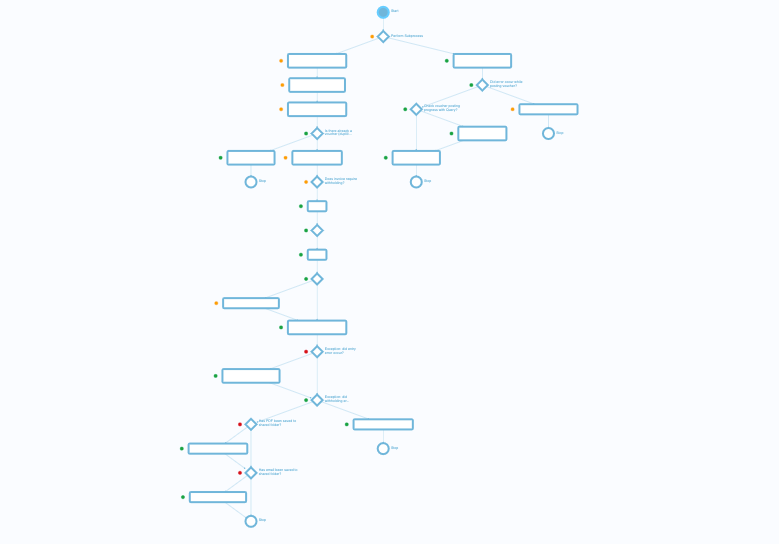
Graph Co-edit Challenge

## **Background**

Mimica automates repetitive computer-based tasks through human observation. Our software records clicks and keystrokes as users complete a task and outputs a step-by-step description of the task. The description takes the form of a flowchart. The flowchart is a graph made of nodes where each node corresponds to a step in the task.

Our web app allows users to edit the graph. For instance, users can add and remove nodes in the graph.



*A Mimica flowchart, where each node corresponds to a step in the task.*

## 

## **Challenge: Co-edit Framework**

Your aim for this challenge is to build a co-edit framework such that multiple users can edit the same graph at the same time. Users should be able to view edits performed by other users in real-time. In particular, for this challenge, you should implement logic to avoid conflicting database updates.

Consider users Anna and Bob editing the same graph G.

If Anna makes a graph edit she will have a new graph Ga on her client-side; her edits will result in a request being sent to the server to update the database to Ga

At that time Bob still sees graph G. If Bob makes a graph edit he will have a new graph Gb. Bob’s edit will result in a request being sent to the server to update the database to Gb.

If Ga and Gb conflict Bob’s request should be rejected.

Your aim for this challenge is to write logic to ensure that such conflicting database update requests are correctly rejected.

In order to minimize inconveniences for users, we should reject as few edits as possible. Hence if Ga and Gb include modifications to different sections of the graph both updates should be accepted. We should only reject an update if it is in direct conflict with a previous one.

In addition to this logic, you should write unit tests to ensure that your logic works as expected. Both the logic and its accompanying unit tests are required for this challenge.

Your work should be exclusively in the backend.

### 

### We have provided you with the following:

* a very simple web application and much of the code for the co-edit framework
* a WebSocket library ([socket.io](https://socket.io/docs/v4/)); you do not need prior knowledge of WebSockets or socket.io to complete the challenge
* all the code to establish the hand-shake between the frontend and backend
* the frontend code; you are not expected to write any frontend code for this challenge
* the database schema

## 

## **Data Structure**

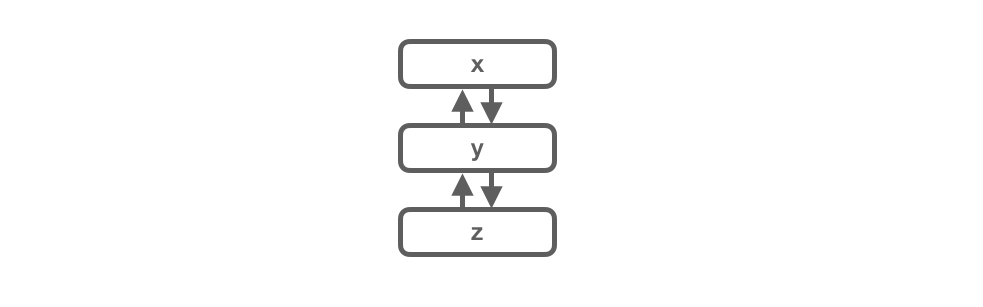
We kept the data structure as simple as possible. The only data object is the Node class.

You can find the Node data schema in BackEnd/src/components/node/node.model.ts.

Nodes are arranged as a doubly-linked list; each node has a pointer to one previous node and to one next node (either pointer can be null). Note that this is a simplification over Mimica flowcharts where each node can have pointers to multiple previous and next nodes; to keep this challenge simple we limited previous and next node pointers to a single node.

