Redux Saga Introduction

August 2018

What is not Saga?


FIONA STAPLES BRIAN K. VAUGHAN
Saga
CHAPTER
FORTY
SEVEN
FS '17
THE FUTURE OF COMICS
25
EST. 1992

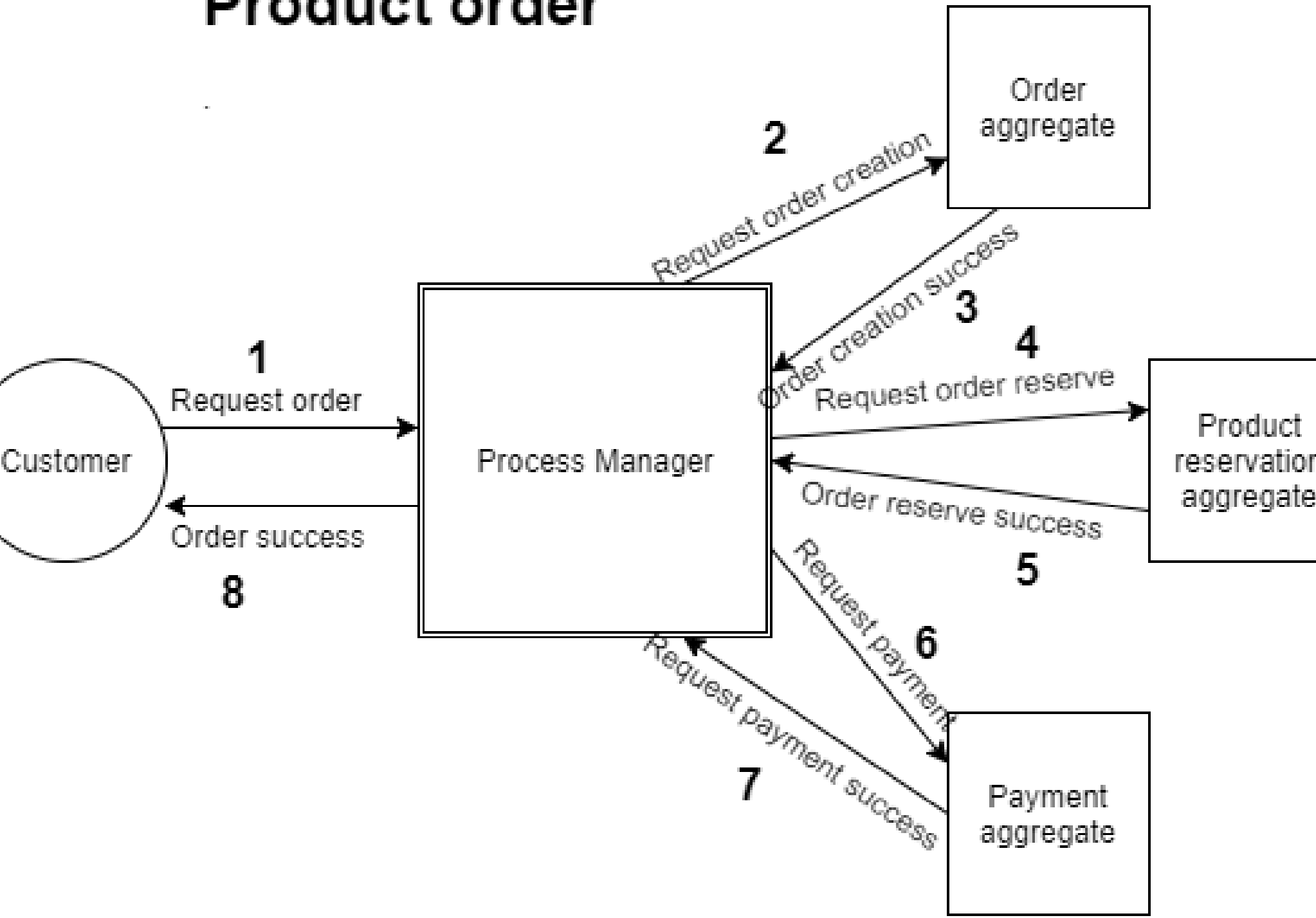
Saga is a process manager for complex systems.

Sagas is a Redux middleware that makes handling those cases easier, and more pure.

