Features Flags

MYMOID

Juan J. Barroso Giménez - Jr FrontEnd Developer - MYMOID

Feature Flags are also referred to as
Feature Toggles, Feature Bits, or Feature Flippers.

Feature Flags allows you to toggle a feature on or off at any point of time, even after you have deployed your code.

Feature Flags are a powerful technique, allowing teams to modify system behaviour without changing code.

Martin Fowler

Not Too Impressed ?

BACKGROUND

Feature Branching

Developers create a branch for each feature. The branch is merged into main branch when the feature is complete

Continuous Integration

Developers merge (integrate) their code into main branch at least once a day. Each merge is verified by an automated build.

PROBLEMS

Feature Branching

Potential for severe merge conflicts, especially if your branch contains a big feature

Continuous Integration

What happens if you have a feature that will take longer to implement than a day?

OTHER PROBLEMS

Code Review

OTHER PROBLEMS

> I want to implement A/B testing

> I want to be able to disable certain features if our servers are under heavy load

> I want my app to have premium features that only paying customers can use

CASES

> Give some users a preview of a new feature and see how it’s working out for them, just like Facebook and Google do.

CASES

> Let the users decide how they like something. Say you want to see if a button will be more visible if it’s yellow or green.

Well why not code both and put them behind feature flags. Show 30% of users yellow button, 30% green button and for the rest 40% keep it as it is. Based on how your users respond choose the right color or just go with yellow because that’s what you wanted to. BTW, this is called A/B Testing.

Core value of Feature Flags

> Feature Flags decouple two very dangerous steps – Deploying and Rollout.

Deploying means when you add your feature enabling code on the server.

Rollout means when your feature can be used by the user.

Both these steps are very risky. When you deploy, your code may not work as you expect in production as production is a completely different environment.

Core properties of FF

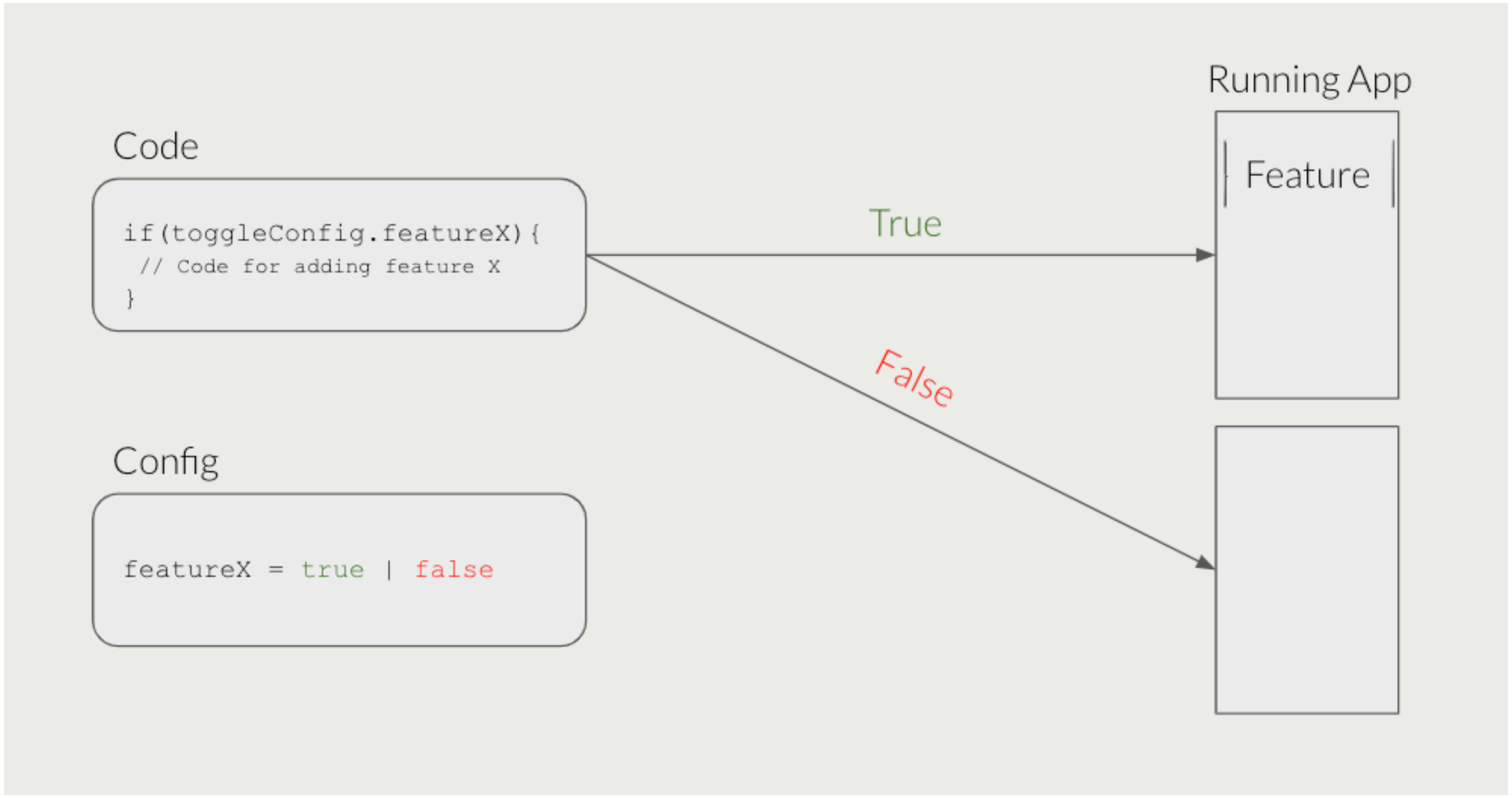
> State: a feature can be in an *on* or off *state*

