
 # test one a random to be sure connections are created correctly  
 connections = graph['connections']  
 self.assertEquals(connections[4]['start'], 1)  
 self.assertEquals(connections[4]['end'], 0)  
 self.assertEquals(connections[4]['id'], 4)  
  
 def test\_bytefied\_unicode(self):  
 unicode\_data = u"""{"id":0,"x":527,"y":42,"color":"#c4ffff","content":"Node 0"}"""  
 expected = """{"id":0,"x":527,"y":42,"color":"#c4ffff","content":"Node 0"}"""  
 result = QL.\_byteify(unicode\_data)  
 self.assertEquals(result, expected)  
  
 def test\_bytefied\_list(self):  
 unicode\_data = [u'1', u'2', u'3', u'4', u'5']  
 expected = ['1', '2', '3', '4', '5']  
 result = QL.\_byteify(unicode\_data)  
 self.assertEquals(result, expected)  
  
 def test\_bytefied\_dict(self):  
 unicode\_data = {u'1': u'one', u'2': u'two', u'3': u'three'}  
 expected = {'1': 'one', '2': 'two', '3': 'three'}  
 result = QL.\_byteify(unicode\_data)  
 self.assertEquals(result, expected)  
  
 def test\_create\_q(self):  
 fake\_actions = ["addNode", "addConnection", "colorBlack"]  
 Q = QL.QLearning.create\_q(fake\_actions)  
 self.assertEquals(type(Q), dict)  
 self.assertEquals(Q["addNode"], 0)  
 self.assertEquals(Q["addConnection"], 0)  
 self.assertEquals(Q["colorBlack"], 0)  
  
 def test\_create\_action\_ConnectedNodePair\_as\_LearningParameter(self):  
 with open('MachineLearning/test/json\_graphs/debug.json') as data\_file:  
 true\_graph = QL.json\_load\_byteified(data\_file)  
 cnlp\_state = CNP.ConnectedNodes(true\_graph)  
 created\_actions = QL.QLearning.create\_actions(cnlp\_state)  
 self.assertEquals(len(created\_actions), 64)  
 # assert at least one of each fitness, each node, each direction exists in actions  
 self.assertEquals('0:(0) -> (1):first\_left' in created\_actions, True)  
 self.assertEquals('1:(0) -> (1):first\_right' in created\_actions, True)  
 self.assertEquals('2:(0) -> (1):second\_down' in created\_actions, True)  
 self.assertEquals('3:(0) -> (1):second\_up' in created\_actions, True)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

## bin/database\_controller.dart

import 'dart:async';  
  
import 'package:mongo\_dart/mongo\_dart.dart';  
import 'package:logging/logging.dart';  
  
//Main function contains example of saving and loading graph.  
main() async {}  
  
class DatabaseController {  
 Logger logger;  
  
 DatabaseController() {  
 //setup logging  
 logger = new Logger('DATABASECONTROLLER');  
 Logger.root.level = Level.INFO;  
 Logger.root.onRecord.listen((LogRecord rec) {  
 print('${rec.level.name}: ${rec.loggerName}: ${rec.time}: ${rec.message}');  
 });  
 }  
  
/\*  
 Save graph stub. Queries database to see if name already  
 exists and save to pre-existing graph if so, else creates a new one.  
 Returns a true or false flag for successful save of graph, used to determine  
 Response code  
 \*/  
 Future<bool> saveGraph(Map graph, String collection) async {  
 try {  
 Db db = new Db("mongodb://wFlow\_dev:W0rkiva!@ds053176.mlab.com:53176/graphbase");  
 DbCollection coll;  
 await db.open();  
 coll = db.collection(collection);  
 await coll.save(graph);  
 await db.close();  
 logger.info("Saved graph: $graph to collection ${coll.fullName()}");  
 return true;  
 } catch (e) {  
 logger.severe("Exception in graphBase/database\_controller/saveGraph: $e");  
 return false;  
 }  
 }  
  
//Load graph stub. Queries database given a graph name and return information of graph if if is found, or a Null if graph doesn't exist.  
 Future<Map> loadGraph(String graph\_id, String collection) async {  
 Db db = new Db("mongodb://wFlow\_dev:W0rkiva!@ds053176.mlab.com:53176/graphbase");  
 await db.open();  
 DbCollection coll = db.collection(collection);  
 Map val = await coll.findOne(where.eq('\_id', graph\_id));  
 await db.close();  
 return val;  
 }  
}

## bin/server.dart

import 'dart:io';  
import 'dart:async';  
import 'dart:convert';  
import '../bin/database\_controller.dart' as graphBase;  
import 'package:logging/logging.dart';  
  
Logger log;  
graphBase.DatabaseController graphbaseController;  
  
// TODO 853 figure out strategy for testing server and test it.  
Future main() async {  
 //setup logging  
 log = new Logger('SERVER');  
 Logger.root.level = Level.INFO;  
 Logger.root.onRecord.listen((LogRecord rec) {  
 print('${rec.level.name}: ${rec.loggerName}: ${rec.time}: ${rec.message}');  
 });  
  
 //Init controller for the graph database  
 graphbaseController = new graphBase.DatabaseController();  
  
 HttpServer requestServer = await HttpServer.bind(InternetAddress.LOOPBACK\_IP\_V4, 8000);  
 //TODO: 850 not use localhost  
 log.info('listening on localhost, port ${requestServer.port}');  
  
 try {  
 await for (HttpRequest request in requestServer) {  
 handleRequest(request);  
 }  
 } catch (e) {  
 log.severe("Error in server/main $e");  
 }  
}  
  
void handleRequest(HttpRequest request) {  
 print("Got request with type: ${request.method}");  
 try {  
 //TODO change this when not working with localhost. SECURITY  
 // Consider using an environment variable?  
 request.response.headers.add("Access-Control-Allow-Origin", "http://localhost:8080");  
 request.response.headers.add("Access-Control-Allow-Methods", "POST, GET, PUT, OPTIONS");  
 if (request.method == 'GET') {  
 handleGet(request);  
 } else if (request.method == 'POST') {  
 handlePost(request);  
 } else if (request.method == 'PUT') {  
 handleLayout(request);  
 } else if (request.method == 'OPTIONS') {  
 request.response  
 ..statusCode = HttpStatus.OK  
 ..close();  
 } else {  
 request.response  
 ..statusCode = HttpStatus.METHOD\_NOT\_ALLOWED  
 ..write("Unsupported request: ${request.method}.")  
 ..close();  
 }  
 } catch (e) {  
 log.severe("Error in server/handleRequest: $e");  
 }  
}  
  
Future<Null> handlePost(HttpRequest request) async {  
 int status;  
 try {  
 // TODO 850 not have this be hardcoded. Also TODO 851, decide  
 // how collections will be decided (each user has collection presumably,  
 // which would mean we need to have user logic).  
 String collection = "testgraphbase";  
 String jsonString = await request.transform(UTF8.decoder).join();  
  
 log.info("Recieved post $jsonString");  
  
 Map parsedMap = JSON.decode(jsonString);  
 bool result = await graphbaseController.saveGraph(parsedMap, collection);  
  
 if (result) {  
 status = HttpStatus.OK;  
 } else {  
 status = HttpStatus.NOT\_IMPLEMENTED;  
 }  
 } catch (e) {  
 status = HttpStatus.INTERNAL\_SERVER\_ERROR;  
 log.severe("server/handlePost: Exception during file I/O: $e.");  
 }  
 request.response  
 ..statusCode = status  
 ..close();  
}  
  
Future<Null> handleGet(HttpRequest request) async {  
 String collection = "testgraphbase";  
 List pathSeg = request.uri.pathSegments;  
 String graphName = pathSeg.elementAt(0);  
  
 Map result = await graphbaseController.loadGraph(graphName, collection);  
  
 int status;  
 if (result == null) {  
 log.warning("server/handleGet: Graph not found");  
 status = HttpStatus.NOT\_FOUND;  
 } else {  
 log.info("server/handleGet: Returning graph");  
 status = HttpStatus.OK;  
 }  
  
 request.response  
 ..statusCode = status  
 ..headers.contentType = new ContentType("application", "json", charset: "utf-8")  
 ..write(JSON.encode(result))  
 ..close();  
}  
  
Future<Null> handleLayout(HttpRequest request) async {  
 String jsonString = await request.transform(UTF8.decoder).join();  
 log.info("Got layout request for: $jsonString");  
  
 ProcessResult results = await Process.run('python', ['MachineLearning/QLearning.py', jsonString]);  
 String output = results.stdout;  
 String err = results.stderr;  
 log.config("Output: $output");  
 log.config("Error: ${err.length > 0} $err");  
  
 request.response  
 ..statusCode = HttpStatus.OK  
 ..headers.contentType = new ContentType("application", "json", charset: "utf-8")  
 ..write(output)  
 ..close();  
}

## graphBase/graph\_schema.md

#### JSON Format  
```  
{   
 \_id: GRAPH\_NAME,  
 fitness: GRAPH\_FITNESS,  
 numNodes: NUMBER\_OF\_NODES,  
 numConnections: NUMBER\_OF\_NODES,  
 nodes: [  
 {  
 x: X\_POS,  
 y: Y\_POS,  
 height: HEIGHT,  
 width: WIDTH  
 },  
 {  
 x: X\_POS,  
 y: Y\_POS,  
 height: HEIGHT,  
 width: WIDTH  
 }  
 ],  
 connections: [  
 {  
 a\_node: A\_NODE,  
 b\_node: B\_NODE,  
 height: HEIGHT,  
 width: WIDTH  
 },  
 {  
 a\_node: A\_NODE,  
 b\_node: B\_NODE,  
 height: HEIGHT,  
 width: WIDTH  
 }  
 ],  
}  
```  
  
  
#### Notes:  
- Fitness is user defined as  
 - 0: Unusable  
 - 1: Could be worse  
 - 2: Could be better  
 - 3: Perfect  
- A\_Node is the originating node, ie, if there exists a connecting arrow between A\_Node and B\_Node, and the arrow is unidirectional, the point is touching B\_Node.  
- Numbers are pixels  
- Orientation of Graph Coordinates  
  
```  
 X=0 --> X++  
Y=0 ---------  
 | | |  
 | | |  
 V | |  
Y++ ---------  
```

## lib/src/components/chart\_component.dart

part of wFlow;  
  
var ChartComponent = react.registerComponent(() => new \_ChartComponent());  
  
class \_ChartComponent extends w\_flux.FluxComponent<ChartActions, ChartStore> {  
 dnd.Dropzone dropzone;  
 int currentDragNode = -1;  
  
 @override  
 void componentDidMount() {  
 super.componentDidMount();  
 setupDragging();  
 calcCanvasSize();  
 }  
  
 @override  
 void componentWillUnmount() {  
 super.componentWillUnmount();  
 dropzone.destroy();  
 }  
  
 render() {  
 List nodeDivs = new List();  
 List connectionDivs = new List();  
  
 store.nodes.forEach((id, node) {  
 nodeDivs.add(NodeComponent({  
 'store': store,  
 'actions': actions,  
 'id': id,  
 'node': node,  
 'key': 'wNode$id',  
 }));  
 });  
  
 store.connections.forEach((id, connection) {  
 connectionDivs.add(ConnectionComponent({  
 'store': store,  
 'actions': actions,  
 'id': id,  
 'connection': connection,  
 'key': 'wConn$id',  
 }));  
 });  
  
 return react.div({  
 'className': 'wFlowChart full-page',  
 'style': {'padding': '10px'}  
 }, [  
 \_buildModal(),  
 nodeDivs,  
 connectionDivs,  
 (store.contextMenuNode == null)  
 ? null  
 : ContextMenuComponent({  
 'store': store,  
 'actions': actions,  
 'key': 'wContextMenu${store.contextMenuNode.id}',  
 'node': store.contextMenuNode,  
 })  
 ]);  
 }  
  
 \_buildModal() {  
 List modals = [];  
  
 if (store.savingGraph) {  
 modals.add(ModalComponent({  
 'store': store,  
 'actions': actions,  
 'key': 'saveModel',  
 'type': ModalType.SAVE,  
 }));  
 }  
 if (store.loadingGraph) {  
 modals.add(ModalComponent({  
 'store': store,  
 'actions': actions,  
 'key': 'loadModel',  
 'type': ModalType.LOAD,  
 }));  
 }  
 if (store.errorMessage != null) {  
 modals.add(ModalComponent({  
 'store': store,  
 'actions': actions,  
 'key': 'loadModel',  
 'message': store.errorMessage,  
 'type': ModalType.ERROR,  
 }));  
 }  
  
 return modals;  
 }  
  
 void setupDragging() {  
 dropzone = new dnd.Dropzone(react\_dom.findDOMNode(this));  
 }  
  
 void calcCanvasSize() {  
 Element me = react\_dom.findDOMNode(this);  
  
 int left = me.clientLeft + 50;  
 int top = me.clientTop + 100;  
 int right = left + me.clientWidth - 2 \* halfNodeWidth - 50;  
 int bottom = top + me.clientHeight - 2 \* halfNodeWidth - 100;  
 actions.setCanvasSize(new CanvasSize(left, top, right, bottom));  
 }  
}

## lib/src/components/connection\_component.dart

part of wFlow;  
  
var ConnectionComponent = react.registerComponent(() => new \_ConnectionComponent());  
  
// TODO: This is required to make the connection render from the center of the node, instead of the upper left corner.  
// This will need to change if we change to dynamic node sizes or change the static size.  
const int halfNodeWidth = 50;  
  
class \_ConnectionComponent extends w\_flux.FluxComponent<ChartActions, ChartStore> {  
 int get \_id => props['id'];  
 Connection get \_connection => props['connection'];  
 Node get \_startNode => store.nodes[\_connection.start];  
 Node get \_endNode => store.nodes[\_connection.end];  
  
 render() {  
 if (\_startNode == null || \_endNode == null) {  
 store.log.warning("Error: Tried to render a connection between nodes that no longer exist :(");  
 return react.div({});  
 }  
 int x1 = \_startNode.x;  
 int y1 = \_startNode.y;  
 int x2 = \_endNode.x;  
 int y2 = \_endNode.y;  
  
 double length = sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));  
 double transform = atan2(y2 - y1, x2 - x1) \* 180 / PI;  
  
 return react.div({  
 'className': 'wConnection' + (store.selectedConnection == \_id ? ' selected' : ''),  
 'id': "wConnection${\_id}",  
 'style': {  
 'position': 'absolute',  
 'transform': 'rotate(${transform}deg)',  
 'left': x1 + halfNodeWidth,  
 'top': y1 + halfNodeWidth,  
 'width': length  
 },  
 'onClick': \_onClick,  
 'width': length,  
 });  
 }  
  
 \_onClick(\_) {  
 actions.selectConnection(\_id);  
 }  
}

## lib/src/components/context\_menu\_components.dart

part of wFlow;  
  
var ContextMenuComponent = react.registerComponent(() => new \_ContextMenuComponent());  
  
class \_ContextMenuComponent extends w\_flux.FluxComponent<ChartActions, ChartStore> {  
 Node get node => props['node'];  
  
 // Adjust by half node width to position menu around center of node.  
 int get x => node.x + halfNodeWidth;  
 int get y => node.y + halfNodeWidth;  
  