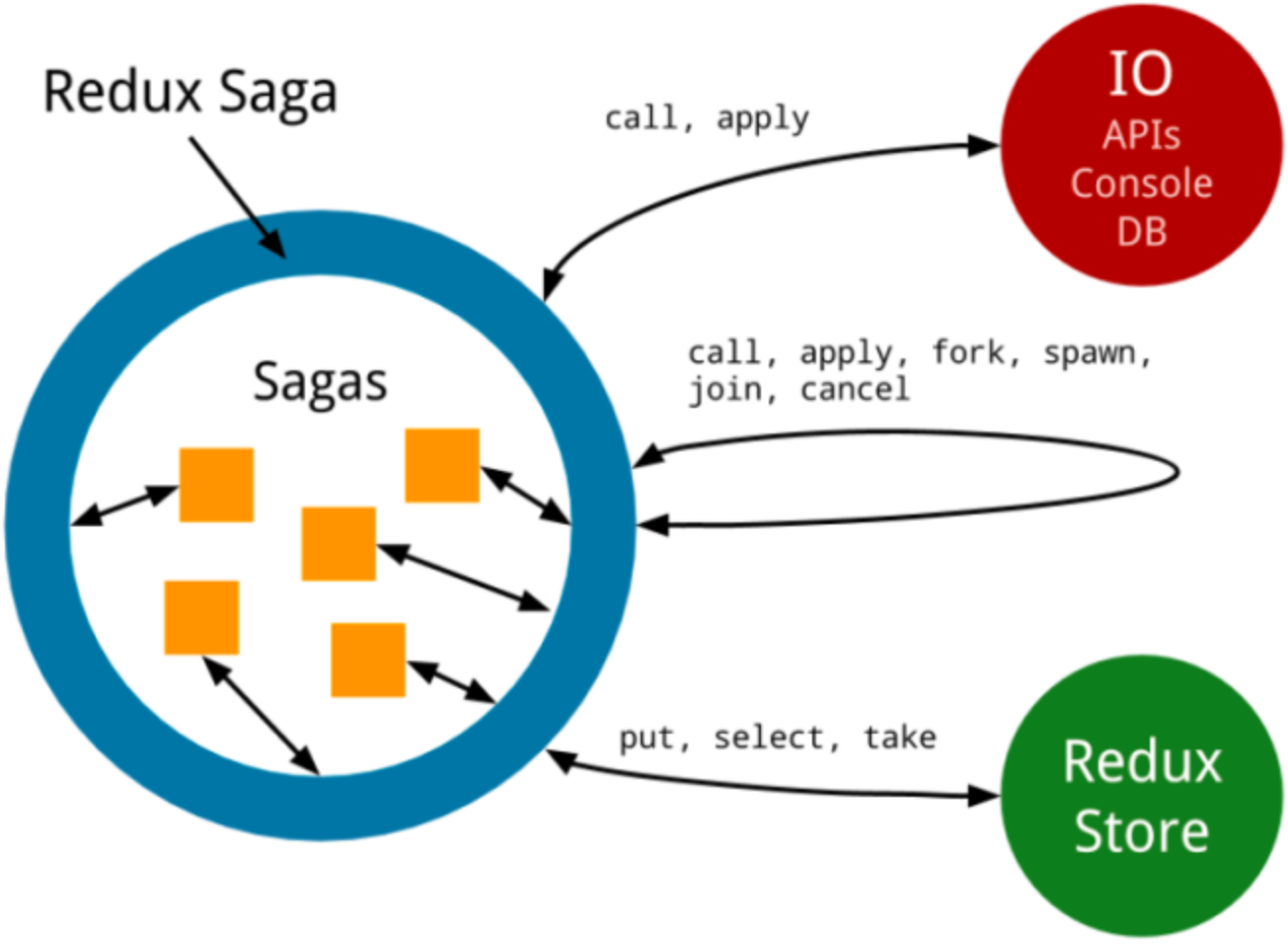
Please, don't put business logic in process managers.

Now: Next slide: User story!

Product order



Saga Hierarchy



bunu okuyan...