> Key: a feature can be identified and used more easily with a client defined name.

Feature Toggle Categories

1. Release Toggles
2. Experiment Toggles
3. Operational Toggles
4. Permission Toggles

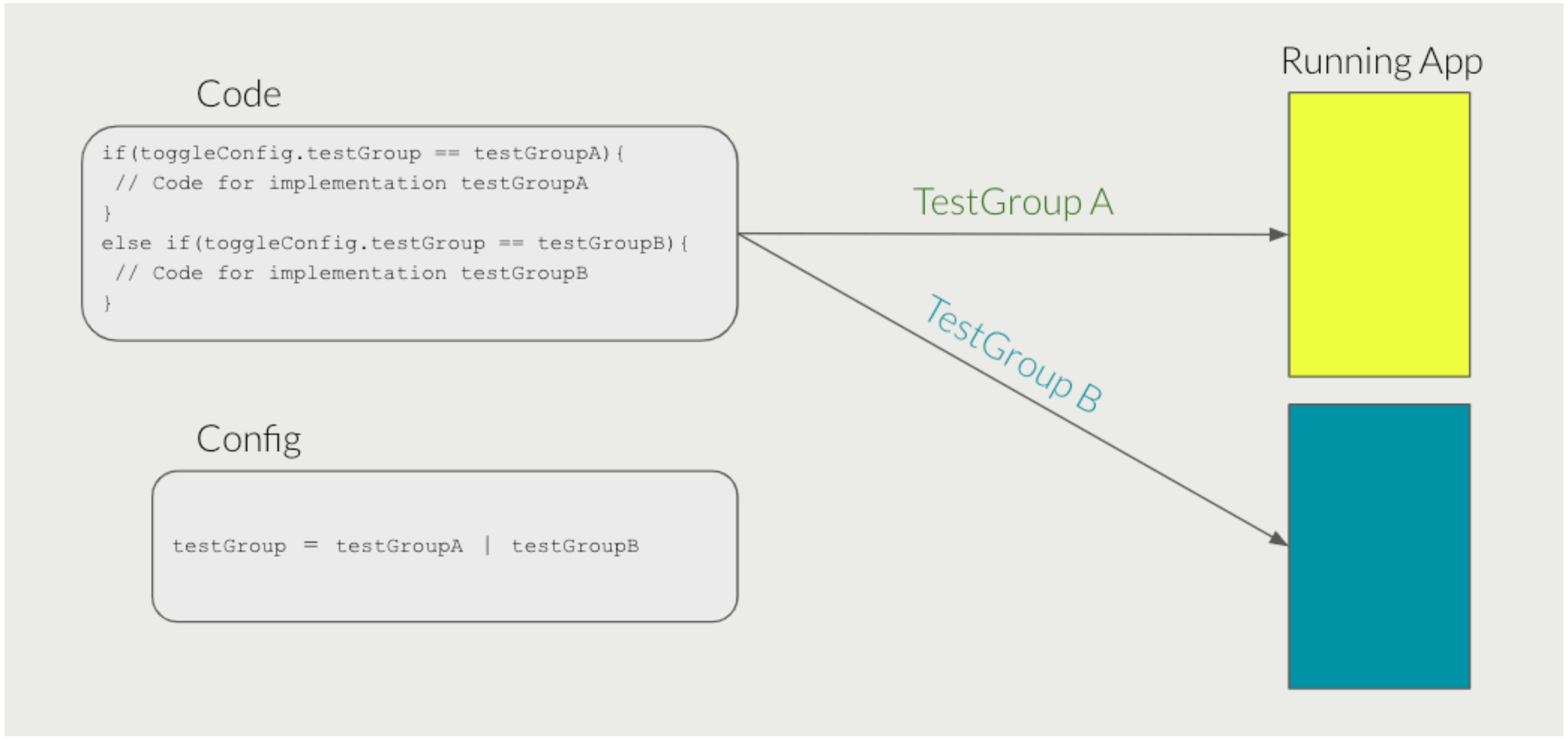
Feature Toggle Categories : Release Toggles



```
if(toggleConfig.featureX){
  // Code for adding feature X
}
```

```
featureX = true | false
```

