Redux Saga Cheat Sheet

Examples of when to use various Redux Saga keywords and techniques:

"takeEvery"

•**Use this when:** You want to watch for EVERY time a specific redux action was dispatched.

•**Use case:** Getting / fetching a list of data from an API.

•**Example:**

```
function* watchGetUsersRequest(){    yield takeEvery(action.Types.GET_USERS_REQUEST, getUsers);}
```

"takeLatest"

•**Use this when:** There's the potential for a redux action to be dispatched multiple times in a short period and could potentially initiate the running of multiple instances of the same saga - use takeLatest to ONLY take the latest currently running saga for the associated dispatched redux action.

•**Use cases:** Creating or updating a record, or;

If you have a complex app that queries the same API endpoint from multiple components at the same time - for example if you have a navbar that displays the currently logged in user's name, but the user is viewing a 'settings' page to view their personal details meaning both the navbar and the settings page will query the same API endpoint - you'll generally want to take the latest call for that data.

•**Example:**

```
function* watchGetLoggedInUserRequest(){    yield
takeLatest(action.Types.GET_LOGGED_IN_USER_REQUEST, getLoggedInUser);}
```

Blocking saga with "take"

•**Use this when:** You want to watch for a particular redux action to be dispatched, but you don't want to listen for that same dispatched action again until the currently running saga for that action has complete. You're "blocking" the ability to watch for when that particular redux action is dispatched until the currently running saga for that redux action has complete.

•**Use case:** Deleting a user, or;

Accepting a payment. Generally you don't want to be able to accept multiple, simultaneous payments - you'd want to wait for the current transaction to complete before allowing the ability to accept another payment.

•**Example:**

```
function* watchGetLoggedInUserRequest(){    while(true){        const {userId} = yield
take(action.Types.DELETE_USER_REQUEST);        yield call(deleteUser, {userId});    }}
```

"call"

•**Use this when:** You want to call a function or a promise but want to wait for that function or promise to finish running before executing the next line of code.

•**Use case:** In conjunction with "take" and blocking saga, or;

Calling a promise within a worker saga that queries an API endpoint.

•**Examples:**

```
function* deleteUser({userId}){    try{        const result = yield call(api.deleteUser, userId);    }catch(e){
}} function* watchDeleteUserRequest(){    while(true){        const {userId} = yield
take(action.Types.DELETE_USER_REQUEST);        yield call(deleteUser, {userId});    }}
```

"put"

•**Use this when:** You want to dispatch a redux action from within a redux saga.

•**Use case:** Any time you want to update your redux state - usually after a call to an API resolves and you want to update your redux state with the resulting data from the API.

•**Examples:**

```
function* getUsers(){    try{        const result = yield call(api.getUsers);        yield
put(actions.getUsersSuccess({            users: result.data.users        }));    }catch(e){        }}
```