 String nodeContent;  
 String nodeColor;  
  
 void componentWillMount() {  
 super.componentWillMount();  
  
 nodeContent = node.content;  
 nodeColor = node.color;  
 }  
  
 render() {  
 var contentForm =  
 react.div({'key': 'contentwrapper'}, react.textarea({'defaultValue': nodeContent, 'onChange': \_updateContent}));  
 var colorForm =  
 react.div({'key': 'colorwrapper'}, react.textarea({'defaultValue': nodeColor, 'onChange': \_updateColor}));  
  
 var cancelButton = react.button({  
 'onClick': \_closeMenu,  
 'key': 'cancel',  
 'style': {'padding': 10},  
 }, 'Cancel');  
 var submitButton = react.button({  
 'onClick': \_submitEdit,  
 'key': 'submit',  
 'style': {'padding': 10},  
 }, 'Confirm');  
  
 return react.div({  
 'className': 'wContextMenu',  
 'id': "wContextMenu",  
 'style': {'left': x, 'top': y},  
 }, [  
 react.div({'key': 'fields'}, [contentForm, colorForm]),  
 react.div({'key': 'controls'}, [cancelButton, submitButton])  
 ]);  
 }  
  
 void \_closeMenu(\_) {  
 actions.setContextMenu(null);  
 }  
  
 void \_submitEdit(\_) {  
 actions.submitContextMenuEdit(new NodeEdit(node.id, nodeContent, nodeColor));  
 \_closeMenu(\_);  
 }  
  
 void \_updateContent(event) {  
 nodeContent = event.target.value;  
 }  
  
 void \_updateColor(event) {  
 nodeColor = event.target.value;  
 }  
}

## lib/src/components/controls\_component.dart

part of wFlow;  
  
var ControlsComponent = react.registerComponent(() => new \_ControlsComponent());  
  
class \_ControlsComponent extends w\_flux.FluxComponent<ChartActions, ChartStore> {  
 render() {  
 return react.div({  
 'className': 'wFlowControls',  
 }, [  
 react.h3({  
 'style': {'textWeight': 'bold'},  
 'key': 'controlsHeader',  
 }, 'Controls'),  
 react.div(  
 {  
 'className': 'nodeControls',  
 'key': 'modGraphHeader',  
 },  
 react.h5({  
 'style': {'textWeight': 'bold'},  
 }, 'Modify Graph')),  
 react.div({'key': 'addNodeButton'}, react.button({'onClick': addNode}, 'Add Node')),  
 react.div({'key': 'deleteNodeButton'}, react.button({'onClick': deleteNode}, 'Delete Node')),  
 react.div({'key': 'addConnectionButton'}, react.button({'onClick': addConnection}, 'Add Connection')),  
 react.div({'key': 'deleteConnectionButton'}, react.button({'onClick': deleteConnection}, 'Delete Connection')),  
 react.div(  
 {'className': 'nodeControls', 'key': 'graphCrudHeader'},  
 react.h5({  
 'style': {'textWeight': 'bold'},  
 }, 'Graph CRUD')),  
 react.div({'key': 'newGraphButton'}, react.button({'onClick': newGraph}, 'New Graph')),  
 react.div({'key': 'saveGraphButton'}, react.button({'onClick': saveGraph}, 'Save Graph')),  
 react.div({'key': 'loadGraphButton'}, react.button({'onClick': loadGraph}, 'Load Graph')),  
 react.div({'key': 'layoutGraphButton'}, react.button({'onClick': layoutGraph}, 'Layout Graph')),  
 ]);  
 }  
  
 void addNode(\_) {  
 actions.addNode('I am a new node!');  
 }  
  
 void deleteNode(\_) {  
 actions.deleteNode(store.selectedNode);  
 }  
  
 void addConnection(\_) {  
 actions.addConnection();  
 }  
  
 void deleteConnection(\_) {  
 actions.deleteConnection(store.selectedConnection);  
 }  
  
 void newGraph(\_) {  
 actions.newGraph();  
 }  
  
 void saveGraph(\_) {  
 actions.isSavingGraph(true);  
 }  
  
 void loadGraph(\_) {  
 actions.isLoadingGraph(true);  
 }  
  
 Future<Null> layoutGraph(\_) async {  
 actions.layoutGraph();  
 }  
}

## lib/src/components/modal\_component.dart

part of wFlow;  
  
var ModalComponent = react.registerComponent(() => new \_ModalComponent());  
  
enum ModalType { SAVE, LOAD, ERROR }  
  
class \_ModalComponent extends w\_flux.FluxComponent<ChartActions, ChartStore> {  
 // Form values:  
 String \_name;  
 int \_fitness =  
 2; // 2 is the default quality - change which radio input below has "defaultChecked" and here to change.  
  
 // Modal versions:  
 String \_modalTitle;  
 String \_message;  
 String \_confirmText;  
 String \_cancelText;  
 Function \_confirmAction;  
 Function \_cancelAction;  
  
 @override  
 void componentWillMount() {  
 super.componentWillMount();  
 \_name = store.title;  
  
 switch (props["type"]) {  
 case ModalType.SAVE:  
 \_modalTitle = "Save Graph";  
 \_confirmText = "Save";  
 \_cancelText = "Cancel";  
 \_confirmAction = (\_) {  
 actions.saveGraph(new GraphInfo(\_name, \_fitness));  
 actions.isSavingGraph(false);  
 };  
 \_cancelAction = (\_) {  
 actions.isSavingGraph(false);  
 };  
 break;  
 case ModalType.LOAD:  
 \_modalTitle = "Load Graph";  
 \_confirmText = "Load";  
 \_cancelText = "Cancel";  
 \_confirmAction = (\_) {  
 actions.loadGraph(\_name);  
 actions.isLoadingGraph(false);  
 };  
 \_cancelAction = (\_) {  
 actions.isLoadingGraph(false);  
 };  
 break;  
 case ModalType.ERROR:  
 \_modalTitle = "Error";  
 \_message = props["message"];  
 \_cancelText = "Okay";  
 \_cancelAction = (\_) {  
 actions.showError(null);  
 };  
 break;  
 }  
 }  
  
 render() {  
 return react.div(  
 {  
 'className': 'wModalOverlay',  
 'key': 'overlay',  
 },  
 \_modalBody(),  
 );  
 }  
  
 \_modalBody() {  
 return react.div({  
 'className': 'wModalBox',  
 }, [  
 \_title(),  
 \_content(),  
 react.div({  
 'className': 'wModalButtons',  
 'key': 'buttons'  
 }, [  
 \_confirmButton(),  
 \_cancelButton(),  
 ]),  
 ]);  
 }  
  
 \_title() {  
 return react.div({  
 'className': 'wModalTitle',  
 'key': 'title',  
 }, react.h3({'key': 'titleText'}, [\_modalTitle]));  
 }  
  
 \_content() {  
 switch (props["type"]) {  
 case ModalType.SAVE:  
 return \_saveContent();  
 case ModalType.LOAD:  
 return \_loadContent();  
 case ModalType.ERROR:  
 return \_errorContent();  
 }  
 }  
  
 \_confirmButton() {  
 if (\_confirmText == null) return null;  
 return react.button(  
 {'className': 'wModalConfirmButton wModalButton', 'key': 'confirm', 'onClick': \_confirmAction}, \_confirmText);  
 }  
  
 \_cancelButton() {  
 if (\_cancelText == null) return null;  
 return react.button(  
 {'className': 'wModalCancelButton wModalButton', 'key': 'cancel', 'onClick': \_cancelAction}, \_cancelText);  
 }  
  
 // Types of modal:  
  
 \_saveContent() {  
 return react.div({  
 'key': 'saveContent',  
 'className': 'wModalSave',  
 }, [  
 react.div({  
 'key': 'titleInput',  
 'className': 'wModalInput',  
 }, [  
 react.span({  
 'key': 'description',  
 }, "Graph Name: "),  
 react.input({  
 'type': 'text',  
 'key': 'input',  
 'onChange': ((var event) => \_name = event.target.value),  
 'defaultValue': \_name,  
 }),  
 ]),  
 react.div(  
 {  
 'key': 'ratingInput',  
 'className': 'wModalInput',  
 },  
 react.span({'key': 'label'}, "Quality Rating: "),  
 \_qualityForm(),  
 ),  
 ]);  
 }  
  
 \_qualityForm() {  
 return react.form({}, [  
 react.label({  
 'className': 'radio-inline',  
 'key': 'radio0',  
 }, [  
 react.input({  
 'type': 'radio',  
 'name': 'quality',  
 'key': 'radio0',  
 'onClick': ((\_) => \_fitness = 0),  
 'value': 0,  
 }),  
 "Worst",  
 ]),  
 react.label({  
 'className': 'radio-inline',  
 'key': 'radio1',  
 }, [  
 react.input({  
 'type': 'radio',  
 'name': 'quality',  
 'key': 'radio1',  
 'onClick': ((\_) => \_fitness = 1),  
 'value': 1,  
 }),  
 "Poor",  
 ]),  
 react.label({  
 'className': 'radio-inline',  
 'key': 'radio2',  
 }, [  
 react.input({  
 'type': 'radio',  
 'name': 'quality',  
 'key': 'radio2',  
 'defaultChecked': true, // selected by default  
 'onClick': ((\_) => \_fitness = 2),  
 'value': 2,  
 }),  
 "Good",  
 ]),  
 react.label({  
 'className': 'radio-inline',  
 'key': 'radio3',  
 }, [  
 react.input({  
 'type': 'radio',  
 'name': 'quality',  
 'key': 'radio3',  
 'onClick': ((\_) => \_fitness = 3),  
 'value': 3,  
 }),  
 "Best",  
 ]),  
 ]);  
 }  
  
 \_loadContent() {  
 return react.div({  
 'key': 'loadContent',  
 'className': 'wModalLoad',  
 }, [  
 react.div({  
 'key': 'titleInput',  
 'className': 'wModalInput',  
 }, [  
 react.span({  
 'key': 'description',  
 }, "Graph Name: "),  
 react.input({  
 'type': 'text',  
 'key': 'input',  
 'onChange': ((var event) => \_name = event.target.value),  
 'defaultValue': \_name,  
 }),  
 ]),  
 ]);  
 }  
  
 \_errorContent() {  
 return react.div({  
 'className': 'wModalError',  
 'key': 'errorContent',  
 }, \_message);  
 }  
}

## lib/src/components/node\_component.dart

part of wFlow;  
  
var NodeComponent = react.registerComponent(() => new \_NodeComponent());  
  
class \_NodeComponent extends w\_flux.FluxComponent<ChartActions, ChartStore> {  
 int get \_id => props['id'];  
 Node get \_node => props['node'];  
  
 dnd.Draggable draggable;  
 List<StreamSubscription> \_dndSubs = [];  
 w\_flux.ActionSubscription cancelSub;  
  
 @override  
 bool shouldComponentUpdate(Map nextProps, Map nextState) {  
 // Don't update during drags  
 return \_node == null || \_node.id != store.draggingNode;  
 }  
  
 @override  
 void componentDidUpdate(Map prevprops, Map prevState) {  
 // TODO (977): for unknown reasons, after attempting to drag outside of the  
 // chart area, sometimes the rendered node gets stuck with outdated styles  
 // that isn't fixed by subsequent rerenders. This method detects those  
 // cases, until we can fix them. (save and reload is the current workaround)  
 Element element = react\_dom.findDOMNode(this);  
  
 int x = \_parseLocation(element.style.left);  
 int y = \_parseLocation(element.style.top);  
  
 if (\_node.x != x || \_node.y != y) {  
 store.log.info("Rendered node mismatch with node state. (See 977)");  
 }  
 }  
  
 @override  
 void componentDidMount() {  
 cancelSub = actions.cancelDrag.listen(\_handleDragCancel);  
 setupDragging();  
 }  
  
 void componentWillUnmount() {  
 cancelSub.cancel();  
 teardownDragging();  
 super.componentWillUnmount();  
 }  
  
 render() {  
 return react.div({  
 'className': 'wNode draggable' + (store.selectedNode == \_id ? ' selected' : ''),  
 'style': {'backgroundColor': \_node.color, 'left': '${\_node.x}px', 'top': '${\_node.y}px', 'position': 'absolute'},  
 'id': "wNode${\_node.id}",  
 'onClick': \_onClick,  
 'onContextMenu': \_onRightClick,  
 }, [  
 \_node.content  
 ]);  
 }  
  
 void \_onClick(\_) {  
 actions.selectNode(\_id);  
 }  
  
 void \_onRightClick(e) {  
 e.preventDefault();  
 actions.setContextMenu(\_node);  
 }  
  
 void setupDragging() {  
 draggable = new dnd.Draggable(react\_dom.findDOMNode(this), avatarHandler: new dnd.AvatarHandler.original());  
 \_dndSubs.add(draggable.onDragStart.listen(\_handleDragBegin));  
 \_dndSubs.add(draggable.onDragEnd.listen(\_handleDragEnd));  
 \_dndSubs.add(draggable.onDrag.listen(\_handleDragIntermediate));  
 }  
  
 void teardownDragging() {  
 \_dndSubs.forEach((sub) {  
 sub.cancel();  
 });  
 draggable.destroy();  
 }  
  
 void \_handleDragBegin(dnd.DraggableEvent event) {  
 actions.setNodeDragging(\_id);  
 }  
  
 void \_handleDragIntermediate(dnd.DraggableEvent event) {  
 if (event.cancelled) return;  
 // Note: we cannot rely on 'event.postion' (see \_handleDragEnd)  
 Element element = react\_dom.findDOMNode(this);  
  
 int x = \_parseLocation(element.style.left);  
 int y = \_parseLocation(element.style.top);  
  
 // If the drag has begun we need to consider the css transform to find the  
 // current location.  
 String transform = element.style.transform;  
 if (transform.length > 0) {  
 transform = transform.replaceFirst("translate3d", "");  
 transform = transform.substring(1, transform.length - 1).replaceAll(" ", "");  
 List coords = transform.split(",");  
 x += \_parseLocation(coords[0]);  
 y += \_parseLocation(coords[1]);  
 }  
  
 actions.moveNode(new NodeCoord(\_id, x, y));  
 }  
  
 void \_handleDragEnd(dnd.DraggableEvent event) {  
 if (event.cancelled) return;  
  
 // Note: 'event.position' is the location of the user's click on the element, not on the element itself.  
 // Instead, query for the actual react node and use its styling to identify the location  
 Element element = react\_dom.findDOMNode(this);  
  
 int x = \_parseLocation(element.style.left);  
 int y = \_parseLocation(element.style.top);  
  
 actions.moveNode(new NodeCoord(\_id, x, y));  
 actions.setNodeDragging(-1);  
 }  
  
 void \_handleDragCancel(\_) {  
 draggable.abort();  
 actions.setNodeDragging(-1);  
 }  
  
 int \_parseLocation(String input) {  
 return int.parse(input.substring(0, input.length - 2)); // Chop off the 'px' then convert to int  
 }  
}

## lib/src/components/title\_component.dart

part of wFlow;  
  
var TitleComponent = react.registerComponent(() => new \_TitleComponent());  
  
class \_TitleComponent extends w\_flux.FluxComponent<ChartActions, ChartStore> {  
 render() {  
 return react.h1({  
 'className': 'wFlowChart',  
 }, store.title);  
 }  
}

## lib/src/models/payload.dart

part of wFlow;  
  
// Action Payloads  
  
class NodeCoord {  
 int nodeId;  
 int x;  
 int y;  
  
 NodeCoord(this.nodeId, this.x, this.y);  
  
 String toString() {  
 return "Node $nodeId: ($x, $y)";  
 }  
}  
  
class NodeEdit {  
 int nodeId;  
 String content;  
 String color;  
  
 NodeEdit(this.nodeId, this.content, this.color);  
}  
  
class CanvasSize {  
 int left, top, right, bottom;  
  
 CanvasSize(this.left, this.top, this.right, this.bottom);  
  
 String toString() {  
 return "left: $left top: $top right: $right bottom: $bottom";  
 }  
}  
  
class GraphInfo {  
 String title;  
 int fitness;  
  
 GraphInfo(this.title, this.fitness);  
}  
  
// Graph Structures  
  
class Node {  
 String \_content;  
 String get content => \_content;  
 String \_color;  
 String get color => \_color;  
 int \_id;  
 int get id => \_id;  
  
 int \_x;  
 int get x => \_x;  
 int \_y;  
 int get y => \_y;  
  
 Node(this.\_id, this.\_content, {String color: "#c4ffff", int x: 50, int y: 100}) {  
 \_color = color;  
 \_x = x;  
 \_y = y;  
 }  
  
 Node.fromJson(Map json) {  
 \_id = json["id"];  
 \_content = json["content"];  
 \_color = json["color"];  
 \_x = json["x"];  
 \_y = json["y"];  
 }  
  
 void setContent(String content) {  
 \_content = content;  
 }  
  
 void setLocation(int x, int y) {  
 \_x = x;  
 \_y = y;  
 }  
  
 void setColor(String hexColor) {  
 \_color = hexColor;  
 }  
  
 Map toJson() {  
 return {  
 "id": id,  
 "x": x,  
 "y": y,  
 "color": color,  
 "content": content,  
 };  
 }  
  