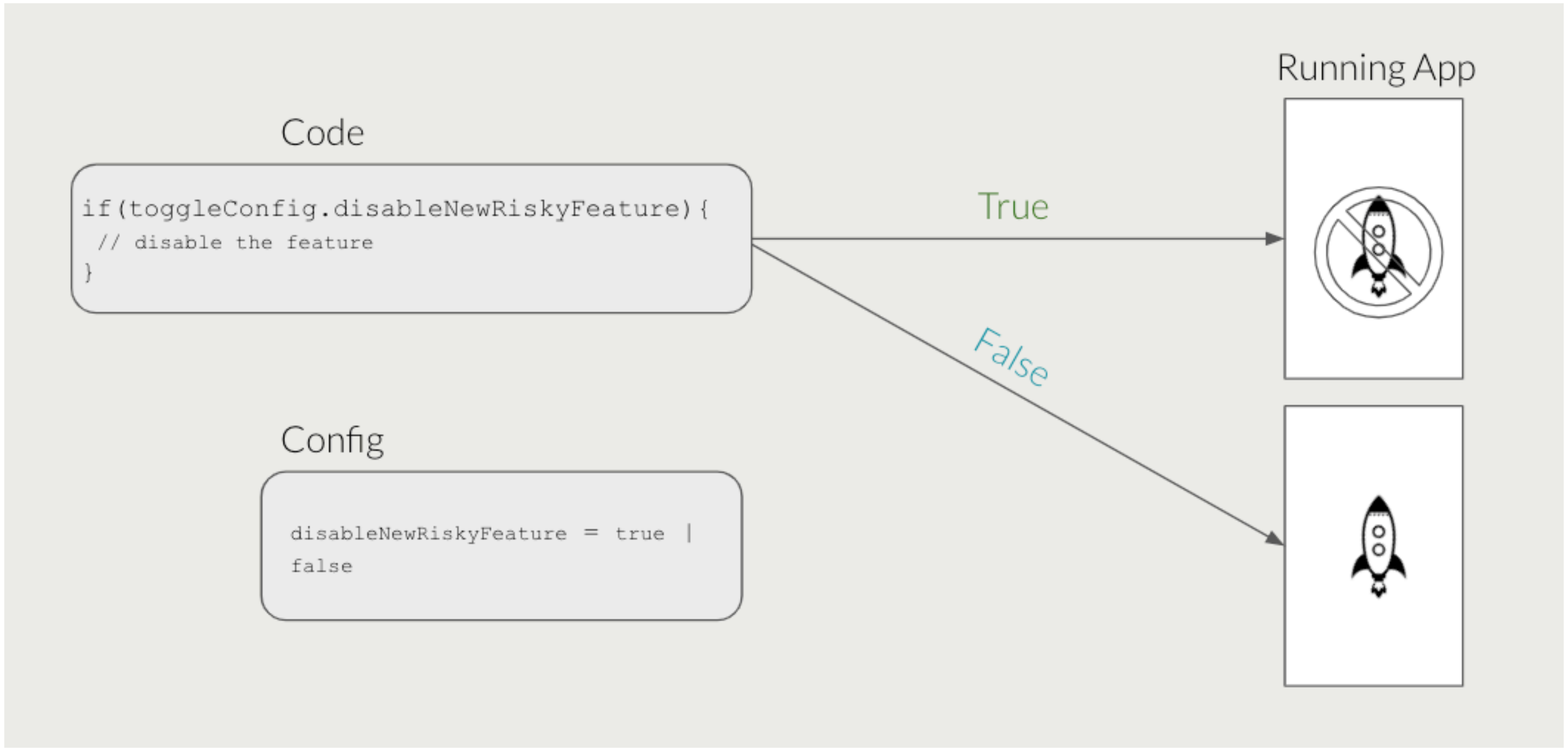
Feature Toggle Categories : Experiment Toggles



```
if(toggleConfig.testGroup == testGroupA){
 // Code for implementation testGroupA
}
else if(toggleConfig.testGroup == testGroupB){
 // Code for implementation testGroupB
}
```

```
testGroup = testGroupA | testGroupB
```