Thunk vs Saga

There are two common ways of dealing with side effects in Redux applications. Thunk is a function that already has everything it needs to execute. In Redux actions are defined with simple objects. And the main benefit of thunk that it allows to send a function instead. So you already able to write some logic to execute immediately and dispatch other actions.

```
export const requestOrganization = () => ({
    type: REQUEST_ORGANIZATION
});

export const successReceiveOrganization = organization =>
    type: RECEIVE_ORGANIZATION_SUCCESS,
    organization
});

export const failReceiveOrganization = error => ({
    type: RECEIVE_ORGANIZATION_FAIL,
    error
});

// will go through thunk middleware
export const fetchOrganization = id => (dispatch) => {
    dispatch(requestOrganization());

    const url = `/organization/`;

    return fetch(url)
        .then(response => response.json())
        .then(json => dispatch(successReceiveOrganization
        .catch(err => dispatch(failReceiveOrganization(er
};
```

Saga

A few short words about Saga approach. Saga is just a series of connected stories. Saga are Long Lived Transaction that can be written as a sequence of transactions that can be interleaved. All transactions in the sequence complete successfully or compensating transactions are ran to amend a partial execution. Compensation transaction are able to undo or add some info about transaction or it’s fail.

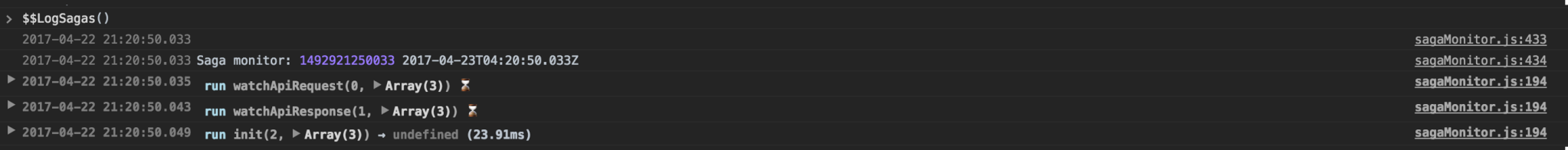
```
import { put, takeEvery } from 'redux-saga/effects';

const url = '/api/data/get';

// define fetch saga
// it contains success and fail steps for fetch scenario
export function* requestFetch() {
   // dispatch FETCH_DATA action
   yield put({ type: 'FETCH_DATA' });
   // wrap our code to catch errors if something went wro
   try {
      // fetch data in async way and write it into dataJS
      const dataJSON = yield fetch(url, {'content-type':
      // parse JSON into object data
      const data = yield dataJSON.json();
      // since parse process is finished, dispatch FETCH_
      yield put({ type: 'FETCH_DATA_SUCCESS', data });
   } catch (error) {
      // if something goes wrong dispatch FETCH_DATA_FAIL
      yield put({ type: 'FETCH_DATA_FAIL', error });
   }
}

export default function* rootSaga() {
  // spawn a new requestFetch on each REQUEST_FETCH actio
  yield takeEvery('REQUEST_FETCH', requestFetch);
}
```