 String toString() {  
 return "Node $id: \n\t content: $content \n\t color: $color \n\t coord: ($x, $y)\n";  
 }  
  
 bool operator ==(other) {  
 return (other is Node &&  
 other.content == content &&  
 other.color == color &&  
 other.id == id &&  
 other.x == x &&  
 other.y == y);  
 }  
}  
  
class Connection {  
 int \_id;  
 int get id => \_id;  
  
 int \_start;  
 int get start => \_start;  
 int \_end;  
 int get end => \_end;  
  
 Connection(this.\_id, this.\_start, this.\_end);  
  
 Connection.fromJson(Map json) {  
 \_id = json["id"];  
 \_start = json["start"];  
 \_end = json["end"];  
 }  
  
 Map toJson() {  
 return {  
 "id": id,  
 "start": start,  
 "end": end,  
 };  
 }  
  
 String toString() {  
 return "Connection $id: \n\t Start NodeId: $start \n\t End NodeId: $end \n";  
 }  
  
 bool operator ==(other) {  
 return (other is Connection && other.id == id && other.start == start && other.end == end);  
 }  
}

## lib/src/stores/chart\_store.dart

part of wFlow;  
  
class ChartStore extends w\_flux.Store {  
 ChartActions \_actions;  
 ChartEvents \_events;  
 ExternalHandler \_externalHandler;  
 Logger log;  
  
 w\_module.DispatchKey \_chartModuleDispatchKey;  
  
 bool \_readOnly;  
 bool get readOnly => \_readOnly;  
 bool \_dragEnabled;  
 bool get dragEnabled => \_dragEnabled;  
  
 String \_title;  
 String get title => \_title;  
 int \_fitness = 0;  
 int get fitness => \_fitness;  
  
 // Graph CRUD State:  
 bool \_savingGraph = false;  
 bool get savingGraph => \_savingGraph;  
 bool \_loadingGraph = false;  
 bool get loadingGraph => \_loadingGraph;  
 String \_errorMessage;  
 String get errorMessage => \_errorMessage;  
  
 Node \_contextMenuNode;  
 Node get contextMenuNode => \_contextMenuNode;  
  
 bool \_addingConnection = false;  
 bool get addingConnection => \_addingConnection;  
 // Start node is the first node clicked during a connection add  
 int \_startNode = -1;  
 // End node is the second node clicked during a connection addf  
 int \_endNode = -1;  
  
 Map<int, Node> \_nodes = {};  
 Map<int, Node> get nodes => \_nodes;  
 int \_maxNodeId = 0;  
  
 Map<int, Connection> \_connections = {};  
 Map<int, Connection> get connections => \_connections;  
 int \_maxConnId = 0;  
  
 // id of the currently selected node, or -1 if no node is selected  
 int \_selectedNode = -1;  
 int get selectedNode => \_selectedNode;  
  
 // id of the currently selected connection, or -1 if no connection is selected  
 int \_selectedConnection = -1;  
 int get selectedConnection => \_selectedConnection;  
  
 // id of the node currently being dragged, or -1 if no node is dragged  
 int \_draggingNode = -1;  
 int get draggingNode => \_draggingNode;  
  
 CanvasSize \_canvas;  
  
 ChartStore(this.log, this.\_actions, this.\_events, this.\_chartModuleDispatchKey, this.\_externalHandler,  
 {CanvasSize canvas}) {  
 \_readOnly = false;  
 \_dragEnabled = true;  
 \_title = "Example Chart";  
 \_canvas = canvas;  
  
 // Subscribe to actions  
 manageActionSubscription(\_actions.addNode.listen(\_addNode));  
 manageActionSubscription(\_actions.deleteNode.listen(\_deleteNode));  
 manageActionSubscription(\_actions.selectNode.listen(\_selectNode));  
 manageActionSubscription(\_actions.moveNode.listen(\_moveNode));  
 manageActionSubscription(\_actions.setNodeDragging.listen(\_setNodeDragging));  
 manageActionSubscription(\_actions.setContextMenu.listen(\_setContextMenu));  
 manageActionSubscription(\_actions.submitContextMenuEdit.listen(\_handleContextMenuEdit));  
 manageActionSubscription(\_actions.addConnection.listen(\_startAddConnection));  
 manageActionSubscription(\_actions.deleteConnection.listen(\_deleteConnection));  
 manageActionSubscription(\_actions.selectConnection.listen(\_selectConnection));  
 manageActionSubscription(\_actions.setDragEnabled.listen(\_setDragEnabled));  
 manageActionSubscription(\_actions.setReadOnlyMode.listen(\_setReadOnlyMode));  
 manageActionSubscription(\_actions.isSavingGraph.listen(\_updateSavingGraph));  
 manageActionSubscription(\_actions.saveGraph.listen(\_saveGraph));  
 manageActionSubscription(\_actions.newGraph.listen(\_newGraph));  
 manageActionSubscription(\_actions.isLoadingGraph.listen(\_updateLoadingGraph));  
 manageActionSubscription(\_actions.loadGraph.listen(\_loadGraph));  
 manageActionSubscription(\_actions.layoutGraph.listen(\_layoutGraph));  
 manageActionSubscription(\_actions.setCanvasSize.listen(\_setCanvasSize));  
 manageActionSubscription(\_actions.showError.listen(\_showError));  
 }  
  
 void \_showError(String error) {  
 \_errorMessage = error;  
 trigger();  
 }  
  
 void \_setCanvasSize(CanvasSize size) {  
 \_canvas = size;  
 }  
  
 bool \_isNodeWithinCanvas(NodeCoord coord) {  
 return (coord.x >= \_canvas.left && coord.x <= \_canvas.right && coord.y >= \_canvas.top && coord.y <= \_canvas.bottom);  
 }  
  
 void \_addNode(String content) {  
 int id = \_maxNodeId++;  
 content = "Node $id"; // TODO: stop ignoring content  
  
 nodes[id] = new Node(id, content);  
 trigger();  
 }  
  
 void \_deleteNode(int nodeId) {  
 if (!nodes.containsKey(nodeId)) {  
 log.warning("Tried to delete non-existent node! $nodeId");  
 return;  
 }  
 if (\_selectedNode == nodeId) {  
 \_selectedNode = -1;  
 }  
  
 nodes.remove(nodeId);  
 \_removeAllConnectionsToNode(nodeId);  
  
 trigger();  
 }  
  
 void \_removeAllConnectionsToNode(int nodeId) {  
 List<int> toRemove = new List();  
  
 connections.forEach((id, conn) {  
 if (conn.start == nodeId || conn.end == nodeId) {  
 toRemove.add(id);  
 }  
 });  
  
 toRemove.forEach((id) => connections.remove(id));  
 }  
  
 void \_selectNode(int nodeId) {  
 if (\_addingConnection) {  
 return \_setConnectionEndpoint(nodeId);  
 }  
  
 if (nodeId == selectedNode) {  
 \_selectedNode = -1;  
 } else {  
 \_selectedNode = nodeId;  
 }  
 trigger();  
 }  
  
 void \_moveNode(NodeCoord coord) {  
 Node node = nodes[coord.nodeId];  
 if (node == null) {  
 log.warning("Tried to move a non-existent node! $coord");  
 return;  
 }  
  
 if (!\_isNodeWithinCanvas(coord)) {  
 log.warning("Tried to move node outside of canvas! $coord \n canvas: $\_canvas");  
 if (draggingNode != -1) \_actions.cancelDrag();  
 return;  
 }  
  
 node.setLocation(coord.x, coord.y);  
 trigger();  
 }  
  
 void \_setNodeDragging(int nodeId) {  
 \_draggingNode = nodeId;  
 trigger();  
 }  
  
 void \_setContextMenu(Node contextNode) {  
 if (\_contextMenuNode == contextNode) {  
 \_contextMenuNode = null;  
 } else {  
 \_contextMenuNode = contextNode;  
 }  
 trigger();  
 }  
  
 void \_handleContextMenuEdit(NodeEdit edit) {  
 Node node = nodes[edit.nodeId];  
 if (node == null) {  
 log.warning("Tried to edit a non-existent node! $edit");  
 return;  
 }  
 node.setContent(edit.content);  
 node.setColor(edit.color);  
  
 trigger();  
 }  
  
 void \_startAddConnection(\_) {  
 \_addingConnection = true;  
 }  
  
 void \_setConnectionEndpoint(int nodeId) {  
 if (\_startNode == -1) {  
 \_startNode = nodeId;  
 return;  
 }  
 \_endNode = nodeId;  
 \_finishConnectionAdd();  
 }  
  
 void \_finishConnectionAdd() {  
 int id = \_maxConnId++;  
 connections[id] = new Connection(id, \_startNode, \_endNode);  
  
 \_startNode = \_endNode = -1;  
 \_addingConnection = false;  
  
 trigger();  
 }  
  
 void \_deleteConnection(int connId) {  
 if (!connections.containsKey(connId)) {  
 log.warning("Tried to delete non-existent connection! $connId");  
 return;  
 }  
 if (\_selectedConnection == connId) {  
 \_selectedConnection = -1;  
 }  
  
 connections.remove(connId);  
 trigger();  
 }  
  
 void \_selectConnection(int connId) {  
 if (connId == \_selectedConnection) {  
 \_selectedConnection = -1;  
 } else {  
 \_selectedConnection = connId;  
 }  
 trigger();  
 }  
  
// Remove all visible nodes and connections  
 void \_newGraph(\_) {  
 \_clearGraph();  
 trigger();  
 }  
  
 void \_updateSavingGraph(bool isSaving) {  
 \_savingGraph = isSaving;  
 trigger();  
 }  
  
 void \_updateLoadingGraph(bool isLoading) {  
 \_loadingGraph = isLoading;  
 trigger();  
 }  
  
 // Initiate a contact database event, with the graph to save, in json format.  
 void \_saveGraph(GraphInfo graphInfo) {  
 \_title = graphInfo.title;  
 \_fitness = graphInfo.fitness;  
  
 Map json = \_graphToJson(graphInfo.title, graphInfo.fitness);  
 if (json == null || json.length < 1) {  
 log.shout("Unable to save ${graphInfo.title}: error while parsing information to JSON");  
 return;  
 }  
 \_externalHandler.saveGraphCall(json);  
 trigger();  
 }  
  
 // Initiate a contact database event, with the graph to load as name.  
 Future<Null> \_loadGraph(String graphName) async {  
 String loadedGraph = await \_externalHandler.loadGraphCall(graphName);  
  
 // Graph did not exist  
 if (loadedGraph == null || loadedGraph.length < 1) {  
 log.warning("Graph $graphName did not exist");  
 return;  
 }  
  
 Map parsedGraph = JSON.decode(loadedGraph);  
  
 // clear any current nodes/connections  
 \_clearGraph();  
  
 \_loadGraphFromJSON(parsedGraph);  
  
 trigger();  
 return;  
 }  
  
 Future<Null> \_layoutGraph(\_) async {  
 String response = await \_externalHandler.layoutGraphCall(\_graphToJson(title, fitness));  
  
 if (response == null || response.length < 1) {  
 log.warning("Unable to get graph layout!");  
 return;  
 }  
  
 Map parsedGraph = JSON.decode(response);  
  
 // clear any current nodes/connetions  
 \_clearGraph();  
 \_loadGraphFromJSON(parsedGraph);  
  
 trigger();  
 }  
  
 // reset state of graph to no nodes/connections  
 void \_clearGraph() {  
 nodes.clear();  
 connections.clear();  
 \_maxNodeId = 0;  
 \_maxConnId = 0;  
 }  
  
 void \_setDragEnabled(bool option) {  
 log.info("Setting dragEnabled to $option");  
 \_dragEnabled = option;  
 }  
  
 void \_setReadOnlyMode(bool option) {  
 log.info("Setting readOnly to $option");  
 \_readOnly = option;  
 }  
  
 Map \_graphToJson(String name, int fitness) {  
 Map jsonGraph = {  
 '\_id': name,  
 'fitness': fitness,  
 };  
  
 List jsonNodes = new List();  
 nodes.forEach((\_, Node node) {  
 jsonNodes.add(node.toJson());  
 });  
  
 List jsonConns = new List();  
 connections.forEach((\_, Connection conn) {  
 jsonConns.add(conn.toJson());  
 });  
  
 jsonGraph["nodes"] = jsonNodes;  
 jsonGraph["connections"] = jsonConns;  
 return jsonGraph;  
 }  
  
 void \_loadGraphFromJSON(Map json) {  
 \_title = json["\_id"];  
 \_fitness = json["fitness"];  
  
 json["nodes"].forEach((Map jsonNode) {  
 Node node = new Node.fromJson(jsonNode);  
 if (nodes.containsKey(node.id)) {  
 log.shout(  
 "Chart trying to load node with id ${node.id} - node with given id already exists. \n Existing node: ${nodes[node.id]}\n Loading node: ${node}");  
 log.warning("Overriding existing node.");  
 }  
 nodes[node.id] = node;  
 \_maxNodeId = max(\_maxNodeId, node.id);  
 });  
  
 json["connections"].forEach((Map jsonConn) {  
 Connection conn = new Connection.fromJson(jsonConn);  
 if (connections.containsKey(conn.id)) {  
 log.shout(  
 "Chart trying to load connection with id ${conn.id} - connection with given id already exists. \n Existing connection: ${connections[conn.id]}\n Loading connection: ${conn}");  
 log.warning("Overriding existing connection.");  
 }  
 connections[conn.id] = conn;  
 \_maxConnId = max(\_maxConnId, conn.id);  
 });  
 }  
  
 // Debugging methods:  
 void \_printStatus() {  
 log.info("Current Nodes:");  
 nodes.forEach((key, value) {  
 log.info(value);  
 });  
 log.info("Current Connections:");  
 connections.forEach((key, value) {  
 log.info(value);  
 });  
 }  
}

## lib/src/actions.dart

part of wFlow;  
  
// Actions are dispatched to drive state changes.  
// See https://github.com/Workiva/w\_flux#action for details  
class ChartActions {  
 final w\_flux.Action<String> addNode = new w\_flux.Action<String>();  
 final w\_flux.Action<int> deleteNode = new w\_flux.Action<int>();  
 final w\_flux.Action<int> selectNode = new w\_flux.Action<int>();  
 final w\_flux.Action<NodeCoord> moveNode = new w\_flux.Action<NodeCoord>();  
  
 final w\_flux.Action addConnection = new w\_flux.Action();  
 final w\_flux.Action<int> deleteConnection = new w\_flux.Action<int>();  
 final w\_flux.Action<int> selectConnection = new w\_flux.Action();  
  
 final w\_flux.Action<int> setNodeDragging = new w\_flux.Action<int>();  
 final w\_flux.Action cancelDrag = new w\_flux.Action();  
  
 final w\_flux.Action<CanvasSize> setCanvasSize = new w\_flux.Action<CanvasSize>();  
  
 final w\_flux.Action<Node> setContextMenu = new w\_flux.Action<Node>();  
 final w\_flux.Action<NodeEdit> submitContextMenuEdit = new w\_flux.Action<NodeEdit>();  
  
 final w\_flux.Action<bool> setDragEnabled = new w\_flux.Action<bool>();  
 final w\_flux.Action<bool> setReadOnlyMode = new w\_flux.Action<bool>();  
  
 final w\_flux.Action<bool> isSavingGraph = new w\_flux.Action<bool>();  
 final w\_flux.Action<GraphInfo> saveGraph = new w\_flux.Action<GraphInfo>();  
 final w\_flux.Action<bool> isLoadingGraph = new w\_flux.Action<bool>();  
 final w\_flux.Action<String> loadGraph = new w\_flux.Action<String>();  
  
 final w\_flux.Action newGraph = new w\_flux.Action();  
 final w\_flux.Action layoutGraph = new w\_flux.Action();  
  
 final w\_flux.Action<String> showError = new w\_flux.Action<String>();  
}

## lib/src/api.dart

part of wFlow;  
  
class ChartApi {  
 ChartActions \_actions;  
 ChartStore \_store;  
  
 ChartApi(this.\_actions, this.\_store);  
  
 // TODO: This is where we will expose an API to consumers.  
 // We may or may not need an intense API - we need to investigate how this  
 // will be consumed.  
 // NOTE: Neither of these flags currently affect anything.  
  
