Member-only story

Implementing Two Factor Authentication Using Formal Methods

9 min read · 1 day ago

Irwansyah

Well, every one knows Two Factor Authentication (TFA). Most probably already implemented it in their own system. So what are the differences with this method?

It lies on using formal methods to implement the spec. Most will go from informal specification which is some kind of PRD and then go straight to UML or codes. Any unclear specification usually addressed ad hoc to the spec writer (the Product Manager) or in a formal forum like a SCRUM grooming session or sprint planning session. The result of the sprint will depends on the developer team (engineer and QA) understanding of the PRD from the conducted sessions which will be chaotic when the sprint is running. This kind of methodology has brought 80% software developers are unhapp based on 2024 survey.

So these are the problems with the current methodologies:
1. There is no rapid feedback on the result of the proposed system or algorithm design this bring more risks to the “project”
2. The quality depends on the QA team and this is a disaster since QA understanding is based on the quality of the PRD and it is impossible to test all the scenarios
3. It creates more pressure to the engineers because they have to grasp the requirements in short amount of time and then coded the solution in short amount

of time
4. It is almost impossible to verify the correctness of distributed systems manually

So what is the solution to those problems? The solution is to use formal methods. IBM has been using formal methods to develop their hardware and software and it has resulted in their system being used 24/7 non-stop in banks all over the world without crash. But we don't need to wear suit like them we can still use t-shirt and informal clothing.

The formal methods tooling that we can use is TLA+. If you want to see in a glance what TLA+ you can refer to this post. One of outstanding performance of TLA+ is on AWS case. The engineers was banging their head because of a bug in their distributed system and TLA+ succesfully help them to solve the bug.

In this article, I will use a real PRD develop by a Product Manager that shared on reddit. These are the pages of the PRD:

Adding Two-factor Authentication

Target release	End of Q3 2018
Epic	[redacted]PD-24 - Adding 2 factor authentication using Google Authenticator IN PROGRESS
Document status	UNDER REVIEW
Document owner	[redacted]
Designer	@Designer
Developers	@SDM @SDE 1, @SDE 2, @SDE 3
QA	@QA 1, @QA 2

Goals

- Offer clients and users the option to add an additional layer of security when accessing their accounts through [redacted] portals
- Ensure the integrity of the client-server connection when a user's Google Authenticator app is used
- Streamline 2FA enablement for users using the same sleek interface expected of a [redacted] product
- Develop a marketing campaign to communicate with our customers about this new feature
- Offer a clear recovery path for users that lose access to their device

Background

Two-factor authentication (2FA) has been widely adopted by every large technology platform and is a component of security that customers have come to expect from businesses. Here at [redacted] we often handle highly sensitive documents and information and it is imperative that we add this additional layer of security for customers that request it. Additionally, in the case of User Onboarding and High Risk Transactions, we should consider making 2FA a mandatory part of the sign-up process as these use cases have the highest probability for fraud.

2FA requires users to have access to an additional identifier beyond username and password. When a user enables 2FA through Google Authenticator, a personal key is created as an identifier for the device used. From there on, users with 2FA enabled will need their username, password, and device in order to log in. While this in itself serves as a layer of security, the Google Authenticator app also introduces a time variable: each code generated is only valid for 30 seconds.

Assumptions

- Customers will use 2FA when they log in through mobile, web, and desktop portals
- Users do not need a Google Account to use Google Authenticator
- All devices with Google Authenticator installed will use NTP for clock syncronization

User Types

The option to enable 2FA will be available to all users with an account. As such, no distinction will be made between different user types.

Use Cases

When reviewing the requirements below, consider that users may be logging in through our mobile app, through our website, or through our desktop portals. While implementation and UX may differ between these channels, core functionality and features should be consistent and the experience should be seamless to the end user (i.e., fully integrated as a native feature).