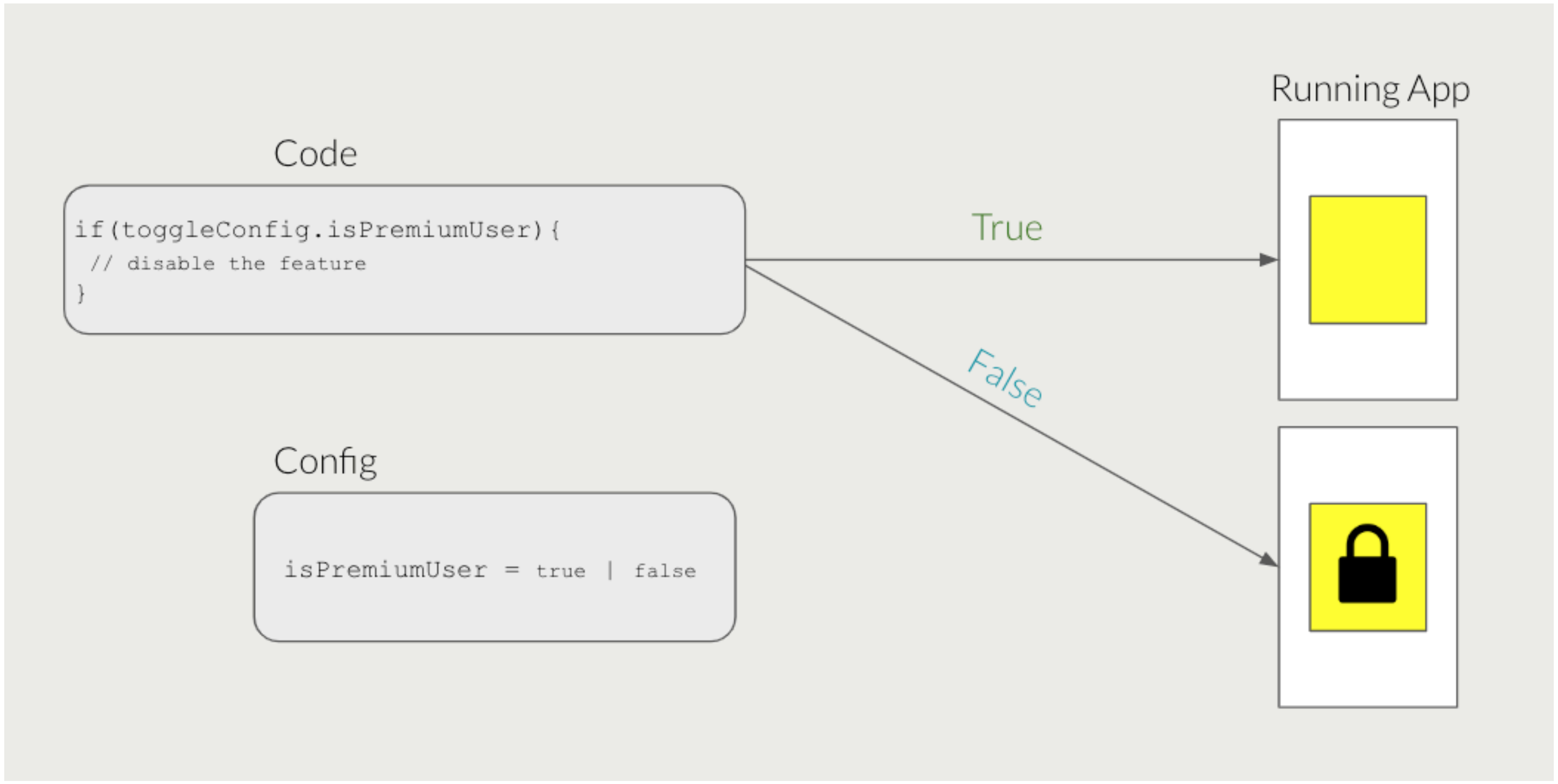
Feature Toggle Categories : Operational Toggles



```
if(toggleConfig.disableNewRiskyFeature){
  // disable the feature
}
```

```
disableNewRiskyFeature = true |
false
```