Nodes have three properties:

* id: the ObjectId of the current node
* prev: the ObjectId of the previous node
* next: the ObjectId of the next node

Below is a schematic representation of a graph with three nodes. Each node has a pointer to its previous and next node, forming a doubly-linked list.

## 

## Graph Edits

For this challenge, only two graph edits are possible: adding a node and removing a node. You should implement logic to perform these operations.

### Adding a Node

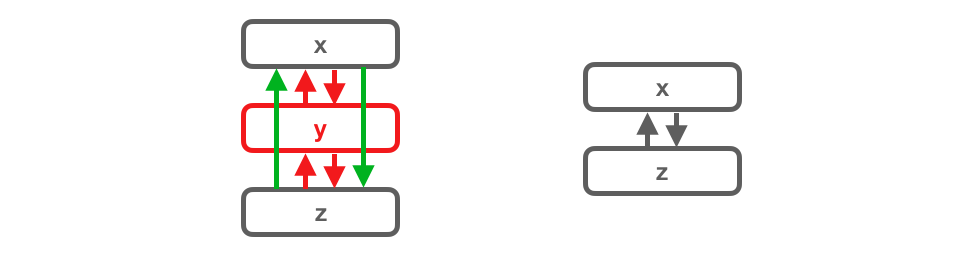
The following operations should be performed when adding node y between nodes x and z:

* create new node y
* set y.prev = x
* set y.next = z
* set x.next = y
* set z.prev = y

### **Removing a Node**

The following operations should be performed when removing node y between nodes x and z:

* delete node y
* set x.next = z
* set z.prev = x



You should implement logic to repoint prev and next pointers, create and delete node objects necessary for these edits.

You can place this logic in two existing (currently empty) functions:

* addNodeExclusive()and
* removeNodeExclusive() (in BackEnd/src/handlers/node.ts)

You can create any helper functions or files you see necessary.

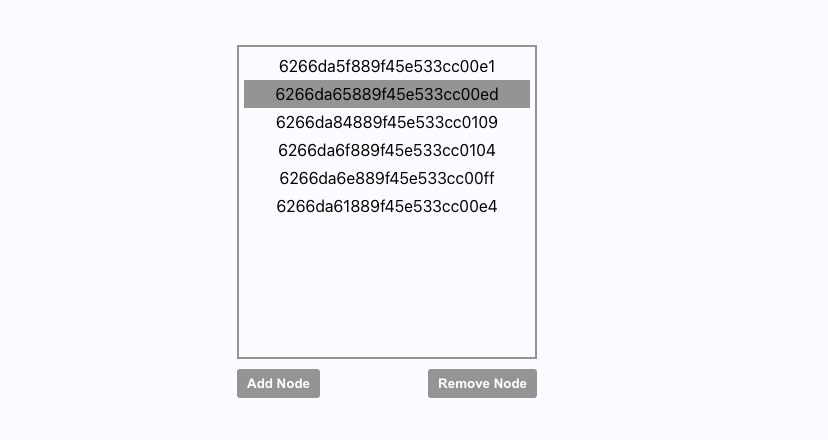
## Frontend

We provide the entire frontend of the web app.

The frontend consists of a simple display of a list of node ids.

There are two buttons, one to add a node after the selected node and one to remove the selected node.

You can click a node id in the list to select it.



## Requirements

* Users should be able to edit the same graph at the same time.
* Edits should not conflict with one another. Please consider all possible edge cases.
* To minimize inconveniences for users we should reject as few edits as possible. For instance, users should be able to edit different parts of the graph concurrently. We should only reject an update if it is in direct conflict with a previous one.
* You should populate the functions addNodeExclusive() and removeNodeExclusive().
* Feel free to create additional helper functions, classes or files as necessary.