Requirements

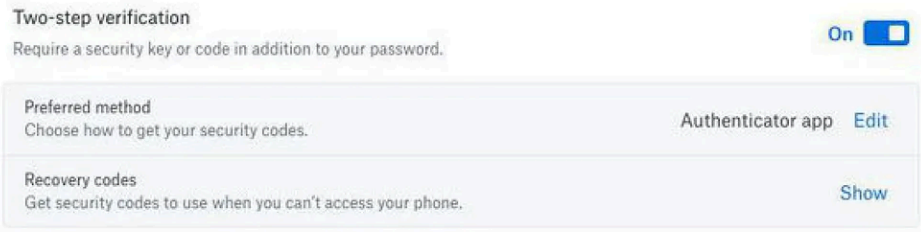
*The below requirements use the **MoSCoW** method of prioritization; from greatest to least importance: **M**ust have, **S**hould have, **C**ould have, **W**on't have*

#	Title	User Story	Importance	Notes
1	**Google Authenticator Implementation**	**Developer:** I need to design and implement framework to interact with the Google Authenticator app. The final framework should be able to generate secrets, generate codes, validate codes, and display a QR code to facilitate quicker 2FA enabling ([redacted]D-31). For security considerations, codes generated should truly be one-time use (i.e., cannot be reused even in the same time window) and the number of unsuccessful verifications per session should be limited.	Must have	[redacted]PD-25 - Google Authenticator Implementation DONE
2	**2FA Onboarding**	**User:** I'm a bit of a security novice, though I recognize its importance. It would be really helpful if your app or website walked me through the process of enabling 2FA and setting up Google Authenticator on my device.	Could have	[redacted]PD-26 - 2FA Onboarding TO DO
3	**2FA Indicator**	**User:** I recognize the importance of securing my account and since my company uses Jumio products to handle sensitive information it is only beneficial to add this extra layer of security. I expect a clear indicator of whether or not 2FA is enabled, and the option to enable 2FA if it is not currently enabled. I would expect this feature to exist in the same tab my other password and security tools.	Must have	[redacted]PD-33 - 2FA Indicator TO DO
4	**2FA Enabling**	**User:** I want the fastest and simplest way to enable 2FA. The Google Authenticator app allows me the option to use a QR code to connect my device with my accounts on other websites. Some websites required me to manually type in an identifier into the app on my phone - I must have gotten it wrong three times before it finally went through. Using the camera on my phone is much easier and faster.	Must have	[redacted]PD-31 - 2FA Enabling IN PROGRESS
5	**2FA Disabling**	**User:** I lost my phone and need to enable 2FA on a new device. I've already installed Google Authenticator on my new phone and would like to temporarily disable 2FA so that I can set it up again.	Must have	[redacted]PD-27 - 2FA Disabling TO DO
6	**2FA Login**	**User:** Now that I have enabled 2FA, I expect the login process to prompt for the code from the Google Authenticator app on my device.	Must have	[redacted]PD-30 - 2FA Login TO DO
7	**Access Recovery**	**User:** My Google account was hacked into and Google support is taking a long time to help with the Authenticator app. I need another way to recover access to my [redacted] account as a last resort! Since this is my last resort, I don't expect this to be an easy or convenient task.	Must have	[redacted]PD-28 - Access Recovery TO DO
8	**In-app Walkthrough**	**PM:** I want an engaging way to let customers know about our new 2FA feature and we have historically gotten best reach through these types of feature tutorials. This should briefly explain the feature, how to enable it, and then overlay graphics on the landing page to help users identify where the feature is located. The walkthrough should only be visible the first time a user logs in after deployment.	Should have	[redacted]PD-32 - In-app Walkthrough TO DO

User interaction and design

2FA Enabling: [redacted] goes to the Security tab on her "My Account" page after logging in. She identifies that 2FA is not enabled and clicks on the indicator to begin the enabling process. A pop-up briefly explains what 2FA is and that the Google Authenticator app will need to be installed on their mobile device. To continue with the process, [redacted] is prompted to enter her password even though she is already logged in. A QR code is displayed to be read by the Google Authenticator app. After successfully connecting the device to [redacted]'s account, she is prompted to enter the code generated in the Google Authenticator app to make sure everything is configured correctly. If unsuccessful, additional text indicates that the code did not work. If successful, the pop up displays a message confirming that 2FA is enable and the indicator in the Security tab should also reflect that 2FA is enabled.

Here is an example of what the enabled feature could look like in the Security tab:

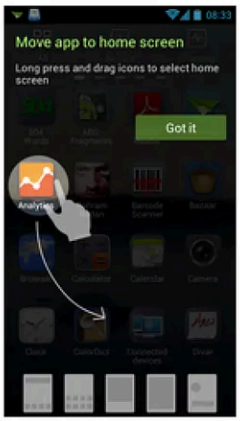


2FA disabling: In order to disable 2FA, [redacted] locates the 2FA indicator in the Security tab and clicks to begin the disabling process. She is prompted to enter her password in order to disable the feature.