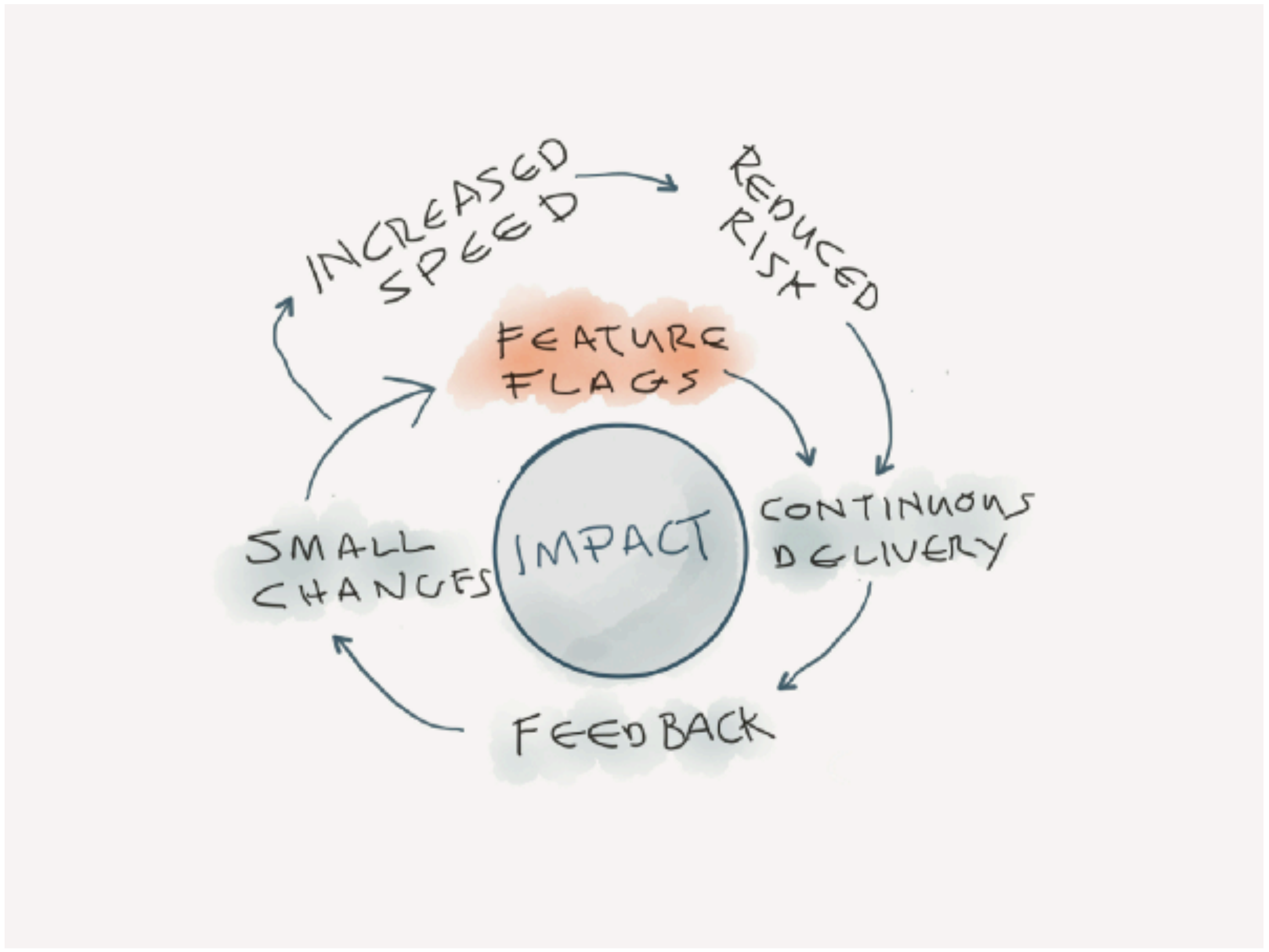
Feature Toggle Categories : Permission Toggles



```
if(toggleConfig.isPremiumUser){
  // disable the feature
}
```

```
isPremiumUser = true | false
```

