

LOAD DATA

Data_LOADER

SubscribeToFirebase class
-subscribes to updates from firebase ref
-transforms those updates and publishes them to any subscribers
-has an onUpdate method to register subscribers
>

```

TREE_LOADER
+ load(treeld)
+

load(treeld) implementation:
const json = await getTreeJson(treeld)
const deserializer = new
SubscribableTreeDeserializer(treeld, json) //TODO: I wish I
could deserialize the normal tree, and then apply the
subscribable behaviors onto the normal tree to transform it
into a subscribable tree
const tree = deserializer.deserialize()
return tree

//now

SUBSCRIBABLE_TREE_DESERIALIZER (json)
+ constructor(json)
+ deserialize(): ISubscribableBasicTree

const subscribableTree = SubscribableTreeLoader.load(treeld)
const firebaseTreesRef = firebase.database().ref('trees/') // rather this would be injected
from the inversify.config object graph
const treeRef = firebaseTreesRef.child(treeld)
const contentIdRef = firebaseTreesRef.child('contentId')
const parentIdRef = firebaseTreesRef.child('parentId')
const childrenRef = firebaseTreesRef.child('children')

const contentIdSaver = new FirebaseSaver({contentIdRef}) //TODO: avoid new. Somehow
can we inject this or use a factory or something?
const parentIdSaver = new FirebaseSaver({parentIdRef}) //TODO: avoid new. Somehow can
we inject this or use a factory or something?
const childrenSaver = new FirebaseSaver({childrenRef}) //TODO: avoid new. Somehow can
we inject this or use a factory or something?

const contentIdSyncerToDB = new SyncToDB({saveUpdatesToDBFunction:
contentIdSaver.save})
const parentIdSyncerToDB = new SyncToDB({saveUpdatesToDBFunction:
parentIdSaver.save})
const childrenSyncerToDB = new SyncToDB({saveUpdatesToDBFunction: childrenSaver.save})

const dbSubscriberToTree = new DBSubscriberToTree(subscribableTree,
contentIdSyncerToDB, parentIdSyncerToDB, childrenSyncerToDB)
dbSubscriberToTree.subscribe()

const dbSubscriberFromTree = new DBSubscriberFromTree(subscribableTree,
```

```

Route '/', component: BranchesAppComponent

BranchesAppComponent.ts
async created() {
  this.init()
}
...
methods: {
  init() {
const app = myContainer.get<IBranchesApp>(TYPES.IBranchesApp) // will inject
the correct databases, localStorage, localStorageRetrievers etc.
app.setURL(getWindowURL())
app.init()

BranchesApp.ts
init() {
  this.datastore.init() // DataStore has LocalStorageHandler already injected into it
  this.ui_bridges.subscribe(this.datastore)
  this.getContentItem
}

UIBridges.ts
TreeUIBridge
ContentItemUIBridge

subscribe(subscribable: ISubscribable)
{
  subscribable.onUpdate(this.handleDataUpdate)
}
handleDataUpdate((dataUpdate: IDataUpdate)) {
  switch (dataUpdate.datatype) {
    case DATA_TYPES.TREES:
      treeUIBridge.handleDataUpdate((dataUpdate))
    case DATA_TYPES.TREE_USER_DATA:
      treeUserDataUIBridge.handleDataUpdate((dataUpdate))
  }
}

TreeUIBridge.ts {
  handleDataUpdate((dataUpdate: IDataUpdate)) {
    const sigmaNodeId = DATA_SIGMA_MAP.getFromTreeId(dataUpdate.treeld)
    sigmaHandler.update({sigmaNodeId: dataUpdate.treeld, updateType:
DATA_TYPES.TREES, newTreeData: dataUpdate.val})
  }
}
TreeUserDataUIBridge.ts {
  handleDataUpdate((dataUpdate: IDataUpdate)) {
    const sigmaNodeId = DATA_SIGMA_MAP.getFromTreeId(dataUpdate.treeld)
    sigmaHandler.update({sigmaNodeId: dataUpdate.treeld, updateType:
DATA_TYPES.TREE_USER_DATA, newTreeUserData: dataUpdate.val})
  }
}

ContentItemUserDataUIBridge.ts {
  handleDataUpdate((dataUpdate: IDataUpdate)) {
    const sigmaNodeIds = DATA_SIGMA_MAP.getFromContentId(dataUpdate.contentId)
    sigmaNodeIds.forEach(sigmaNodeId => {
      sigmaHandler.update({sigmaNodeId: dataUpdate.treeld, updateType:
DATA_TYPES.TREE_USER_DATA, newTreeUserData: dataUpdate.val})
    })
  }
}

DataStore.ts() {
  graph = (nodes: [], edges: []) /
```