Here is an example of what the disabled feature could look like in the Security tab:

Logging in with 2FA enabled: After entering her username and password, [redacted] sees a separate web page prompting her for the code in her Google Authenticator app. After opening the app, she notes that the time window for the codes displayed is closing soon and waits until the code refreshes before entering it on the web page. If validation is successful, she is forwarded to her account landing page.

An example of an "In-app" tutorial or walkthrough:



Release Criteria

- **Functionality**: All "Must-have" requirements must be met and tested.
- **Usability**: All new pages and workflows must match the visual aesthetics and design of the existing [redacted] log in and account management features.
- **Reliability/Performance**: Implementation of the Google Authenticator framework must not significantly increase the login failure rate or the client-server connection times for logging in. Workflow time for login with 2FA enabled should be < 1 min, and < 10 min for enabling 2FA as a first time user.

Questions

Below is a list of questions to be addressed as a result of this requirements document:

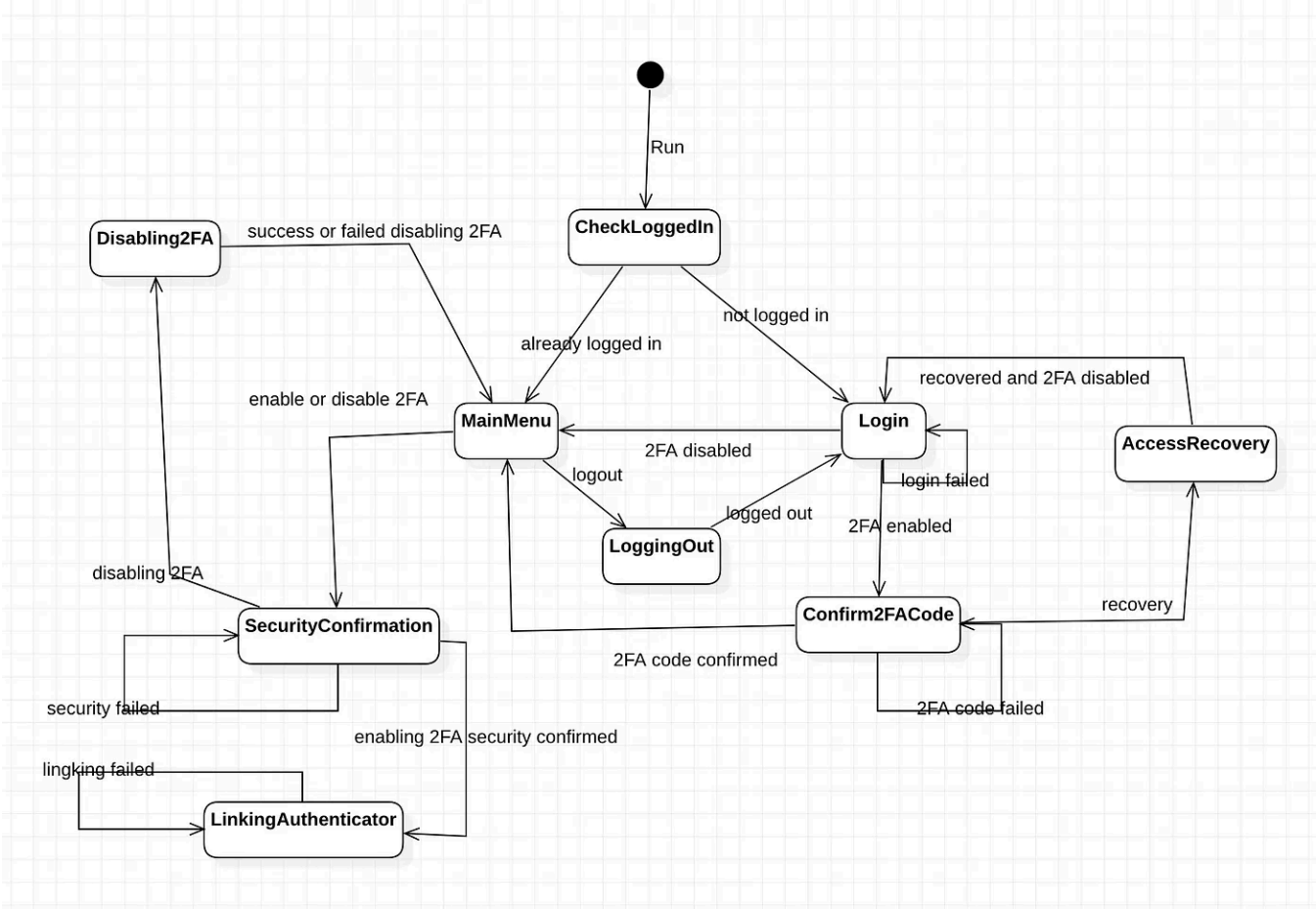
Question	Outcome

How will we communicate the feature to customers?	Product team is working on the email campaign and blog post to introduce the feature to our customers. If bandwidth allows, we should also have a web-based or app-based walkthrough that will let users know about the feature and where/how to enable it when the log in for the first time after rollout.
Will we have to support Google Authenticator app installation or issues?	
How many users do we intend to support initially?	
What kind of UX testing will be done?	

Not Doing

- Non-Google Time-based One Time Password app for users that can't or do not wish to use Google Authenticator - out of scope for initial release but we may want to consider this as an additional option in the future.
- Device recognition - some users may wish to enable 2FA initially and then only when logging in from new devices.

By reading the PRD we come up with this system design:



We refined the PRD into a statechart diagram so we can see the transitions from login to enabling the 2FA setting and seen the impact when logging out and re-loggin in again, the sytem will ask for 2FA code. Also when disabling the 2FA the statechart reflect the impact when re-logging in again. After successfull recovery the system will ask the user to relogin again and the 2FA settings will be disabled so after successful login the system will not asked for 2FA code anymore.

We will continue refining the PRD with TLA+.

```
------------------------- MODULE TwoFactor -------------------------
EXTENDS Naturals, Sequences, TLC

USERS_DATA_IN_DATABASE == [
                            uid \in {"1", "2", "3", "4"} |->
                                CASE    uid = "1" -> [id |-> "1", 2FA_enabled |