 // ---- Getters  
  
 bool get isReadOnly => \_store.readOnly;  
  
 bool get dragEnabled => \_store.dragEnabled;  
  
 // ---- Setters  
  
 Future enterReadOnlyMode() => \_actions.setReadOnlyMode(true);  
  
 Future exitReadOnlyMode() => \_actions.setReadOnlyMode(false);  
  
 Future enableDrag() => \_actions.setDragEnabled(true);  
  
 Future disableDrag() => \_actions.setDragEnabled(false);  
}

## lib/src/components.dart

part of wFlow;  
  
// ChartComponents is where we will expose UI components to consumers.  
class ChartComponents extends w\_module.ModuleComponents {  
 ChartActions \_actions;  
 ChartStore \_store;  
  
 ChartComponents(this.\_actions, this.\_store);  
  
 // The title of the chart module.  
 title() {  
 return TitleComponent({'actions': \_actions, 'store': \_store, 'key': 'titleComponent'});  
 }  
  
 // The controls for the chart module.  
 controls() {  
 return ControlsComponent({'actions': \_actions, 'store': \_store, 'key': 'controlsComponent'});  
 }  
  
 // The content of the chart.  
 content() {  
 return ChartComponent({'actions': \_actions, 'store': \_store, 'key': 'chartComponent'});  
 }  
}

## lib/src/external\_handler.dart

part of wFlow;  
  
// Handler for external calls, ie to database  
class ExternalHandler {  
 Logger log;  
  
 //TODO 850 change url when not using localhost  
 String url = 'http://localhost:8000';  
 BrowserClient client;  
 ChartActions \_actions;  
 ExternalHandler(this.log, this.\_actions) {  
 client = new BrowserClient();  
 }  
  
 Future<bool> saveGraphCall(Map json\_graph) async {  
 String json\_data = JSON.encode(json\_graph);  
 bool success = true;  
 try {  
 http.Response response = await client.post(url, body: json\_data);  
 log.info("saveGraphCall response code ${response.statusCode}");  
 if (response.statusCode != 200) {  
 success = false;  
 \_actions.showError("Unable to save graph. (Server status code: ${response.statusCode})");  
 }  
 } catch (e) {  
 log.info("Error in lib/src/ExternalHandler/saveGraphCall $e");  
 success = false;  
 \_actions.showError("Unable to save graph. An unexpected error occurred.");  
 }  
 client.close();  
 return success;  
 }  
  
 Future<String> loadGraphCall(String graph\_name) async {  
 String result;  
 try {  
 http.Response response = await client.get('$url/$graph\_name');  
 log.info("loadGraphCall response code ${response.statusCode}");  
 if (response.statusCode == 200) {  
 result = response.body;  
 } else {  
 \_actions.showError("Unable to load graph. (Server status code: ${response.statusCode})");  
 }  
 } catch (e) {  
 log.info("Error in lib/src/ExternalHandler/getGraphCall $e");  
 \_actions.showError("Unable to load graph. An unexpected error occurred.");  
 }  
 client.close();  
 return result;  
 }  
  
 Future<String> layoutGraphCall(Map json\_graph) async {  
 String json\_data = JSON.encode(json\_graph);  
 http.Response response;  
 try {  
 response = await client.put('$url', body: json\_data);  
 log.info("layoutCall response code ${response.statusCode}");  
 if (response.statusCode != 200) {  
 \_actions.showError("Unable to layout graph. (Server status code: ${response.statusCode})");  
 }  
 } catch (e) {  
 log.info("Error in lib/src/ExternalHandler/layoutCall $e");  
 \_actions.showError("Unable to layout graph. An unexpected error occurred.");  
 }  
 client.close();  
 return response?.body;  
 }  
}

## lib/src/module.dart

part of wFlow;  
  
class ChartModule extends w\_module.Module {  
 ChartComponents \_components;  
 ChartComponents get components => \_components;  
  
 ChartApi \_api;  
 ChartApi get api => \_api;  
  
 ChartEvents \_events;  
 ChartEvents get events => \_events;  
  
 ChartStore \_store;  
 ChartActions \_actions;  
  
 Logger log;  
  
 ChartModule() {  
 // This is the library's root logger, meaning all children  
 // loggers will inherit WFLOW's logging level. You can override level in  
 // classes for finer grain logging.  
 log = new Logger('WFLOW');  
 hierarchicalLoggingEnabled = true;  
 Logger.root.level = Level.ALL;  
 Logger.root.onRecord.listen((LogRecord rec) {  
 print('${rec.level.name}: ${rec.loggerName}: ${rec.time}: ${rec.message}');  
 });  
  
 w\_module.DispatchKey \_chartModuleDispatchKey = new w\_module.DispatchKey("wFlow");  
 \_events = new ChartEvents(\_chartModuleDispatchKey);  
 \_actions = new ChartActions();  
 \_store = new ChartStore(new Logger("WFLOW.CHARTSTORE"), \_actions, \_events, \_chartModuleDispatchKey,  
 new ExternalHandler(new Logger("WFLOW.EXTERNALHANDLER"), \_actions));  
 \_api = new ChartApi(\_actions, \_store);  
 \_components = new ChartComponents(\_actions, \_store);  
 }  
  
 Future onUnload() async {  
 \_store.dispose();  
 }  
}  
  
class ChartEvents {  
 w\_module.DispatchKey \_chartModuleDispatchKey;  
  
 ChartEvents(this.\_chartModuleDispatchKey);  
}

## lib/README.md

# wFlow Library  
  
### Client Structure  
`w\_flow.dart` exposes the wFlow library and defines which source files are included.  
  
`src/` contains all client source code.  
  
- `components/` contains the definition of all internal w\_flux UI components used in the module.  
- `models/` contains the internal payload and object models, including the node and connection classes.  
- `stores/` contains all client stores, which maintain the state of the application. Currently there is only one, `chart\_store.dart`, but as the complexity of the app increase the state may be seperated into multiple stores.  
- `actions.dart` defines all actions, which are dispatched by components to update state.  
- `api.dart` defines the API exposed to consumers, which can be used to dispatch internal actions. Currently the api is ~nonexistant~ lightweight.  
- `components.dart` defines the top level components that consumers use to display the module.  
- `external\_handler.dart` is used by the `chart\_store` to make http reqests to the server.  
- `module.dart` is the top level class of the library, which is instatiated by the consumer and exposes the API, events, and components to them.

## lib/w\_flow.dart

library wFlow;  
  
import 'dart:async';  
import 'dart:convert';  
import 'dart:math';  
import 'dart:html';  
  
import 'package:w\_module/w\_module.dart' as w\_module;  
import 'package:w\_flux/w\_flux.dart' as w\_flux;  
import 'package:react/react.dart' as react;  
import 'package:react/react\_dom.dart' as react\_dom;  
import 'package:dnd/dnd.dart' as dnd;  
import 'package:http/browser\_client.dart';  
import 'package:http/http.dart' as http;  
import 'package:logging/logging.dart';  
  
part 'src/module.dart';  
part 'src/api.dart';  
part 'src/components.dart';  
part 'src/actions.dart';  
part 'src/stores/chart\_store.dart';  
part 'src/models/payload.dart';  
part 'src/components/chart\_component.dart';  
part 'src/components/controls\_component.dart';  
part 'src/components/title\_component.dart';  
part 'src/components/node\_component.dart';  
part 'src/components/connection\_component.dart';  
part 'src/components/context\_menu\_component.dart';  
part 'src/components/modal\_component.dart';  
part 'src/external\_handler.dart';

## test/unit/components/chart\_component\_test.dart

import 'package:test/test.dart';  
  
import 'package:logging/logging.dart';  
import 'package:react/react\_test\_utils.dart' as react\_test\_utils;  
  
import 'package:wFlow/w\_flow.dart';  
  
main() {  
 group("Chart Component", () {  
 ChartStore store;  
 ChartActions actions;  
 var renderedInstance;  
  
 setUp(() {  
 Logger log = new Logger("Test");  
 actions = new ChartActions();  
 ExternalHandler handler = new ExternalHandler(log, actions);  
 store =  
 new ChartStore(log, actions, new ChartEvents(null), null, handler, canvas: new CanvasSize(0, 0, 100, 100));  
  
 renderedInstance = react\_test\_utils  
 .renderIntoDocument(ChartComponent({'actions': actions, 'store': store, 'key': 'chartComponent'}));  
 });  
  
 test("renders", () {  
 expect(renderedInstance, isNotNull);  
 });  
 });  
}

## test/unit/components/connection\_component\_test.dart

import 'dart:async';  
import 'package:test/test.dart';  
  
import 'package:logging/logging.dart';  
import 'package:react/react\_test\_utils.dart' as react\_test\_utils;  
  
import 'package:wFlow/w\_flow.dart';  
  
main() {  
 group("Connection Component", () {  
 ChartStore store;  
 ChartActions actions;  
 var renderedInstance;  
  
 setUp(() async {  
 Logger log = new Logger("Test");  
 actions = new ChartActions();  
 ExternalHandler handler = new ExternalHandler(log, actions);  
 store =  
 new ChartStore(log, actions, new ChartEvents(null), null, handler, canvas: new CanvasSize(0, 0, 100, 100));  
  
 actions.addNode("Test Node 1");  
 Completer actionListener = new Completer();  
 actions.addNode.listen(actionListener.complete);  
 actions.addNode("Test Node 2");  
  
 await actionListener.future;  
  
 Connection conn = new Connection(0, 0, 1);  
  
 renderedInstance = react\_test\_utils.renderIntoDocument(ConnectionComponent({  
 'store': store,  
 'actions': actions,  
 'id': conn.id,  
 'connection': conn,  
 'key': 'wConn',  
 }));  
 });  
  
 test("renders", () {  
 expect(renderedInstance, isNotNull);  
 });  
  
 test("dispatches selectConnection on click", () {  
 Completer actionListener = new Completer();  
 actions.selectConnection.listen(actionListener.complete);  
  
 react\_test\_utils.Simulate.click(react\_test\_utils.findRenderedDOMComponentWithTag(renderedInstance, "div"));  
  
 expect(actionListener.future, completes);  
 });  
 });  
}

## test/unit/components/context\_men\_components\_test.dart

import 'dart:async';  
import 'package:test/test.dart';  
  
import 'package:logging/logging.dart';  
import 'package:react/react\_test\_utils.dart' as react\_test\_utils;  
  
import 'package:wFlow/w\_flow.dart';  
  
main() {  
 group("Context Menu Component", () {  
 ChartStore store;  
 ChartActions actions;  
 var renderedInstance;  
 submitButton() => react\_test\_utils.scryRenderedDOMComponentsWithTag(renderedInstance, "button")[1];  
 cancelButton() => react\_test\_utils.scryRenderedDOMComponentsWithTag(renderedInstance, "button")[0];  
  
 setUp(() {  
 Logger log = new Logger("Test");  
 actions = new ChartActions();  
 ExternalHandler handler = new ExternalHandler(log, actions);  
 store =  
 new ChartStore(log, actions, new ChartEvents(null), null, handler, canvas: new CanvasSize(0, 0, 100, 100));  
  
 Node node = new Node(1, "Test Node");  
  
 renderedInstance = react\_test\_utils.renderIntoDocument(ContextMenuComponent({  
 'store': store,  
 'actions': actions,  
 'key': 'wContextMenu1',  
 'node': node,  
 }));  
 });  
  
 test("renders", () {  
 expect(renderedInstance, isNotNull);  
 });  
  
 test("dispatches submitContextMenuEdit on submit", () {  
 Completer actionListener = new Completer();  
 actions.submitContextMenuEdit.listen(actionListener.complete);  
  
 react\_test\_utils.Simulate.click(submitButton());  
  
 expect(actionListener.future, completes);  
 });  
  
 test("dispatches setContextMenu on cancel", () {  
 Completer actionListener = new Completer();  
 actions.setContextMenu.listen(actionListener.complete);  
  
 react\_test\_utils.Simulate.click(cancelButton());  
  
 expect(actionListener.future, completes);  
 });  
 });  
}

## test/unit/components/control\_component\_test.dart

import 'dart:async';  
import 'package:test/test.dart';  
  
import 'package:logging/logging.dart';  
import 'package:react/react\_test\_utils.dart' as react\_test\_utils;  
  
import 'package:wFlow/w\_flow.dart';  
  
main() {  
 group("Controls Component", () {  
 ChartStore store;  
 ChartActions actions;  
 var renderedInstance;  
  
 setUp(() {  
 Logger log = new Logger("Test");  
 actions = new ChartActions();  
 ExternalHandler handler = new ExternalHandler(log, actions);  
 store =  
 new ChartStore(log, actions, new ChartEvents(null), null, handler, canvas: new CanvasSize(0, 0, 100, 100));  
  
 renderedInstance = react\_test\_utils  
 .renderIntoDocument(ControlsComponent({'actions': actions, 'store': store, 'key': 'controlsComponent'}));  
 });  
  
 test("renders", () {  
 expect(renderedInstance, isNotNull);  
 });  
  
 test("dispatches addNode action on Add Node click", () async {  
 List controlButtons = react\_test\_utils.scryRenderedDOMComponentsWithTag(renderedInstance, "button");  
  
 Completer actionListener = new Completer();  
 actions.addNode.listen(actionListener.complete);  
  
 react\_test\_utils.Simulate.click(controlButtons[0]); // Click the Add Node button  
  
 expect(actionListener.future, completes);  
 });  
  
 test("dispatches deleteNode action on Delete Node click", () async {  
 List controlButtons = react\_test\_utils.scryRenderedDOMComponentsWithTag(renderedInstance, "button");  
  
 Completer actionListener = new Completer();  
 actions.deleteNode.listen(actionListener.complete);  
  
 react\_test\_utils.Simulate.click(controlButtons[1]); // Click the Delete Node button  
  
 expect(actionListener.future, completes);  
 });  
  
 test("dispatches addConnection action on Add Connection click", () async {  
 List controlButtons = react\_test\_utils.scryRenderedDOMComponentsWithTag(renderedInstance, "button");  
  
 Completer actionListener = new Completer();  
 actions.addConnection.listen(actionListener.complete);  
  
 react\_test\_utils.Simulate.click(controlButtons[2]); // Click the Add Connection button  
  
 expect(actionListener.future, completes);  
 });  
  
 test("dispatches deleteConnection action on Delete Connection click", () async {  
 List controlButtons = react\_test\_utils.scryRenderedDOMComponentsWithTag(renderedInstance, "button");  
  
 Completer actionListener = new Completer();  
 actions.deleteConnection.listen(actionListener.complete);  
  
 react\_test\_utils.Simulate.click(controlButtons[3]); // Click the Delete Connection button  
  
 expect(actionListener.future, completes);  
 });  
  
 test("dispatches newGraph action on New Graph click", () async {  
 List controlButtons = react\_test\_utils.scryRenderedDOMComponentsWithTag(renderedInstance, "button");  
  
 Completer actionListener = new Completer();  
 actions.newGraph.listen(actionListener.complete);  
  
 react\_test\_utils.Simulate.click(controlButtons[4]); // Click the New Graph button  
  
 expect(actionListener.future, completes);  
 });  
  
 test("dispatches isSavingGraph action on Save Graph click", () async {  
 List controlButtons = react\_test\_utils.scryRenderedDOMComponentsWithTag(renderedInstance, "button");  
  
 Completer actionListener = new Completer();  
 actions.isSavingGraph.listen(actionListener.complete);  
  
 react\_test\_utils.Simulate.click(controlButtons[5]); // Click the Save Graph button  
  
 expect(actionListener.future, completes);  
 });  
  
 test("dispatches isLoadingGraph action on Load Graph click", () async {  
 List controlButtons = react\_test\_utils.scryRenderedDOMComponentsWithTag(renderedInstance, "button");  
  