Pros: Testability

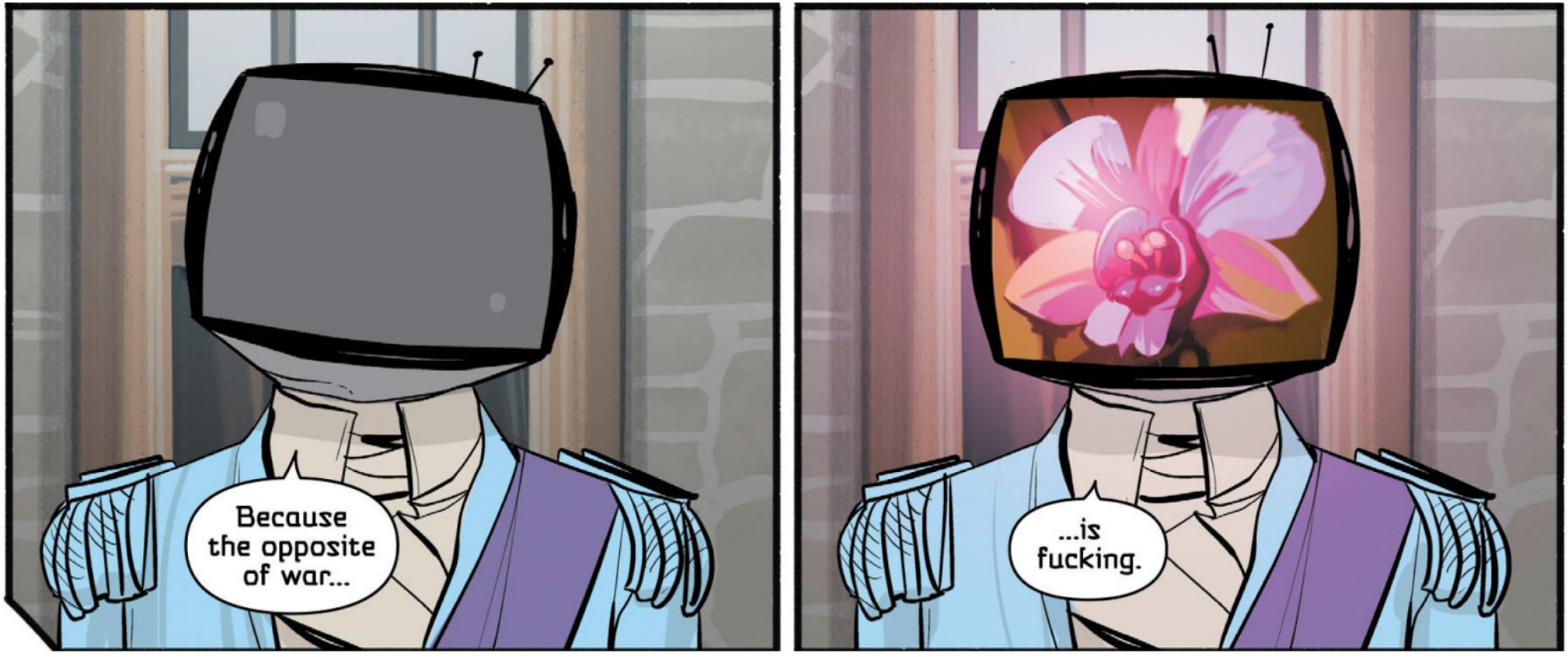
```
describe('requestFetch generator check', () => {
  const gen = requestFetch();

  it('should return an action, which will be dispatched by
      expect(gen.next()).toEqual(put({ type: 'FETCH_DATA'
  });

  it('should return fetch promise to resolve data', () =>
      expect(gen.next().value instanceof Promise).toBe(tru
  });

  it('should successfuly parse payload JSON into Object',
      expect(gen.next().value instanceof Object).toBe(true
  });

  it('should return an action, which will be dispatched by
      expect(gen.next().value.type).toEqual(put({ type: '
  });
};
```

```
2017-04-22 21:05:54.267 actionDispatched  ▼Object {type: "@@router/LOCATION_CHANGE", payload: Object} i                    sagaMonitor.js:102
                                            ▶payload: Object
                                              type: "@@router/LOCATION_CHANGE"
                                            ▶__proto__: Object
2017-04-22 21:05:54.413 effectTriggered   ▼Object {effectId: 1, root: true, parentEffectId: 0, effect: Object} i           sagaMonitor.js:65
                                            ▶effect: Object
                                              effectId: 1
                                              parentEffectId: 0
                                              root: true
                                            ▶__proto__: Object
2017-04-22 21:05:54.417 effectTriggered   ▶Object {effectId: 2, parentEffectId: 1, label: "", effect: Object}           sagaMonitor.js:65
2017-04-22 21:05:54.418 effectResolved    2 ▼Object {take: function, flush: function, close: function} i                sagaMonitor.js:81
                                              ▶close: function close()
                                              ▶flush: function flush(cb)
                                              ▶take: function take(cb)
                                              ▶__proto__: Object
2017-04-22 21:05:54.419 effectTriggered   ▶Object {effectId: 3, parentEffectId: 1, label: "", effect: Object}           sagaMonitor.js:65
```

bunu okuyan...


$$LogSagas()
2017-04-22 21:20:50.033 sagaMonitor.js:433
2017-04-22 21:20:50.033 Saga monitor: 1492921250033 2017-04-23T04:20:50.033Z sagaMonitor.js:434
2017-04-22 21:20:50.035 run watchApiRequest(0, Array(3)) sagaMonitor.js:194
2017-04-22 21:20:50.043 run watchApiResponse(1, Array(3)) sagaMonitor.js:194
2017-04-22 21:20:50.049 run init(2, Array(3)) → undefined (23.91ms) sagaMonitor.js:194

bunu okuyan...



bunu okuyan...

TENK you ❤

You can contribute this documentation on GitHub.