                                     [] uid = "2" -> [id |-> "2", 2FA_enabled |
                                     [] uid = "3" -> [id |-> "3", 2FA_enabled |
                                     [] uid = "4" -> [id |-> "4", 2FA_enabled |

                           ]

VALID_USERIDS == DOMAIN USERS_DATA_IN_DATABASE
```

```
ALL_REQUEST_DATA ==
                [
                    uid \in {"1", "2", "3", "4"} |->
                        CASE uid = "1" ->     [
                                                  requested_operation |-> "UN
                                                  userId |-> "1",
                                                  auth_token |-> FALSE,
                                                  security_confirmation_toker
                                              ]
                          [] uid = "2" ->     [
                                                  requested_operation |-> "UN
                                                  userId |-> "2",
                                                  auth_token |-> TRUE,
                                                  security_confirmation_toker
                                              ]

                          [] uid = "3" ->     [
                                                  requested_operation |-> "UN
                                                  userId |-> "3",
                                                  auth_token |-> FALSE,
                                                  security_confirmation_toker
                                              ]
                          [] uid = "4" ->     [
                                                  requested_operation |-> "UN
                                                  userId |-> "4",
                                                  auth_token |-> FALSE,
                                                  security_confirmation_toker
                                              ]

                ]
MAX_ITERATIONS == 100

VARIABLES
    operations_log,
    request_data,
    user_rows

vars ==
    <<
        operations_log,
        request_data,
        user_rows
    >>
```

First we include the `Naturals, Sequences, TLC` module. Then we abstracted the database using a TLA+ function that return these TLA+ structures:

```
[id |-> "1", 2FA_enabled |-> FALSE]
```

Each structure representing the 2FA settings for each user ids. The structure above is equivalent to this JavaScript object literal:

```
{
  id: "1",
  2FA_enabled: false
}
```

The `USERS_DATA_IN_DATABASE` is a TLA+ function that maps each user ids into the structure above. It is equivalent to this:

```
{
  "1" : {
          id: "1",
          2FA_enabled: false
        },
  "2" : {
          id: "2",
          2FA_enabled: true
        },
  "3" : {
          id: "3",
          2FA_enabled: false
        },
  "4" : {
          id: "4",
          2FA_enabled: true
        }
}
```

The `VALID_USERIDS` is a TLA+ set that equivalent to this JS array:

```
["1", "2", "3", "4"]
```