 Completer actionListener = new Completer();  
 actions.isLoadingGraph.listen(actionListener.complete);  
  
 react\_test\_utils.Simulate.click(controlButtons[6]); // Click the Load Graph button  
  
 expect(actionListener.future, completes);  
 });  
  
 test("dispatches layoutGraph action on Layout Graph click", () async {  
 List controlButtons = react\_test\_utils.scryRenderedDOMComponentsWithTag(renderedInstance, "button");  
  
 Completer actionListener = new Completer();  
 actions.layoutGraph.listen(actionListener.complete);  
  
 react\_test\_utils.Simulate.click(controlButtons[7]); // Click the Layout Graph button  
  
 expect(actionListener.future, completes);  
 });  
 });  
}

## test/unit/components/modal\_component\_test.dart

import 'package:test/test.dart';  
  
import 'package:logging/logging.dart';  
import 'package:react/react\_test\_utils.dart' as react\_test\_utils;  
  
import 'package:wFlow/w\_flow.dart';  
  
main() {  
 group("Modal Component", () {  
 ChartStore store;  
 ChartActions actions;  
 var renderedInstance;  
  
 setUp(() {  
 Logger log = new Logger("Test");  
 actions = new ChartActions();  
 ExternalHandler handler = new ExternalHandler(log, actions);  
 store =  
 new ChartStore(log, actions, new ChartEvents(null), null, handler, canvas: new CanvasSize(0, 0, 100, 100));  
 });  
  
 test("renders save modal", () {  
 renderedInstance = react\_test\_utils.renderIntoDocument(  
 ModalComponent({'actions': actions, 'store': store, 'key': 'modalComponent', 'type': ModalType.SAVE}));  
 expect(renderedInstance, isNotNull);  
 });  
  
 test("renders load modal", () {  
 renderedInstance = react\_test\_utils.renderIntoDocument(  
 ModalComponent({'actions': actions, 'store': store, 'key': 'modalComponent', 'type': ModalType.LOAD}));  
 expect(renderedInstance, isNotNull);  
 });  
  
 test("renders error modal", () {  
 renderedInstance = react\_test\_utils.renderIntoDocument(ModalComponent({  
 'actions': actions,  
 'store': store,  
 'key': 'modalComponent',  
 'type': ModalType.ERROR,  
 'message': 'test error'  
 }));  
 expect(renderedInstance, isNotNull);  
 });  
 });  
}

## test/unit/components/node\_component\_test.dart

import 'dart:async';  
import 'package:test/test.dart';  
  
import 'package:logging/logging.dart';  
import 'package:react/react\_test\_utils.dart' as react\_test\_utils;  
  
import 'package:wFlow/w\_flow.dart';  
  
main() {  
 group("Node Component", () {  
 ChartStore store;  
 ChartActions actions;  
 var renderedInstance;  
  
 setUp(() {  
 Logger log = new Logger("Test");  
 actions = new ChartActions();  
 ExternalHandler handler = new ExternalHandler(log, actions);  
 store = new ChartStore(log, actions, new ChartEvents(null), null, handler);  
  
 Node node = new Node(1, "Test Node");  
  
 renderedInstance = react\_test\_utils.renderIntoDocument(  
 NodeComponent({'actions': actions, 'store': store, 'id': node.id, 'node': node, 'key': 'nodeComponent'}));  
 });  
  
 test("renders", () {  
 expect(renderedInstance, isNotNull);  
 });  
  
 test("fires selectNode on click", () {  
 Completer actionListener = new Completer();  
 actions.selectNode.listen(actionListener.complete);  
  
 react\_test\_utils.Simulate.click(react\_test\_utils.findRenderedDOMComponentWithTag(renderedInstance, "div"));  
  
 expect(actionListener.future, completes);  
 });  
  
 test("fires setContextMenu on right click", () {  
 Completer actionListener = new Completer();  
 actions.setContextMenu.listen(actionListener.complete);  
  
 react\_test\_utils.Simulate.contextMenu(react\_test\_utils.findRenderedDOMComponentWithTag(renderedInstance, "div"));  
  
 expect(actionListener.future, completes);  
 });  
 });  
}

## test/unit/api\_test.dart

import 'dart:async';  
import 'package:test/test.dart';  
  
import 'package:logging/logging.dart';  
  
import 'package:wFlow/w\_flow.dart';  
  
main() {  
 group("API", () {  
 ChartStore store;  
 ChartActions actions;  
 ChartApi api;  
  
 setUp(() {  
 Logger log = new Logger("Test");  
 actions = new ChartActions();  
 ExternalHandler handler = new ExternalHandler(log, actions);  
 store =  
 new ChartStore(log, actions, new ChartEvents(null), null, handler, canvas: new CanvasSize(0, 0, 100, 100));  
 api = new ChartApi(actions, store);  
 });  
  
 test("initializes", () {  
 expect(api, isNotNull);  
 });  
  
 test("toggles read only", () async {  
 Completer actionListener = new Completer();  
 var sub = actions.setReadOnlyMode.listen(actionListener.complete);  
  
 api.enterReadOnlyMode();  
 await actionListener;  
 sub.cancel();  
  
 expect(store.readOnly, isTrue);  
 expect(api.isReadOnly, isTrue);  
  
 actionListener = new Completer();  
 actions.setReadOnlyMode.listen(actionListener.complete);  
 api.exitReadOnlyMode();  
 await actionListener;  
  
 expect(store.readOnly, isFalse);  
 expect(api.isReadOnly, isFalse);  
 });  
  
 test("toggles drag enabled", () async {  
 Completer actionListener = new Completer();  
 var sub = actions.setDragEnabled.listen(actionListener.complete);  
  
 api.enableDrag();  
 await actionListener;  
 sub.cancel();  
  
 expect(store.dragEnabled, isTrue);  
 expect(api.dragEnabled, isTrue);  
  
 actionListener = new Completer();  
 actions.setDragEnabled.listen(actionListener.complete);  
 api.disableDrag();  
 await actionListener;  
  
 expect(store.dragEnabled, isFalse);  
 expect(api.dragEnabled, isFalse);  
 });  
 });  
}

## test/unit/chart\_store.dart

import 'dart:async';  
import 'package:test/test.dart';  
  
import 'package:logging/logging.dart';  
  
import 'package:wFlow/w\_flow.dart';  
  
main() {  
 group("Chart Store", () {  
 ChartStore store;  
 ChartActions actions;  
  
 setUp(() {  
 Logger log = new Logger("Test");  
 actions = new ChartActions();  
 ExternalHandler handler = new ExternalHandler(log, actions);  
 store =  
 new ChartStore(log, actions, new ChartEvents(null), null, handler, canvas: new CanvasSize(0, 0, 1000, 1000));  
 });  
  
 test("initializes", () {  
 expect(store, isNotNull);  
 });  
  
 test("sets default values", () {  
 expect(store.readOnly, isFalse);  
 expect(store.dragEnabled, isTrue);  
 expect(store.selectedNode, -1);  
 expect(store.selectedConnection, -1);  
 expect(store.nodes, isNotNull);  
 expect(store.nodes.length, 0);  
 expect(store.connections, isNotNull);  
 expect(store.connections.length, 0);  
 });  
  
 group("actions", () {  
 test("handle test graph creation", () async {  
 await \_createTestGraph(actions);  
  
 expect(store.nodes.length, 4);  
 expect(store.nodes[0], isNotNull);  
 expect(store.nodes[3], isNotNull);  
 expect(store.connections.length, 2);  
 expect(store.connections[0], isNotNull);  
 expect(store.connections[1], isNotNull);  
 });  
  
 test("addNode with default content and location", () async {  
 // Fire addNode action  
 Completer actionListenter = new Completer();  
 actions.addNode.listen(actionListenter.complete);  
  
 actions.addNode("");  
 await actionListenter;  
  
 Node expectedNode = new Node(0, "Node 0", color: "#c4ffff", x: 50, y: 100);  
  
 expect(store.nodes.length, 1);  
 expect(store.nodes[0], isNotNull);  
 expect(store.nodes[0], equals(expectedNode));  
 });  
  
 test("selectNode", () async {  
 await \_createTestGraph(actions);  
  
 Completer actionListenter = new Completer();  
 actions.selectNode.listen(actionListenter.complete);  
  
 actions.selectNode(1);  
 await actionListenter;  
  
 expect(store.selectedNode, 1);  
 });  
  
 test("selectNode on selected node deselects", () async {  
 await \_createTestGraph(actions);  
  
 Completer actionListenter = new Completer();  
 var sub = actions.selectNode.listen(actionListenter.complete);  
  
 actions.selectNode(1);  
 await actionListenter;  
 sub.cancel();  
  
 expect(store.selectedNode, 1);  
  
 actionListenter = new Completer();  
 actions.selectNode.listen(actionListenter.complete);  
  
 actions.selectNode(1);  
 await actionListenter;  
  
 expect(store.selectedNode, -1);  
 });  
  
 test("deleteNode", () async {  
 await \_createTestGraph(actions);  
  
 expect(store.selectedNode, -1);  
  
 Completer actionListener = new Completer();  
 actions.deleteNode.listen(actionListener.complete);  
  
 actions.deleteNode(3);  
 await actionListener;  
  
 // expect the node is removed  
 expect(store.nodes.length, 3);  
 expect(store.nodes[3], isNull);  
 // expect the node's connection is removed  
 expect(store.connections.length, 1);  
 expect(store.connections[1], isNull);  
 });  
  
 test("deleteNode on selected node deselects", () async {  
 await \_createTestGraph(actions);  
  
 actions.selectNode(3);  
  
 Completer actionListener = new Completer();  
 actions.deleteNode.listen(actionListener.complete);  
  
 actions.deleteNode(3);  
 await actionListener;  
  
 expect(store.selectedNode, -1);  
 });  
  
 test("deleteNode on nonexistent node", () async {  
 await \_createTestGraph(actions);  
  
 Completer actionListener = new Completer();  
 actions.deleteNode.listen(actionListener.complete);  
  
 actions.deleteNode(5);  
 await actionListener;  
  
 // no nodes or connections should be removed  
 expect(store.nodes.length, 4);  
 expect(store.connections.length, 2);  
 });  
  
 test("moveNode", () async {  
 await \_createTestGraph(actions);  
  
 expect(store.nodes[2].x, 50);  
 expect(store.nodes[2].y, 50);  
  
 Completer actionListener = new Completer();  
 actions.moveNode.listen(actionListener.complete);  
  
 actions.moveNode(new NodeCoord(2, 202, 303));  
 await actionListener;  
  
 expect(store.nodes[2], isNotNull);  
 expect(store.nodes[2].x, 202);  
 expect(store.nodes[2].y, 303);  
 });  
  
 test("modeNode on nonexistent node", () async {  
 await \_createTestGraph(actions);  
  
 Completer actionListener = new Completer();  
 actions.moveNode.listen(actionListener.complete);  
  
 actions.moveNode(new NodeCoord(5, 202, 303));  
 await actionListener;  
  
 expect(store.nodes.length, 4);  
 expect(store.connections.length, 2);  
 // no nodes should be moved  
 expect(store.nodes[0].x, 100);  
 expect(store.nodes[0].y, 100);  
 expect(store.nodes[1].x, 100);  
 expect(store.nodes[1].y, 50);  
 expect(store.nodes[2].x, 50);  
 expect(store.nodes[2].y, 50);  
 expect(store.nodes[3].x, 50);  
 expect(store.nodes[3].y, 100);  
 });  
  
 test("setContextMenuNode", () async {  
 await \_createTestGraph(actions);  
  
 Completer actionListener = new Completer();  
 var sub = actions.setContextMenu.listen(actionListener.complete);  
  
 actions.setContextMenu(store.nodes[3]);  
 await actionListener;  
 sub.cancel();  
  
 expect(store.contextMenuNode, equals(store.nodes[3]));  
  
 // resetting the same value should clear contextMenuNode  
  
 actionListener = new Completer();  
 actions.setContextMenu.listen(actionListener.complete);  
  
 actions.setContextMenu(store.nodes[3]);  
 await actionListener;  
  
 expect(store.contextMenuNode, isNull);  
 });  
  
 test("submitContextMenuEdit", () async {  
 await \_createTestGraph(actions);  
  
 Completer actionListener = new Completer();  
 actions.submitContextMenuEdit.listen(actionListener.complete);  
  
 String newContent = "I am new content!";  
 String newColor = "#ffffff";  
 actions.submitContextMenuEdit(new NodeEdit(2, newContent, newColor));  
 await actionListener;  
  
 Node changedNode = store.nodes[2];  
 expect(changedNode, isNotNull);  
 expect(changedNode.content, newContent);  
 expect(changedNode.color, newColor);  
 });  
  
 test("submitContextMenuEdit on non-existent node", () async {  
 await \_createTestGraph(actions);  
  
 Completer actionListener = new Completer();  
 actions.submitContextMenuEdit.listen(actionListener.complete);  
  
 actions.submitContextMenuEdit(new NodeEdit(5, "", ""));  
 await actionListener;  
  
 // expect state to have not changed  
 expect(store.nodes.length, 4);  
 expect(store.connections.length, 2);  
 });  
  
 test("selectConnection", () async {  
 await \_createTestGraph(actions);  
  
 expect(store.selectedConnection, -1);  
  
 Completer actionListener = new Completer();  
 actions.selectConnection.listen(actionListener.complete);  
  
 actions.selectConnection(1);  
 await actionListener;  
  
 expect(store.selectedConnection, 1);  
 });  
  
 test("selectConnection on already selected connection", () async {  
 await \_createTestGraph(actions);  
  
 expect(store.selectedConnection, -1);  
  
 Completer actionListener = new Completer();  
 var sub = actions.selectConnection.listen(actionListener.complete);  
  
 actions.selectConnection(1);  
 await actionListener;  
 sub.cancel();  
  
 expect(store.selectedConnection, 1);  
  
 actionListener = new Completer();  
 actions.selectConnection.listen(actionListener.complete);  
  
 actions.selectConnection(1);  
 await actionListener;  
  
 expect(store.selectedConnection, -1);  
 });  
  
 test("deleteConnection", () async {  
 await \_createTestGraph(actions);  
  
 Completer actionListener = new Completer();  
 actions.deleteConnection.listen(actionListener.complete);  
  
 actions.deleteConnection(1);  
 await actionListener;  
  
 expect(store.connections.length, 1);  
 expect(store.connections[1], isNull);  
 });  
  
 test("deleteConnection on selected connection", () async {  
 await \_createTestGraph(actions);  
  
 actions.selectConnection(1);  
  
 Completer actionListener = new Completer();  
 actions.deleteConnection.listen(actionListener.complete);  
  
 actions.deleteConnection(1);  
 await actionListener;  
  
 expect(store.selectedConnection, -1);  
 expect(store.connections.length, 1);  
 expect(store.connections[1], isNull);  
 });  
  
 test("deleteConnection on nonexistent connection", () async {  
 await \_createTestGraph(actions);  
  
 Completer actionListener = new Completer();  
 actions.deleteConnection.listen(actionListener.complete);  
  
 actions.deleteConnection(3);  
 await actionListener;  
  
 expect(store.connections.length, 2);  
 expect(store.connections[1], isNotNull);  
 expect(store.connections[0], isNotNull);  
 });  
 });  
 });  
}  
  
\_createTestGraph(ChartActions actions) async {  
 actions.addNode("Test Node 0");  
 actions.addNode("Test Node 1");  
 actions.addNode("Test Node 2");  
 actions.addNode("Test Node 3");  
 actions.moveNode(new NodeCoord(0, 100, 100));  
 actions.moveNode(new NodeCoord(1, 100, 50));  
 actions.moveNode(new NodeCoord(2, 50, 50));  
 actions.addConnection();  
 actions.selectNode(0);  
 actions.selectNode(2);  
 actions.addConnection();  
 actions.selectNode(1);  
  
 Completer actionListenter = new Completer();  
 var sub = actions.selectNode.listen(actionListenter.complete);  
 actions.selectNode(3);  
  
 sub.cancel();  
  
 await actionListenter;  
}

## test/unit/module\_test.dart

import 'dart:async';  
import 'package:test/test.dart';  
  
import 'package:wFlow/w\_flow.dart';  
  
main() {  
 group("Module", () {  
 ChartModule chartModule;  
  
 setUp(() {  
 chartModule = new ChartModule();  
 });  
  
 test("initializes api, events, and components", () {  
 expect(chartModule.api, isNotNull);  
 expect(chartModule.events, isNotNull);  
 expect(chartModule.components, isNotNull);  
 });  
  
 test("loads and unloads", () async {  
 await chartModule.load();  
 expect(chartModule, isNotNull);  
  
 Completer unloadCompleter = new Completer();  
 chartModule.didUnload.listen(unloadCompleter.complete);  
 await chartModule.unload();  
 expect(await unloadCompleter.future, chartModule);  
 });  
 });  
}

## tool/codecov.sh

#!/bin/bash  
  
set -e # causes the whole script to fail if any commands fail  
set -o pipefail # causes pipes to fail if the input command fails  
  
echo "Reporting unit test coverage to codecov..."  
  
if [ -z "$GIT\_BRANCH" ]; then  
 echo "GIT\_BRANCH environment variable not set, skipping Codecov push"  
else  
 TRACKING\_REMOTE="$(git for-each-ref --format='%(upstream:short)' $(git symbolic-ref -q HEAD) | cut -d'/' -f1 | xargs git ls-remote --get-url | cut -d':' -f2 | sed 's/.git$//')"  
 echo $GIT\_BRANCH  
 echo $TRACKING\_REMOTE  
 bash <(curl -s https://codecov.workiva.net/bash) -u https://codecov.workiva.net -X fix -t $CODECOV\_TOKEN -B $GIT\_BRANCH -r $TRACKING\_REMOTE -f coverage/coverage.lcov || echo "ERROR: Codecov failed to upload reports."  
fi  
  
echo "Reporting unit test coverage to codecov...done"

## tool/dev.dart

library tool.dev;  
  
import 'package:dart\_dev/dart\_dev.dart' show dev, config, Environment, TestRunnerConfig;  
  
main(List<String> args) async {  
 // https://github.com/Workiva/dart\_dev  
  
 // Perform task configuration here as necessary.  
  