Feature Flags and Service Development



Feature flags reinforce the benefits from small changes and continuous delivery, providing a flywheel to get you round the loop faster.

Top 5 Use Cases for Feature Flags

1. Deployments With Less Hassle
2. Experimentation Through A/B Testing
3. Verbose Logging for Troubleshooting
4. Always Be Ready for Continuous Integration
5. Customise Behaviour Based on User Preferences

1. Deployments With Less Hassle

Have you ever wondered how some companies release new things just after they’ve announced them at a live conference? Well, with feature flags, that’s possible

2. Experimentation Through A/B Testing

Facebook, Netflix, and other companies claim that they run experiments all the time. In fact, this is how Amazon launched their recommendations section you see on each product page.

3. Verbose Logging for Troubleshooting

It’s understandable if you don’t want to have verbose logging because disks could run out of space. No one wants to be awakened in the middle of the night with this news. But what if you could launch your code with verbose logging turned off and only enabled when you need it? Yes, with feature flags, it’s possible.

4. Always Be Ready for Continuous Integration

That’s because you can always push your code with the feature toggled off. Merge conflicts will be reduced, and it will be much easier and cheaper to fix them because everyone is pushing their changes more frequently.

5. Customise Behaviour Based on User Preferences

Feature flags can also help you enable certain things for premium users. How easy would it be to upgrade a user? Quite easy. Just turn the premium features on for that user, and you’re done.

4 tips for getting started with feature flags

Tip 1

Understand why you want to use feature flags.

Turning on features and rolling them out progressively gives our product teams more control over the experience we want to deliver.

4 tips for getting started with feature flags

Tip 2

Measure the success of your features

We prevent major errors when we turn a flag on by monitoring the logs. If we see errors, we switch the flag off, fix the problem, and then resume the rollout when ready

Atlassian

4 tips for getting started with feature flags

Tip 3

Make sure you have guiding principles in place for managing feature flags

we always delete feature flags after a feature has been rolled out to 100% of customers.

Atlassian

4 tips for getting started with feature flags

Tip 4

Create a workflow around feature flagging so there’s a clear process for how flags should be rolled out, managed, and cleaned up.