The `ALL_REQUEST_DATA` is combined with the `USERS_IN_DATABASE` to specify these four different scenarios:

User ID	Authentication Token	2FA Enabled
1	FALSE	FALSE
2	TRUE	TRUE
3	FALSE	FALSE
4	FALSE	TRUE

We have to specify how many iterations our spec will continue to run in `MAX_ITERATIONS` so the model checker can finished in less time (it should be running in less than a minute otherwise it is too long). What it means is, TLA+ model checker will visit all the states automatically and if we don't specify the termination (`MAX_ITERATIONS`) then we have to wait for a long time for it to finished, probably never. Below it is our variables that is required for TLA+ model checker to transition between states.

For each states in our statechart diagram we will implement it as a TLA+ operator. Inside the state's operator it will contains the preconditions and the formula to do the transitions.

```
REQUEST_DATA_FOR_TESTING == {
        request_data["1"]
\*        , request_data["2"]
\*        , request_data["3"]
        }
Init ==
    /\ operations_log = [uid \in VALID_USERIDS |-> <<>>]
    /\ request_data = ALL_REQUEST_DATA
    /\ user_rows = USERS_DATA_IN_DATABASE

Next ==
    /\ \E request \in REQUEST_DATA_FOR_TESTING:
        /\  \/ CheckAlreadyLoggedIn(request)
            \/ Login(request)
            \/ AccessRecovery(request)
            \/ MainMenu(request)
            \/ Confirm2FACode(request)
            \/ SecurityConfirmation(request)
```

```
            \/ Disabling2FA(request)
            \/ LinkingAuthenticator(request)
            \/ Logout(request)


Spec == Init /\ [][Next]_vars
```

Every TLA+ spec must have an `Init` and `Next` operator. In `Init` we initialize the `VARIABLES` with the initial values. In our case we initialize the `operations_log` with an empty TLA+ sequence. It is equivalent to an empty array in a programming language. We assign the `request_data` with the contents of the `ALL_REQUEST_DATA`. We need to create the variable because we have to change the contents of it. The last variable is the `user_rows` which will emulate our database rows.

In our `Next` we iterate the `REQUEST_DATA_FOR_TESTING` set and for each item we hand over it to TLA+ model checker and it is up to the model checker which states to visit.

The `Spec` contains a temporal operator `[][Next]_vars` it is just to specify our spec allows stuttering. A stuttering is required since TLA+ is built based on physics where in physics the system can transition without any changed in the variables but it is just to specify the other variables in the universe is changing.