 List<String> directories = ['lib/', 'test/', 'web/', 'bin/'];  
  
 config.analyze  
 ..entryPoints = directories  
 ..fatalHints = false  
 ..strong = false;  
  
 config.format  
 ..directories = directories  
 ..lineLength = 120;  
  
 config.coverage..pubServe = true;  
  
 config.genTestRunner.configs = [  
 new TestRunnerConfig(  
 directory: 'test',  
 env: Environment.browser,  
 filename: 'generated\_runner\_test',  
 genHtml: true,  
 dartHeaders: [  
 "import 'package:react/react\_client.dart' as react\_client;",  
 ],  
 preTestCommands: [  
 "react\_client.setClientConfiguration();",  
 ],  
 htmlHeaders: [  
 '<script src="packages/react/react\_with\_addons.js"></script>',  
 '<script src="packages/react/react\_dom.js"></script>',  
 ]),  
 ];  
  
 config.test  
 ..concurrency = 1  
 ..platforms = ['content-shell']  
 ..pubServe = true  
 ..unitTests = ['test/generated\_runner\_test.dart'];  
  
 await dev(args);  
}

## web/index.hmtl

<!DOCTYPE html>  
  
<html>  
 <head>  
 <meta charset="utf-8">  
 <meta http-equiv="X-UA-Compatible" content="IE=edge">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
  
 <title>wFlow</title>  
  
 <!-- Bootstrap -->  
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">  
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-theme.min.css" integrity="sha384-rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp" crossorigin="anonymous">  
 <link rel="stylesheet" type="text/css" href="testing.css">  
 </head>  
  
  
<body>  
 <nav class="navbar navbar-default navbar-fixed-top" role="navigation">  
 <div class="container">  
 <div class="navbar-header">  
 <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1">  
 <span class="sr-only">Toggle navigation</span>  
 <span class="icon-bar"></span>  
 <span class="icon-bar"></span>  
 <span class="icon-bar"></span>  
 </button>  
 <a class="navbar-brand" href="#">wFlow</a>  
 </div>  
 <!-- Collect the nav links, forms, and other content for toggling -->  
 <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">  
 <ul class="nav navbar-nav">  
 <li>  
 <a href="#">About</a>  
 </li>  
 <li>  
 <a href="#">Kanban</a>  
 </li>  
 <li>  
 <a href="#">Feedback</a>  
 </li>  
 </ul>  
 </div>  
 </div>  
 </nav>  
  
 <div class="container">  
 <div class="row">  
 <div class="col-md-9">  
 <div class="well full-page" id="content-container">  
 </div>  
 </div>  
 <div class="col-md-3">  
 <div class="well" id="side-panel-container">  
 </div>  
 </div>  
 </div>  
 <hr>  
 <footer>  
 <div class="row">  
 <div class="col-lg-12">  
 <p>MacKenzie O'Bleness &amp Lisa Peters</p>  
 <p>An MSU Software Factory &amp Workiva Project</p>  
 </div>  
 </div>  
 </footer>  
 </div>  
  
 <script src="packages/react/react.js"></script>  
 <script src="packages/react/react\_dom.js"></script>  
 <script type="application/dart" src="main.dart"></script>  
 <script src="packages/browser/dart.js"></script>  
  
 </body>  
</html>

## web/main.dart

import 'dart:html';  
  
import 'package:react/react.dart' as react;  
import 'package:react/react\_dom.dart' as react\_dom;  
import 'package:react/react\_client.dart' as react\_client;  
  
import 'package:wFlow/w\_flow.dart';  
  
void main() {  
 react\_client.setClientConfiguration();  
  
 ChartModule module = new ChartModule();  
  
 react\_dom.render(  
 react.div({}, [module.components.title(), module.components.content()]), querySelector('#content-container'));  
  
 react\_dom.render(react.div({}, [module.components.controls()]), querySelector('#side-panel-container'));  
}

## web/testing.css

button {  
 background: transparent;  
 border: none;  
 margin: 5px;  
 text-decoration: bold;  
}  
  
button:hover {  
 background-color: #EAEAEA;  
}  
  
dialog {  
 position: relative;  
  
 width: 600px;  
 margin-left: -300px;  
  
 height: 400px;  
 margin-top: -200px;  
}  
  
body {  
 padding-top: 70px; /\* Required padding for .navbar-fixed-top. Remove if using .navbar-static-top. Change if height of navigation changes. \*/  
 background: #F1F1F1;  
}  
  
footer {  
 margin: 50px 0;  
}  
  
.well {  
 background: #FFFFFF;  
}  
  
.full-page {  
 height: 650px;  
}  
  
.wNode {  
 border-style: solid;  
 border-color: #AAAAAA;  
 border-radius: 5px;  
 border-width: 1px;  
 display: inline-block;  
 word-wrap: break-word;  
 overflow:hidden;  
 height: 100px;  
 width: 100px;  
 padding: 20px;  
 position: absolute;  
 z-index: 1;  
}  
  
.wContextMenu {  
 background: #F9F9F9;  
 border-style: solid;  
 border-radius: 3px;  
 border-color: #AAAAAA;  
 border-width: 2px;  
 display: inline-block;  
 position: absolute;  
 padding: 10px;  
 z-index: 10;  
}  
  
.wNode.selected {  
 border-width: 3px;  
 border-color: #000000;  
}  
  
.wConnection {  
 height: 2px;  
 background: #AAAAAA;  
 -webkit-transform-origin: 0 100%;  
}  
  
.wConnection.selected {  
 height: 4px;  
 background: #000000;  
}  
  
.wModalOverlay {  
 background-color: rgba(0,0,0,0.6);  
 position: absolute;  
 left: 0;  
 margin-left: 15px;  
 z-index: 1000;  
 width: 96.5%;  
 height: 100%;  
 padding-top: 15%;  
 top: 0;  
}  
  
.wModalBox {  
 background-color: #fefefe;  
 margin: auto;  
 padding: 20px;  
 border: 1px solid #888;  
 width: 80%;  
 display: block;  
}  
  
.wModalTitle h3 {  
 text-align: center;  
 margin-top: 0px;  
}  
  
.wModalInput {  
 padding: 5px;  
}  
  
.wModalInput input {  
 width: 80%;  
}  
  
.wModalInput form {  
 display: inline;  
 padding: 10px;  
}  
  
.wModalButton {  
 width: 45%;  
 border: 1px solid;  
 border-radius: 3px;  
}  
  
.wModalConfirmButton {  
 background-color: rgba(47, 185, 71, 0.71);  
}  
  
.wModalConfirmButton:hover {  
 background-color: rgba(47, 185, 71, 0.90);  
}  
  
.wModalCancelButton {  
 background-color: rgba(210, 14, 14, 0.71);  
}  
  
.wModalCancelButton:hover {  
 background-color: rgba(210, 14, 14, 0.90);  
}  
  
.wModalError {  
 text-align: center;  
}  
  
/\* Navbar CSS - generated by TWBSColor \*/  
.navbar-default {  
 background-color: #66cc00;  
 border-color: #59b200;  
 background-image: linear-gradient(to bottom,#66cc00 0,#64C503 100%);  
}  
.navbar-default .navbar-brand {  
 color: #ffffff;  
}  
.navbar-default .navbar-brand:hover,  
.navbar-default .navbar-brand:focus {  
 background-color: #59b200;  
}  
.navbar-default .navbar-text {  
 color: #ffffff;  
}  
.navbar-default .navbar-nav > li > a {  
 color: #ecf0f1;  
}  
.navbar-default .navbar-nav > li > a:hover,  
.navbar-default .navbar-nav > li > a:focus {  
 color: #ffffff;  
}  
.navbar-default .navbar-nav > .active > a,  
.navbar-default .navbar-nav > .active > a:hover,  
.navbar-default .navbar-nav > .active > a:focus {  
 color: #ffffff;  
 background-color: #59b200;  
}  
.navbar-default .navbar-nav > .open > a,  
.navbar-default .navbar-nav > .open > a:hover,  
.navbar-default .navbar-nav > .open > a:focus {  
 color: #ffffff;  
 background-color: #59b200;  
}  
.navbar-default .navbar-toggle {  
 border-color: #59b200;  
}  
.navbar-default .navbar-toggle:hover,  
.navbar-default .navbar-toggle:focus {  
 background-color: #59b200;  
}  
.navbar-default .navbar-toggle .icon-bar {  
 background-color: #ecf0f1;  
}  
.navbar-default .navbar-collapse,  
.navbar-default .navbar-form {  
 border-color: #ecf0f1;  
}  
.navbar-default .navbar-link {  
 color: #ffffff;  
}  
.navbar-default .navbar-link:hover {  
 color: #ecf0f1;  
}  
  
@media (max-width: 767px) {  
 .navbar-default .navbar-nav .open .dropdown-menu > li > a {  
 color: #ecf0f1;  
 }  
 .navbar-default .navbar-nav .open .dropdown-menu > li > a:hover,  
 .navbar-default .navbar-nav .open .dropdown-menu > li > a:focus {  
 color: #ffffff;  
 }  
 .navbar-default .navbar-nav .open .dropdown-menu > .active > a,  
 .navbar-default .navbar-nav .open .dropdown-menu > .active > a:hover,  
 .navbar-default .navbar-nav .open .dropdown-menu > .active > a:focus {  
 color: #ffffff;  
 background-color: #59b200;  
 }  
}

## PULL\_REQUEST\_TEMPLATE.md

Kanban Card: [Kanban Link]  
# Problem  
  
# Solution  
  
# Review

## README.md

# wFlow-spike  
> A machine learning assisted flowchart module.  
## About  
wFlow is an [Montana State University Software Factory project](http://www.bobcatsoftwarefactory.com/) sponsored by Workiva. The ultimate goal of the project is to create a flowchart creation and editing module that can be integrated into wDesk, primarily for SOX process flow diagram creation. Eventually we hope to be able to automatically create flowcharts for users using the SOX data that they've already provided, using a machine learning system to lay out the graph.  
  
Currently this project contains a front-end w\_module library for simple flowchart editing, a python implementation of the Q-learning algorithm for graph layout, a mongodb database for saving graphs for future ML training, and a lightweight dart server for interfacing with the ML and the database.  
  
### Resources  
Since this project was based out of the Software Factory, it uses no non-public Workiva tools or code. Below is a list of resources and technology used and their documentation.  
#### Front-End Tools & Tech  
- [Dart Language Tour](https://www.dartlang.org/guides/language/language-tour)  
- [w\_module Documentation](https://github.com/Workiva/w\_module#w\_module)  
- [w\_flux Documentation](https://github.com/Workiva/w\_flux#w\_flux)  
- [react-dart Documentation](https://github.com/cleandart/react-dart#dart-wrapper-library-for-facebookreact)  
- [What is Flux?](http://fluxxor.com/what-is-flux.html)  
  
#### Back-End Tools & Tech  
- [Server-side Dart Introduction](https://dart-lang.github.io/server/server.html)  
- [MongoDB Documentation](https://docs.mongodb.com/manual/introduction/)  
- [mongo\_dart Documentation](https://github.com/mongo-dart/mongo\_dart#mongo-dart---mongodb-driver-for-dart-programming-language)  
- [Q-Learning Video Series](https://www.youtube.com/playlist?list=PLWi7UcbOD\_0u1eUjmF59XW2TGHWdkHjnS)  
  
#### Dev Tools  
- [dart\_dev](https://github.com/Workiva/dart\_dev#dart-dev-tools)  
  
  
## Project Structure  
`MachineLearning/` contains the python implementation of the q-learning algorithm. The server calls `MachineLearning/Q-Learning.py` when it recieves a request to layout a graph.  
  
`bin/` contains the server code. According to [pub package conventions](https://www.dartlang.org/tools/pub/package-layout#command-line-apps) dart command-line/vm scripts should all go in this folder.   
  
`graphBase/` contains the schema for the mongodb database.  
  
`lib/` contains the consumable wFlow client library. The client follows the w\_module pattern.  
  
`test/` contains the tests for the client library.  
  
`tool/` contains development tools, including the dart\_dev configuration.  
  
`web/` contains an example web application which consumes and displays the client library.  
  
  
## Setup & Running the Site  
- [Install Dart](https://www.dartlang.org/install).  
- [Install Dartium](https://webdev.dartlang.org/tools/dartium).  
  
Pub should be installed with the Dart SDK, but if you run into issues check out https://www.dartlang.org/tools/pub/installing  
  
### Run the Server  
From the root project directory:  
```bash  
$ dart bin/server.dart  
```  
  
### Run the Example Site  
From the root project directory:   
```bash  
$ pub get  
$ pub serve  
```  
Navigate localhost:8080 in Dartium. You can use any other browser - but it may take longer to load as the Dart is transpiled to JavaScript.  
  
  
## Development  
### Contributing  
```bash  
# to auto-format dart code  
$ pub run dart\_dev format  
# to analyze dart code for errors  
$ pub run dart\_dev analyze  
```  
Smithy will fail if there is unformatted code or if the analyzer returns any errors or warnings.  
  
### Testing  
```bash  
# to run all dart unit tests  
$ pub run dart\_dev test  
# to run matching dart tests  
$ pub run dart\_dev test --name "test regex goes here"  
# to generate a code coverage report  
$ pub run dart\_dev coverage  
  
# to run matching python tests  
# run following from top dir of project  
$ export PYTHONPATH=.  
# then run  
$ python "path to test"  
```  
  
After adding or renaming dart tests, run `pub run dart\_dev gen-test-runner` to regenerate the test runner.  
  