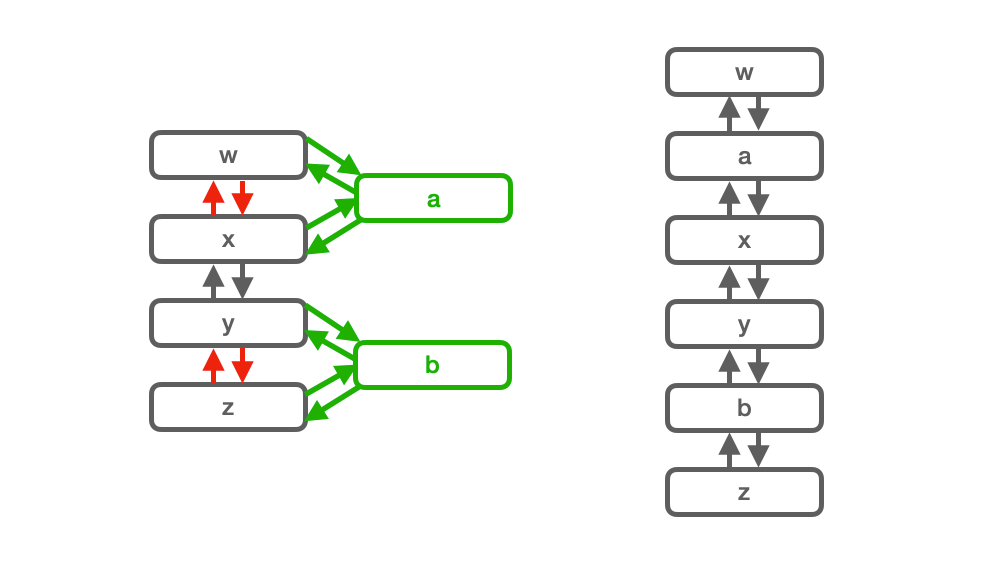
## 

## Example Edit Requests

Consider again Anna and Bob editing the same graph G.

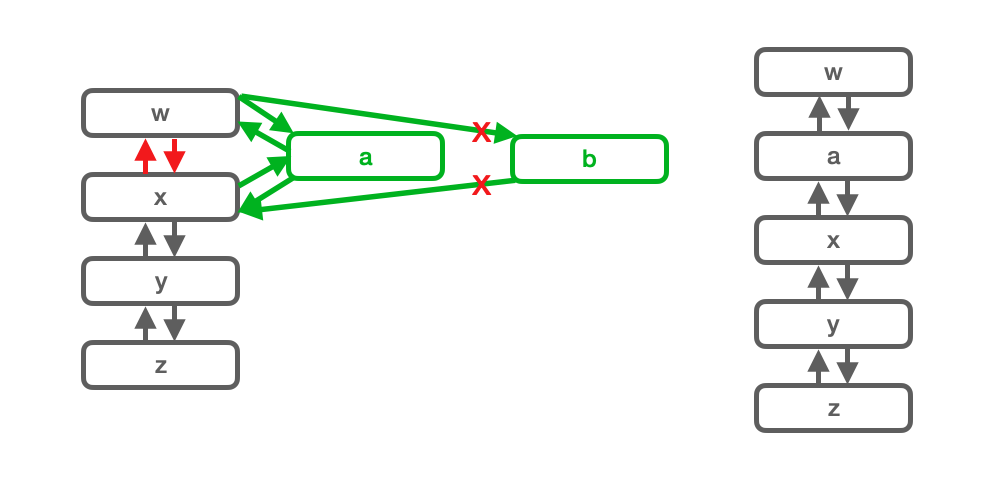
### Non-conflicting Edit Request

Anna and Bob each add a node in different locations of the graph, nodes a and b respectively. The nodes are added between two different sets of nodes. The edits are non-conflicting because none of the updated elements overlap.



### Accepted Concurrent Edit Requests

Anna and Bob both attempt to add a node between the same two nodes. In this case, the edits are conflicting because they attempt to modify the same pointers. In order to minimise error messages to users we accept both edits, applying them in the order they reach the backend.



## 

### Rejected Concurrent Edit Requests

Anna deletes a node x and Bob adds a new node after the same node x. In this case, we have to reject Bob’s edit because we deleted the reference node x. (Note that it’s not strictly necessary to reject this edit but for simplicity, for this challenge, we deem this edit conflicting and reject it.)

## 

## Unit Tests

We provide you with three unit tests to test the basic functionality of your logic. You should add any unit tests you think are necessary to cover all edge cases in your logic.

## Solution Format / Deliverables

You should populate logic for the functions addNodeExclusive() and removeNodeExclusive(). Feel free to create any supporting functions as necessary.

You should create new unit tests to cover all edge cases in your logic.

You should send a zipped folder containing the source code of your application.

When you email us your solution, please take a few minutes to answer these questions:

* About how long did you spend on it?
* If you had more time, what would you do?
* What is your feedback on the assignment?

Finally, you should be prepared to describe your solution and thought process in a follow-up meeting. We will be curious to know about any other approaches you considered and why you ultimately selected your solution. If you would like you can prepare a short written answer to these questions and send it to us with your solution.

# Technical Details

## Requirements

* node >= v16
* mongo
* mongoose

## Setup

* make sure you have the requirements above
* make sure Mongo is running
* go to graphcoedit/BackEnd and run npm i
* make a copy of .env.example as .env (you shouldn’t have to change anything in this file)
* run npm run dev and the BackEnd should start
* open a new terminal, go to graphcoedit/FrontEnd and run npm i
* make a copy of .env.example as .env (you shouldn’t have to change anything in this file)
* run npm start and a tab with FrontEnd should open

## Code

Here is the provided code for the web app.

[https://drive.google.com/file/d/1LzKkIZ8Sunu0\_u0Sc7kS-vmfBNyop994/](https://drive.google.com/file/d/1LzKkIZ8Sunu0_u0Sc7kS-vmfBNyop994/view?usp=sharing)