Now, let's see the specification for each node in the statechart:

```
\******** State machine implementation

CheckAlreadyLoggedIn(request) ==
    /\ request.requested_operation = "UNSET"
    /\ IF request.auth_token THEN
        RedirectTo("MainMenu", request)
       ELSE
        RedirectTo("Login", request)
    /\ LogOperation("CheckAlreadyLoggedIn", request, request_data'[request.user
    /\ UNCHANGED <<user_rows>>


IsLoginSuccess(request) == RandomElement({FALSE, TRUE})

Login(request) ==
    /\ request.requested_operation = "Login"
    /\ ~request.auth_token
    /\ request.userId \in VALID_USERIDS
```

```
    /\ IF IsLoginSuccess(request) THEN
        IF user_rows[request.userId].2FA_enabled THEN
                RedirectTo("Confirm2FACode", request)
            ELSE
                RedirectToAndAssignAuthToken("MainMenu", request)
       ELSE
        RedirectTo("Login", request)
    /\ LogOperation("Login", request, request_data'[request.userId], user_rows)
    /\ UNCHANGED user_rows


AccessRecovery(request) ==
    /\ request.requested_operation = "AccessRecovery"
    /\ ~request.auth_token
    /\ request.userId \in VALID_USERIDS
    /\ RedirectTo("Login", request)
    /\ LogOperation("AccessRecovery", request, request_data'[request.userId], u
    /\ UNCHANGED <<user_rows>>

ChooseAMenu(request) ==
        \/ RedirectTo(RandomElement({"Logout", "Enable2FA", "Disable2FA"}), rec

MainMenu(request) ==
    /\ request.auth_token
    /\ request.requested_operation = "MainMenu"
    /\ request.userId \in VALID_USERIDS
    /\ ChooseAMenu(request)
    /\ LogOperation("MainMenu", request, request_data'[request.userId], user_rc
    /\ UNCHANGED <<user_rows>>

Logout(request) ==
    /\ request.auth_token
    /\ request.requested_operation = "Logout"
    /\ RedirectToAndDiscardAuthToken("Login", request)
    /\ LogOperation("Logout", request, request_data'[request.userId], user_rows
    /\ UNCHANGED <<user_rows>>

Confirm2FACode(request) ==
    /\ ~request.auth_token
    /\ request.requested_operation = "Confirm2FACode"
    /\ IF RandomElement({"MainMenu", "AccessRecovery"}) = "MainMenu" THEN
        RedirectToAndAssignAuthToken("MainMenu", request)
       ELSE
        RedirectTo("AccessRecovery", request)
    /\ LogOperation("Confirm2FACode", request, request_data'[request.userId], u
    /\ UNCHANGED <<user_rows>>

SecurityConfirmation(request) ==
    /\ request.auth_token
    /\ (
            \/ request.requested_operation = "Enable2FA"
            \/ request.requested_operation = "Disable2FA"
        )
```

```
    /\ IF request.requested_operation = "Enable2FA" /\ ~2FA_IsEnabled(request)
        RedirectToAndAssignSecurityConfirmationToken("LinkingAuthenticator", re
       ELSE
        IF request.requested_operation = "Disable2FA" /\ 2FA_IsEnabled(request)
            RedirectToAndAssignSecurityConfirmationToken("Disabling2FA", reques
        ELSE
            RedirectTo("MainMenu", request)
    /\ LogOperation("SecurityConfirmation", request, request_data'[request.user
    /\ UNCHANGED <<user_rows>>

Disabling2FA(request) ==
    /\ request.auth_token
    /\ request.security_confirmation_token
    /\ 2FA_IsEnabled(request)
    /\ request.requested_operation = "Disabling2FA"
    /\ Disable2FASetting(request)
    /\ RedirectToAndDiscardSecurityConfirmationToken("MainMenu", request, "Logi
    /\ LogOperation("Disabling2FA", request, request_data'[request.userId],  us


LinkingAuthenticator(request) ==
    /\ request.auth_token
    /\ request.security_confirmation_token
    /\ ~2FA_IsEnabled(request)
    /\ request.requested_operation = "LinkingAuthenticator"
    /\ Enable2FASetting(request)
    /\ RedirectToAndDiscardSecurityConfirmationToken("MainMenu", request, "Conf
    /\ LogOperation("LinkingAuthenticator", request, request_data'[request.user
```

In TLA+ `/\` is a propositional logic `AND` it is equivalent to `&&` operator in JavaScript, `\/` is an `OR` , `~` is a negation. The first few `/\` specify preconditions for each state. And in every visit to each state we log the request and the response in the `operational_logs` parameter. Later on we will iterate the contents of the log to verify our invariants.

This is our utility operators that being called by each states:

```
\******** Utility operators

RedirectTo(operation_name, request) ==
    request_data' = [request_data EXCEPT ![request.userId].requested_operation

RedirectToAndAssignSecurityConfirmationToken(operation_name, request) ==
    request_data' = [request_data EXCEPT ![request.userId].requested_operation

RedirectToAndDiscardSecurityConfirmationToken(operation_name, request, before_c
```

```
    request_data' = [request_data EXCEPT ![request.userId].requested_operation

RedirectToAndAssignAuthToken(operation_name, request) ==
request_data' = [request_data EXCEPT ![request.userId].requested_operation = op

RedirectToAndDiscardAuthToken(operation_name, request) ==
    request_data' = [request_data EXCEPT ![request.userId].requested_operation


Enable2FASetting(request) ==
    user_rows' = [user_rows EXCEPT ![request.userId].2FA_enabled = TRUE]

Disable2FASetting(request) ==
    user_rows' = [user_rows EXCEPT ![request.userId].2FA_enabled = FALSE]

LogOperation(operation_name, original_request, updated_request, dbuser_rows) ==
    operations_log' = [
                        operations_log EXCEPT ![original_request.userId] = Appe









                      ]
    /\ Len(operations_log'[original_request.userId]) <= MAX_ITERATIONS

2FA_IsEnabled(request) == user_rows[request.userId].2FA_enabled
```