  
## Contacts  
##### Workiva:  
Devin Gray: devin.gray@workiva.com   
  
##### MSU:  
Clemente (Clem) Izurieta: clem.izurieta@gmail.com   
  
##### Contributors:  
MacKenzie O'Blenes: mackenzie.obleness@gmail.com  
  
Lisa Peters: lisapeters.peters@gmail.com

## Codecov.yml

codecov:  
 bot: codecov-wf  
 notify:  
 require\_ci\_to\_pass: true  
comment:  
 behavior: default  
 layout: header, diff, changes, sunburst, uncovered, tree  
 require\_changes: false  
coverage:  
 precision: 2  
 range: "100...100"  
 round: down  
 status:  
 changes: false  
 patch: false  
 project: false

## Pubspec.yaml

name: wFlow  
description: A graph layout program that needs a better description.  
authors:  
 - MacKenzie O'Bleness <mackenzie.obleness@gmail.com>  
 - Lisa Peters <lisapeters.peters@gmail.com>  
  
dependencies:  
 react: '^3.0.0'  
 dnd: '^0.3.0'  
 http: '^0.11.3'  
 w\_flux: ^2.0.0  
 w\_module: '^1.0.0'  
 mongo\_dart: any  
 bson: '^0.2.0'  
 crypto: '>=0.9.2 <3.0.0'  
 logging: '>=0.8.0 <2.0.0'  
 mongo\_dart\_query: '^0.2.2'  
 collection: ">=1.4.0 <2.0.0"  
 uuid: "^0.5.0"  
 dialog: "^0.5.0"  
  
dev\_dependencies:  
 coverage: "^0.7.3"  
 dart\_dev: "^1.0.0"  
 dart\_style: ">=0.2.0 <0.3.0"  
 dartdoc: ">=0.8.0 <=0.10.0"  
 test: "^0.12.0"  
  
transformers:  
- test/pub\_serve:  
 $include: test/\*\*\_test.dart  
  
environment:  
 sdk: '>=1.19.1 <2.0.0'

## Smithy.yaml

language: javascript  
  
runner\_image: drydock-prod.workiva.org/workiva/smithy-runner-dart:109965 # 1.21.0  
  
env:  
 - CODECOV\_TOKEN='83dbdb01-478a-42a9-8eaa-e2fc2357f6a7'  
  
before\_script:  
 - pub get  
  
script:  
 - pub run dart\_dev format --check  
 - pub run dart\_dev analyze  
 - xvfb-run -s '-screen 0 1024x768x24' pub run dart\_dev test  
 - xvfb-run -s '-screen 0 1024x768x24' pub run dart\_dev coverage --no-html  
  
after\_script:  
 - tool/codecov.sh

# 

# Section 2: Teamwork

While both team members were involved in all aspects of the project, due to differences in work and school experience, rough focuses were prescribed for each team member at the beginning of the project.

MacKenzie O’Bleness

**Primary Focus**: Front-End Architecture and UI

**Lines Contributed**: 3,850 ++ / 742 --

**Percentage of Time**: 50%

Lisa Peters

**Primary Focus**: Back-End Architecture and Machine Learning

**Lines Contributed**: 2,937 ++ / 1,645 --

**Percentage of Time**: 50%

The team would meet to work together three times a week - one hour sessions took place Tuesday and Thursday mornings, with a longer two to four hour session Thursday afternoon. The team took these times for status updates, design discussion, pair programming, and project planning. Outside of these times, the team members worked independently on their respective work, coordinating as needed through email and text.

# 

# 

# Section 3: Design Patterns

One of the core patterns use by Workiva in their front-end applications is the Flux pattern, an alternative to the MVC pattern developed by Facebook. In addition to the benefits of following the Flux pattern in terms of ease of integration with wDesk and continuity across Workiva repositories, the Flux pattern is also well suited to applications like wFlow that are expected to scale in size and complexity of state. The use of the pattern provides the ability to split state across multiple stores, trigger re-renders of components from multiple stores or limit re-rendering, all while maintaining the unidirectional dataflow that makes system behavior easy to trace and debug.

The Flux pattern consists of four components - Actions, Stores, Views, and the Dispatcher. wFlow is using the w\_flux library, which builds off of RefluxJS. These libraries remove some of the boilerplate surrounding the implementation of the Flux pattern. The Dispatcher component, which is responsible for handling dispatched actions and calling the appropriate registered callbacks, is implemented by w\_flux, so it isn’t present in the above code.

The Actions components are collected in the ChartActions class, which contains the declaration of all actions in the application. The ChartStore uses this class to register callbacks for each action, while each component has a reference to the class in order to dispatch events on user interactions.

The Stores components in wFlow is handled by one store, the ChartStore, which extends w\_flux.Store. In the initialization of the store, it subscribes to the various actions and registers a handler method with each, listening for those actions to take place and performing operations as needed.

The Views consist of the various types of Components - each component is responsible for rendering some part the view to the user. Some components render base html elements like divs and text (for example, the NodeComponent class), while other components (like the ChartComponent class) also render other components.

Each of these pieces is highlighted in the code above in yellow.

# Section 4: Project Report

### Introduction & Background

Workvia is a cloud-based software company specializing in financial accountability and reporting solutions for businesses. The core of these solutions is the wDesk platform, an integrated data management, analysis, and reporting system that combines and links data from documents, spreadsheets, presentations in a single environment.

One such wDesk solution is based around Sarbane-Oxley compliance (SOX), which requires that businesses establish and report on the efficacy of internal audits and controls in order to protect shareholders and the general public from accounting errors or fraudulence. In addition to significant amounts of documentation and data that must be kept on file, an important aspect to SOX compliance is the inclusion of process flowcharts to describe key controls.

Currently, these flowcharts must be made in an external program to the wDesk ecosystem, then embedded into reports as images, repeating this process whenever edits need to be made. This process is not in line with the self-sufficient and integrated wDesk environment, and the user experience could be improved significantly.

With this pain point in mind, our team aimed to improve the user experience surrounding SOX process flowcharts by creating a module for the wDesk platform that generates flowcharts from existing SOX data and allows the user to edit it in-line. In addition, we hoped to improve the generation and layout of these graphs without additional burden on the user through the use of machine learning. This introduction of a machine learning integrated application will help pave the way for future machine learning integrations into the wDesk ecosystem.

### Requirements

As the long term goal of this project is to create a product that can be easily integrated into wDesk, it was essential to follow the practices, technologies, and architectures that are standard at Workiva.

#### Languages

A large proportion of new development for wDesk are built on a similar technology stack. For the front-end, Dart is the primary language. Dart is a Google-created programming language that is popular for front-end development due to its ability to be transpiled to JavaScript. A variety of Workiva front-end frameworks and libraries, such as w\_flux and w\_module, discussed below, are written for dart web applications. Dart’s ability to be transpiled and the access its use provides to Workiva’s public libraries made it a natural choice for a front-end language for wFlow.

The technical requirements for the back-end of wFlow are slightly more complicated. While Go, another Google created language in the style of C, is the standard choice for a server-side language, Workiva has few public back-end libraries available that could be leveraged for this project. In addition, there’s a non-trivial amount of server-side integration that is only realistically done with internal technology, including managing permissions, accounts, security, and access to other wDesk information. Without the access to these systems, the wFlow server is fairly lightweight and easily modified or maintained.

Since the usual reasons for using Go for the server weren’t particularly compelling in this case, and to limit the number of languages in the project, Dart was used for the server. This provided consistency with the language of choice front-end, easily compatible libraries for communications between front-end and back-end, as well as a simple way to call the machine learning algorithm, which is in Python.

For the machine learning algorithm, Python was originally chosen in order to leverage the PyBrain library, which had thriving community support and established machine learning libraries and tutorials. However, shortly after the beginning of the project, Pybrain was unexpectedly deprecated. Furthermore, Pybrain’s website was no longer functional, preventing access to the download, documentation, and tutorials. The team continued to use Python as it is a common language at Workiva, the team also had familiarity with the language, and for the potential situation of Pybrain once more becoming a valid option (as of this writing, there is an existing ticket and volunteer to fix the website on Pybrain’s github page).

#### Architecture & Interfaces

The Workiva front-end architecture culminates into two separate but compatible libraries, w\_module and w\_flux.

* [**w\_module**](https://github.com/Workiva/w_module) prescribes a pattern for a module’s public interface, and standardizing the way modules are consumed. In order for wFlow to be easily consumed, it should implement the w\_module pattern.
* While w\_module describes a public facing interface, [**w\_flux**](https://github.com/Workiva/w_flux) describes the internal architecture of the module. It is based on Facebook’s Flux pattern, which is similar to MVC, but enforces a unidirectional dataflow through its actions, stores, and components.

The combination of w\_module and w\_flux results in an easily consumable module that scales well with added complexity, through its reusable components and unidirectional data flow.

#### Performance

Workiva prides itself on the quality of its product, and performance is essential, especially given that the platform itself is web-based and financial documents that wDesk is frequently used for are very large. It’s important that wFlow be quick and unobtrusive to the user, and that editing the graph feel responsive even when the graph is large or dense.

#### Development

It is also important that the quality of the code is commensurate with the quality expected for code written at Workiva. In that aim, the same best practices and review standards were used for wFlow that are used at Workiva.

Each pull request (PR), must fill out a templement that describes the change that was made, the problem or improvement it addresses, the ticket number associated with the work, and a description of how to test the change.

Additionally, the project utilized Workiva’s continuous integration (CI) tools to automatically run unit tests, check formatting, and perform static analysis of the code base. Should any of these checks fail, the issue is reported on the PR and merge will be prevented until the issue is fixed.

Each PR also required a +10 from a teammate, which consists of a thorough read-through of the code and manual testing (including the instructions described in the PR). A +10 indicates an endorsement of the quality of the code and confirmation that the behavior of the change is as described and does not cause any regressions. On conclusion of the CI checks and the +10, the teammate who gave the +10 was allowed to merge. While reviewing code that one wrote themselves is not in the spirit of code review, sometimes PRs will be authored by both teammates. In these cases, the teammate who wrote the minority of changes are responsible for the +10, with the majority teammate reviewing the commits authored by their partner.

While tests were not required for each feature as it was written, in an attempt to keep velocity up and allow functionality to stabilize before significant time was spent building testing infrastructure, the development of unit tests was a priority. At the conclusion of the project, the client side and machine learning is just under 70% covered, with an effort made to cover functionality over boilerplate or UI. This level of coverage is commensurate with the level of coverage on many Workiva repositories. Due to the complexity of created server and database tests, and the potential malleability of both, tests were deferred for Server and DatabaseController classes.

Following these standards helps to make an auditable trail that shows the project’s commitment to quality, and creates a history of development that will allow future collaborators to understand the decisions that were made and the issues that were encountered.

### Methodology

The team’s primary form of project planning was the creation of a series of deliverables, from which units of work was pulled. One of the primary concerns during the initial planning work was the feasibility of creating a working product without any Workiva internal tools that still complied with the required patterns to be consumed by wDesk. Hoping to reduce the amount of throw-away code and give the team as much time as possible to potentially get access to the necessary private libraries the original deliverable list prioritized non-Workiva dependent as much as possible.

**Original Deliverable Schedule**

#### **MVP 1: Basic User Interaction**

1. Simple graph format
   1. One node type
   2. Only bi-directional connections
2. A user-interactable graph
   1. Allows:
      1. Adding nodes/connections
      2. Deleting nodes/connections
      3. Moving nodes/connections
   2. Can be created blank from scratch
   3. Can be saved to a database, for later use in an ML training set
3. Trivial or open source layout system

## MVP 2: Extended Graph Capabilities

1. Basic SOX compatible graph representation
   1. Multiple node types
   2. Directed connections
2. Extended user interaction
   1. Modification of existing nodes/connections
3. Allow input of new graph from SOX database

## MVP 3: Machine Learning Initial Implementation

1. Train ML implementation off of training database graphs
2. Integrate ML as the main layout system

## MVP 4: SOX Specific Graph Requirements

1. Support Swimlanes
   1. Support swimlanes in graph structure
   2. Support rendering of swimlanes
   3. Support layout of nodes in associated swimlanes
2. Support Node Annotations

## MVP 5: wDesk Graph Module Integration

1. Implement the w\_module pattern
2. Implement the w\_flux pattern

## MVP 6: General wML Library

1. Modify graph ML implementation to be extendible to a generic project
2. Develop API for ML library

After further research was conducted, it was concluded that, while the SOX integration deliverable would still be difficult while external to Workiva, implementing the patterns required to be consumed by wDesk was achievable without too much additional work. Because implementing these patterns early would reduce the amount of refactoring later, this deliverable was moved to the beginning of the project instead of the end. In addition, the team made the decision to prioritize a working ML integration for future teams to iterate on over implementing additional graph features. The final deliverable schedule, including future deliverables is shown below.

**Final Deliverable Schedule**

## MVP 1: wDesk Graph Module Integration (Originally MVP 5)

1. Implement the w\_module pattern
2. Implement the w\_flux pattern

#### **MVP 2: Basic User Interaction** (Originally MVP 1)

1. Simple graph format
   1. One node type
   2. Only bi-directional connections
2. A user-interactable graph
   1. Allows:
      1. Adding nodes/connections
      2. Deleting nodes/connections
      3. Moving nodes/connections
   2. Can be created blank from scratch
   3. Can be saved to a database, for later use in an ML training set
3. Trivial or open source layout system

## MVP 3: Machine Learning Initial Implementation

1. Train ML implementation off of training database graphs
2. Integrate ML as the main layout system

## (Future) MVP 4: Extended Graph Capabilities (Originally MVP 2)

1. Basic SOX compatible graph representation
   1. Multiple node types
   2. Directed connections
2. Extended user interaction
   1. Modification of existing nodes/connections
3. Allow input of new graph from SOX database

## (Future) MVP 5: SOX Specific Graph Requirements (Originally MVP 4)

1. Support Swimlanes
   1. Support swimlanes in graph structure
   2. Support rendering of swimlanes
   3. Support layout of nodes in associated swimlanes
2. Support Node Annotations

## (Future) MVP 6: General wML Library

1. Modify graph ML implementation to be extendible to a generic project
2. Develop API for ML library

The team believes that the final order of deliverables creates the best well rounded project base for future iterations to build on.

### 

### 

### 

### Results and Interpretation

Currently the wFlow system has a functioning front-end for user interactions and back-end for database operations and the machine learning layout algorithm. On the front-end, the user can create simple graphs by adding nodes and connections or modify the graph by moving the nodes within the canvas. Currently, all nodes are the same size and shape, but node text and color can be changed. Additionally, there are controls to save and load graphs, as well as to initiate the graph layout algorithm.

On the back-end, the server receives messages from the client, which can be requests to save, load, or layout a graph. These messages are structured in http GET and POST requests, with any graphs formatted in json according the the graph schema included in the project’s documentation.

The server forwards save and load requests onto a database controller, which performs the appropriate operations and returns the results or errors to the server, where they are then returned as http responses to the client.

When the layout program is called, it creates an instance of the Q-Learning reinforcement learning algorithm. The Q-Learning algorithm uses a class designed for this application called *LearningParameter* to define what parameters and metrics make a graph useful and action to take to obtain such a graph. One such *LearningParameter* is *ConnectedNodePair*, which details that if any two nodes are connected, and are overlapping, they should be separated in order to maximize graph readability. Thus, when a graph contained overlapping connected nodes is sent into the layout algorithm, the resulting graph after the algorithm has run has been modified to where the previously overlapping nodes are moved further apart. This graph is returned to the server, where is it formatted into a response object and returned to the client. The client then displays the laid out graph for the user.

### Limitations of the Q-Learning Algorithm

Q-Learning has the advantages of being intuitively matchable to the layout problem, as it iterates over state-action pairs to find the best action to take while in a particular state. In the layout problem, the state is the position of the nodes and connections of the graph, and the actions are moving nodes. The result of the Q-Learning algorithm is a policy dictionary, where given a state-action key, the value is a numeric representation of how useful for the end goal it is to take that action while in that state.

The Q-Learning algorithm has two limitations as it is currently implemented. While graph can be saved to a database, during the timeframe of the project it was infeasible to create and save the large number of appropriate graphs to the database. The algorithm requires a very large set of graphs in order to create a robust and persistent policy. As the code works now, a single graph goes through the algorithm until optimal fitness is reached or the algorithm reaches its max iterations. This means that while the policy is created, it is ephemeral, lasting only for that particular stack call and applying only to that particular graph. Thus, the main advantage of using the Q-Learning algorithm is not realizable until the database of training graphs is created and the created policy begins to have persistence.

The second limitation of the algorithm is the amount of time it takes to finish and form a complete policy dictionary. As the algorithm has been implemented in Python and thus the code is not complied before runtime, this problem was exacerbated. While no graphs of extreme complexity were tested, even simplistic graphs of less than 10 nodes occasionally hit the max iteration upper bound before finding optimal fitness, and creating a full policy would have taken longer still. Fortunately, for future iterations there are solutions to this problem. Several different improvements exist for the standard Q-Learning algorithm that dramatically increase its performance time, including Speedy Q-Learning and a deep learning variation. Additionally, parallelization of the algorithm is a viable option that could result in drastic improvements.

### Limitations of the Functionality

The client currently only supports a limited set of operations. Editing the text and color of the nodes, the node’s position in the canvas, and adding connections between the nodes. To handle the SOX use cases, the graphs would need to support swimlanes, multiple (potentially user customizable) types and sizes of nodes, full text editing support inside of the text, and the ability to add annotations and other attachments to nodes, among other features. This functionality would require changes to the components, to render the appropriate UI for controlling and displaying each feature, the graph schema, to store and retrieve the new data in the graph, and potentially new learning parameters for the ML algorithm.

### Future Direction

In addition to addressing the limitations described above, there is plenty of room for future work on this project. Prior to the final integration with the wDesk platform, which may itself require some code changes depending on the needs on the consuming component, the application should be integrated and tested with real SOX data. This step poses several challenges. Workiva customers expect a very high level of security, and wFlow would need to undergo a thorough review to ensure the application meets that level of security. This would likely require switching database systems to be Workiva internal, using Workiva secure messaging systems, and implementing a system of accounts and permissions, among any issues found by the security review. The actual technical integration of the SOX system would also require the use of Workiva internal technology, which was out of the scope of this iteration of the project.

However, once the integration with the SOX system is complete, and the ML algorithm has access to real world data, there will lots of interesting opportunities to tune the algorithm, and potentially even compare and contrast other ML systems.

### Conclusions

While there remains much to do on this project, it is the hope of our team that we have provided a solid basis for future experimentation and improvement. We believe that the Q-Learning algorithm provides a great basis for ML introduction into this module and into wDesk, both in its modified form for the short term, and in the fully trained long term.

### References

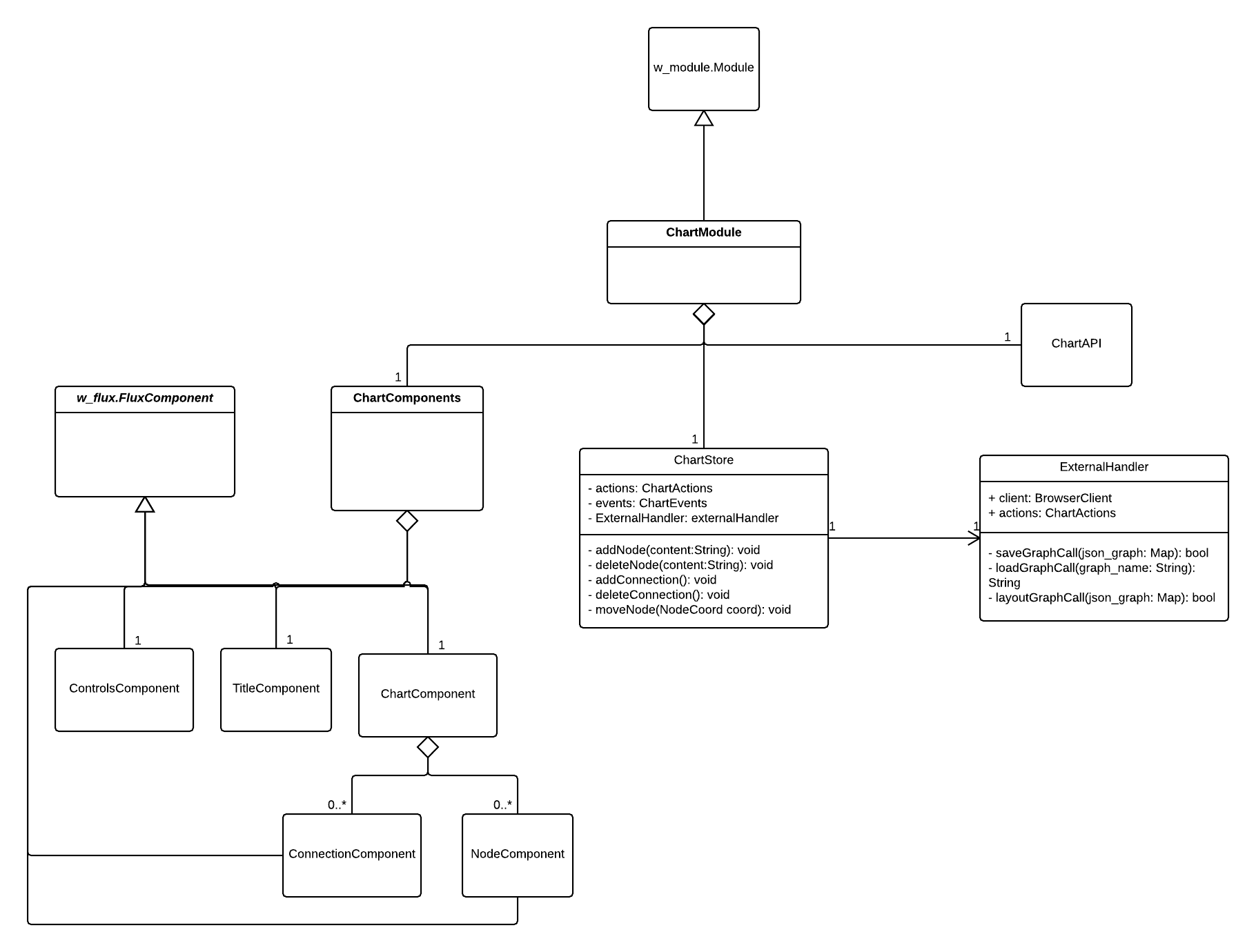
* w\_module Documentation: <https://github.com/Workiva/w_module#w_module>
* w\_flux Documentation: <https://github.com/Workiva/w_flux#w_flux>
* react\_dart Documentation: <https://github.com/cleandart/react-dart#dart-wrapper-library-for-facebookreact>
* Flux Pattern Overview: <http://facebook.github.io/flux/docs/in-depth-overview.html#content>
* *Artificial Intelligence: A Modern Approach (3rd ed.)* by Stuart Russell and Peter Norvig
* *Learning from Delayed Rewards* by C.J. Watkins

### 

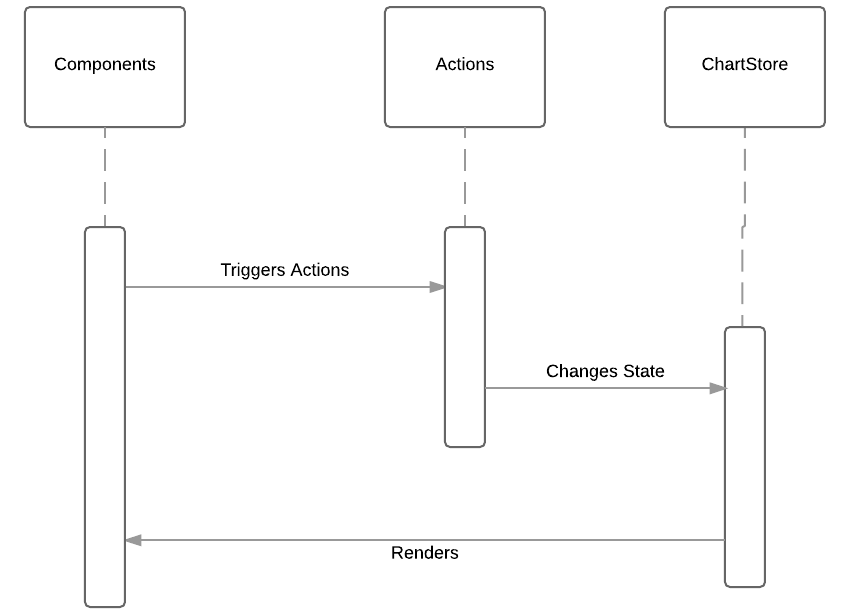
### **Section 5: UML**

# 

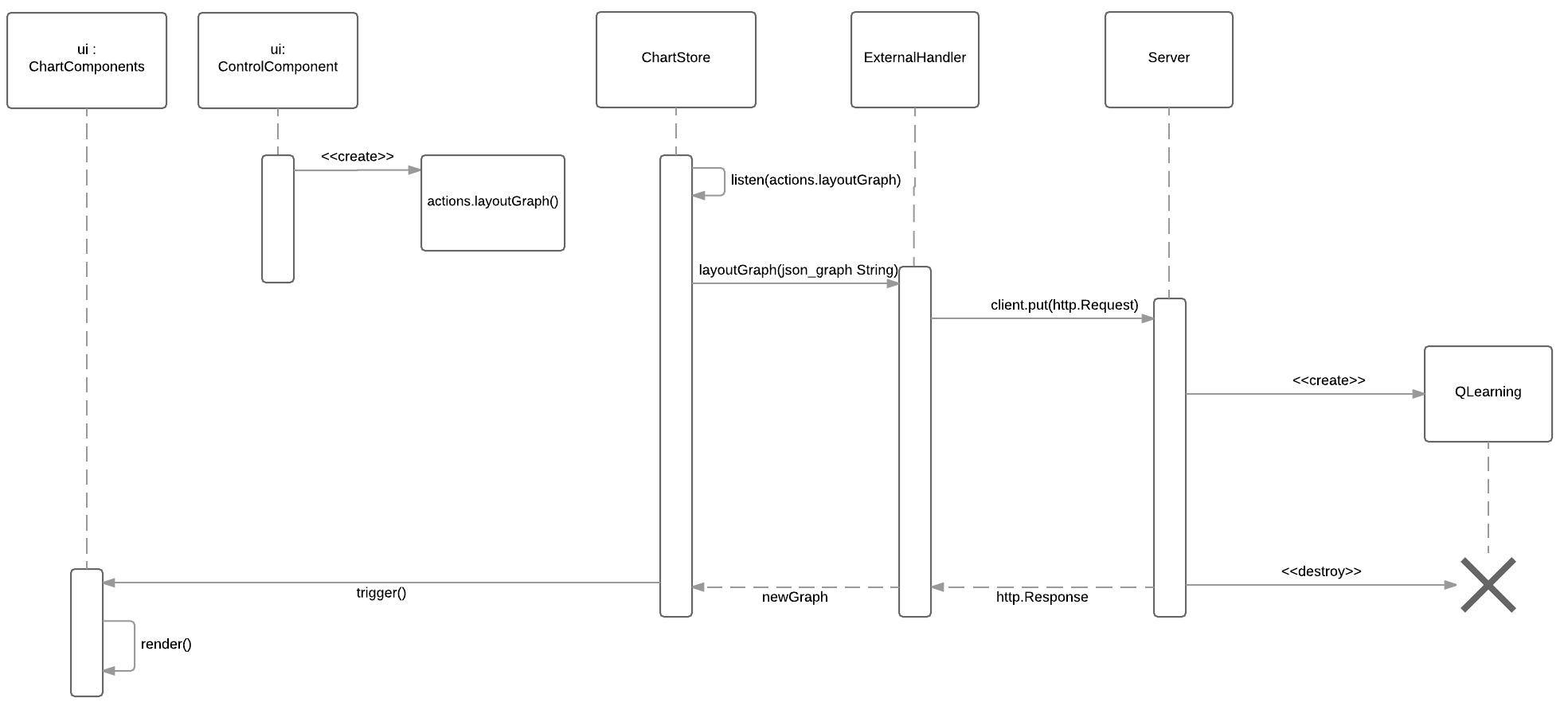
Back-End Class Diagram



Front-End Class Diagram



wFlux Sequence Diagram



Layout Graph Sequence Diagram

### 

### 

### **Section 6: Design Trade-Offs**

The main design tradeoffs that had to be considered were choices of languages, especially in regards to the server and the machine learning algorithm. For the former, Go was originally the top choice, as it is the most common server language used at Workiva. However, since Workiva has no open source Go libraries, the project would be unable to leverage any of the existing infrastructure. This made it very likely that a large portion of the server, regardless of language, would have to be reimplemented once the project was made Workiva internal. In the end, Dart was chosen as the server-side language for the following reasons. It limited the number of languages used, meaning that future collaborators would potentially have a shorter learning curve. It had easy to use server side libraries and api’s for handling HTTP requests. It is likely that the server will one day need to be completely rewritten in Go, or another more naturally server oriented language, but given the lightweight nature of the server we hope it will be minimal impact.

The choice of Python for the machine learning algorithm was done after extensive research into ML libraries. Python and Go were the top two choices, as both are used at Workiva for back-end processes and the group members were familiar with them. The deciding factor was the availability of machine learning libraries; Python had an open source and established library known as Pybrain while Go’s libraries are all still in their early stages. Pybrain’s website with comprehensive documentation and tutorials, was consistently among the top recommendations for machine learning libraries, and had a large supportive community. Go, on the other, simply hasn’t been around long enough to have a comparable library.

However, at the beginning of the project implementation, it was discovered that Pybrain had abruptly been deprecated, and their website became unusable. While this was a large blow to the appeal of Python, it was ultimately still the language of choice. The main group member working on the machine learning algorithm was more familiar with it than go; this meant coding in python would happen faster, a high priority considering the end of the semester cutoff.

### **Section 7: Software Development Life Cycle Model**

While the SDLC method varies a bit from team to team at Workiva, in general teams practice a form of scrumban. Teams follow two week sprints, which begin by committing to a body of work to complete, often with a key result or deliverable, and end with a review attended by the team, product owner(s), and stakeholders. Standups are a daily occurrence, and ticket grooming/work estimation occur at least once a sprint. Tickets are tracked using Jira, and are organized via epics, components, and the deliverables the work contributes to.

Wherever possible, the team worked to follow the norms of the Workiva development cycle, however, some of the circumstances of the project required modifications to be made to the process.

* The class and work schedule of the team made daily standups impractical. Instead, “standups” occurred twice a week at a combination of a status meeting and joint work period. In addition to these meetings, 2-4 hour joint work session occurred Thursday afternoon in the Software Factory space. Outside of these periods, most work was done independently and in separate locations.
* Workiva’s development cycle is designed for teams of 4 - 15 full time developers. Since this project consisted of two part time developers, it took time and experimentation to determine optimal sprint length and number of deliverables that could be committed to. This process was further complicated by shifting school work load, which made it difficult to anticipate the amount of time that could be dedicated at any given time when looking more than a week out. Ultimately the team adjusted to these challenges by choosing a shorter sprint length of one week with smaller commitments. School schedules were fairly known and stable a week ahead of time, and it allowed the team to hold themselves accountable to small incremental improvements, instead of large pieces of work that were easily derailed.
* Since the team’s work hours were not necessarily within standard business hours, and the busy schedule of our primary contact at Workiva, it was very difficult to find time for a face to face review. The team considered doing recorded or written reviews, but the short sprint lengths would’ve made these updates very frequent and too granular to be of use to the stakeholders, and the format wouldn’t have encouraged the feedback that would’ve been useful to the team. The team was unable to find a way to make a useful structured review process, and relied on ad hoc conversations and messages where convenient to the stakeholder and team to provide updates. While we still fail to see a good solution for all parties, we were dissatisfied with the level of communication we provided to our stakeholders,
* Because the project lacked a dedicated product owner and user experience designer that is common for many Workiva teams, the team members were required to fulfill more roles than simply being developers. Normally a product owner and delivery manager prioritize work for each sprint, and UX designers develop and approve specs prior to developers beginning work. Since all of these roles were fulfilled by two people simultaneously, there was significantly less structure than expected in Workiva’s brand of scrumban.

In the end, attempting to follow a Workiva-like SDLC had the benefits of a common process and understanding between the development team and our stakeholders, but the modifications that had to be made to fit that process to the realities of the Software Factory project somewhat marred the purpose and spirit of the system. In retrospect, it may have been more effective to commit fully to a pure version of kanban despite its differences than to partially commit to the Workiva method.