To take advantage of the TLA+ model checker we have to write invariants to verify our spec is conform withthe invariants:

```
Confirm2FACodeRequestedProperlyAfterLoginPerUser(operations_log_per_user) ==
    \/ operations_log_per_user = <<>>
    \/ \A i \in 1..Len(operations_log_per_user):
            LET log == operations_log_per_user[i] IN
                IF log.operation = "Login" /\ log.updated_request.auth_token TH
                    IF log.dbuser_rows[log.original_request.userId].2FA_enabled
                        log.updated_request.requested_operation = "Confirm2FACo
                    ELSE
                        TRUE
                ELSE
                    TRUE

Confirm2FACodeRequestedProperlyAfterLogin ==
    \A request \in REQUEST_DATA_FOR_TESTING:
            /\ Confirm2FACodeRequestedProperlyAfterLoginPerUser(operations_log[
```

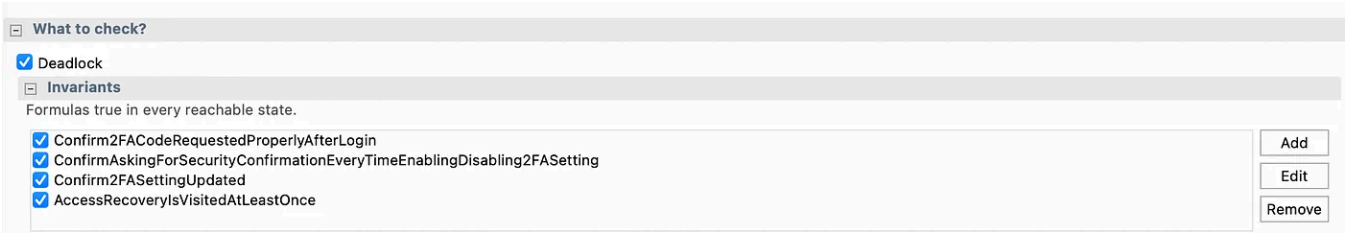
```
ConfirmAskingForSecurityConfirmationEveryTimeEnablingDisabling2FASettingPerUser
    \/ operations_log_per_user = <<>>
    \/ \A i \in 1..Len(operations_log_per_user):
            LET log == operations_log_per_user[i] IN
                IF log.original_request.requested_operation = "Enable2FA" \/ lo
                    log.operation = "SecurityConfirmation"
                ELSE
                    TRUE

ConfirmAskingForSecurityConfirmationEveryTimeEnablingDisabling2FASetting ==
    \A request \in REQUEST_DATA_FOR_TESTING:
        ConfirmAskingForSecurityConfirmationEveryTimeEnablingDisabling2FASettir

Confirm2FASettingUpdatedPerUser(operations_log_per_user) ==
    \/ operations_log_per_user = <<>>
    \/ \A i \in 1..Len(operations_log_per_user):
            LET log == operations_log_per_user[i] IN
                IF log.operation = "LinkingAuthenticator" /\ log.updated_reques
                    log.dbuser_rows[log.original_request.userId].2FA_enabled =
                ELSE
                    IF log.operation = "Disabling2FA" /\ log.updated_request.re
                        log.dbuser_rows[log.original_request.userId].2FA_enable
                    ELSE
                        TRUE



Confirm2FASettingUpdated ==
    \A request \in REQUEST_DATA_FOR_TESTING:
        Confirm2FASettingUpdatedPerUser(operations_log[request.userId])


AccessRecoveryIsVisitedAtLeastOncePerUser(operations_log_per_user) ==
    \/ operations_log_per_user = <<>>
    \/ Len(operations_log_per_user) <= MAX_ITERATIONS - 1
    \/ \E i \in 1..Len(operations_log_per_user):
            LET log == operations_log_per_user[i] IN
                log.operation = "AccessRecovery"

AccessRecoveryIsVisitedAtLeastOnce ==
    \A request \in REQUEST_DATA_FOR_TESTING:
        AccessRecoveryIsVisitedAtLeastOncePerUser(operations_log[request.userId
```

These are the invariants that we specify:

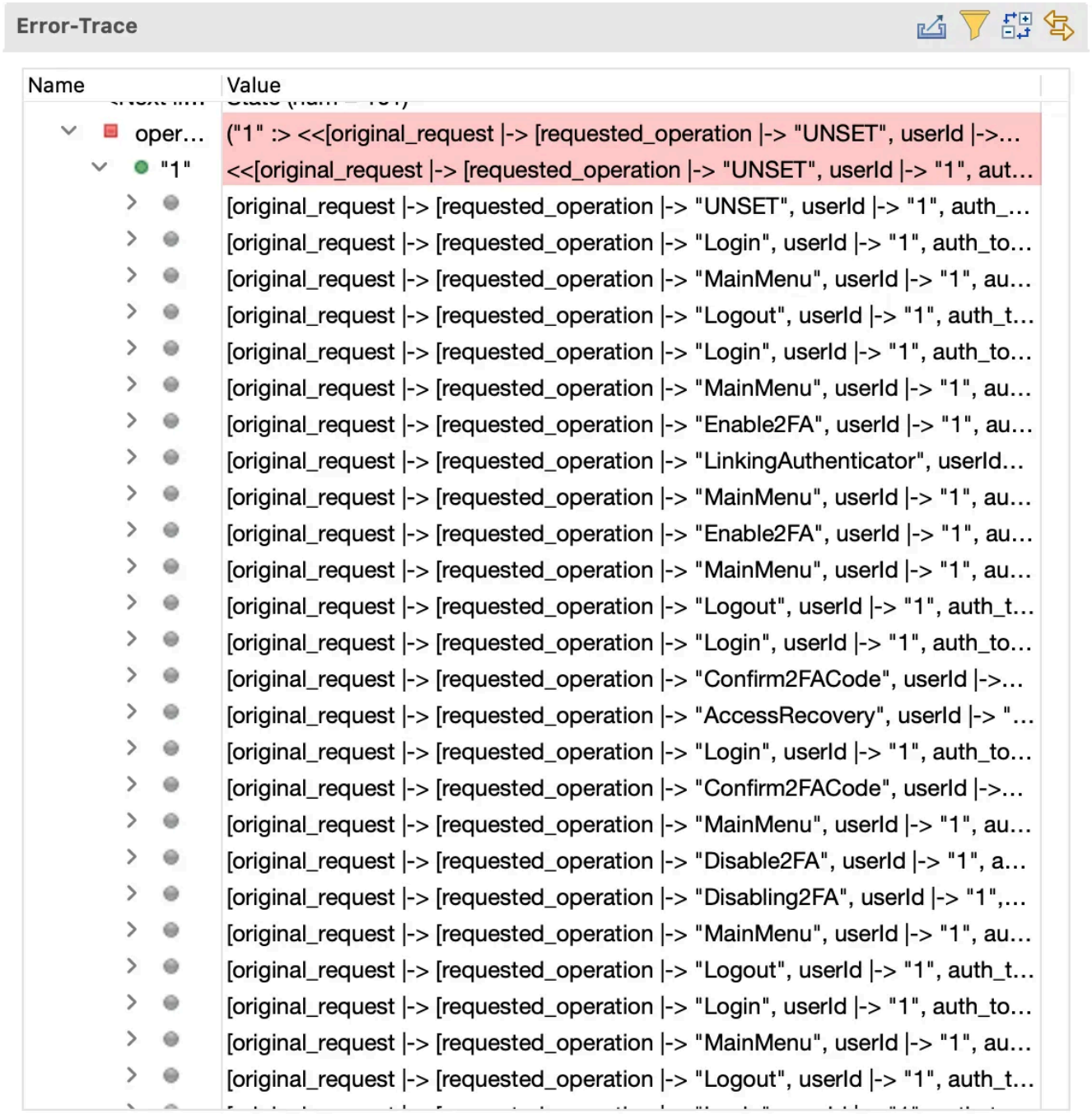
1. Make sure every time the user is logged in the system asking for the 2FA code (if the 2FA setting is enabled for the user)

2. Make sure every time the user is enabling or disabling the 2FA setting, the system will asked for security confirmation
3. Make sure the system updates the 2FA setting for each enabling or disabling 2FA
4. Make sure the access recovery is working as specified

Then we can ask TLA+ model checker to verify our spec and make sure there is no deadlock and it passed the invariants checking.



Beside using invariants we can also check the trace manually to see how TLA+ model checker verify our spec. Below is an example of the traces:



You can see from the trace the model checker automatically visits all the states.

After our spec passed the invariants checking then we can continue to implement our spec into our chosen programming language. I will wrote another article for the implementation later on.

Formal Verification | Tla Plus | Software Development | Software Engineering | Software Testing