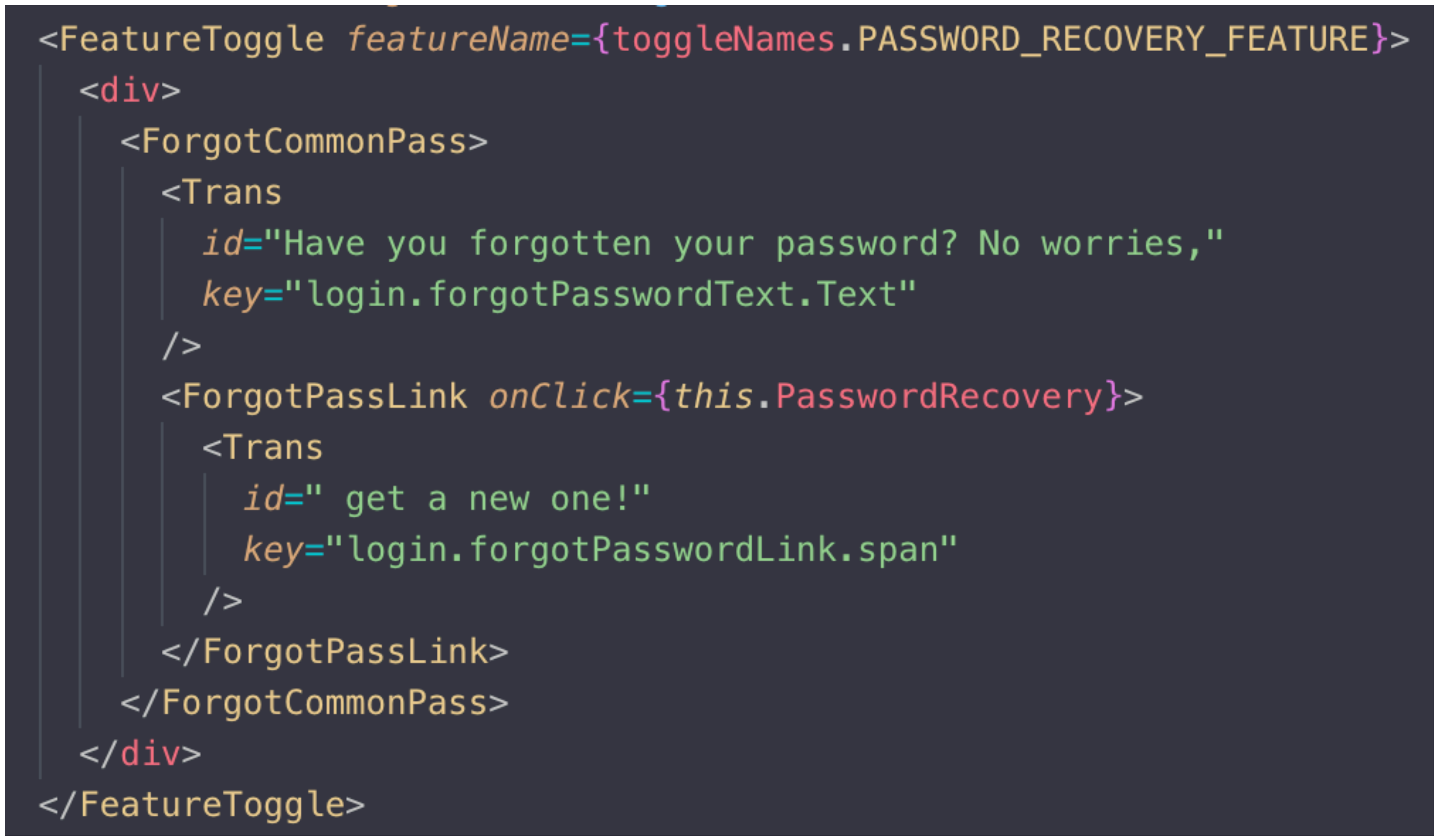
We’ve developed a kanban board with a workflow to capture each state of a feature flag: flag creation, feature rollout to internal instances, feature rollout to customers, feature rollout to 100% (and ready for cleanup), and when the process is complete.

Atlassian

Our implementation..

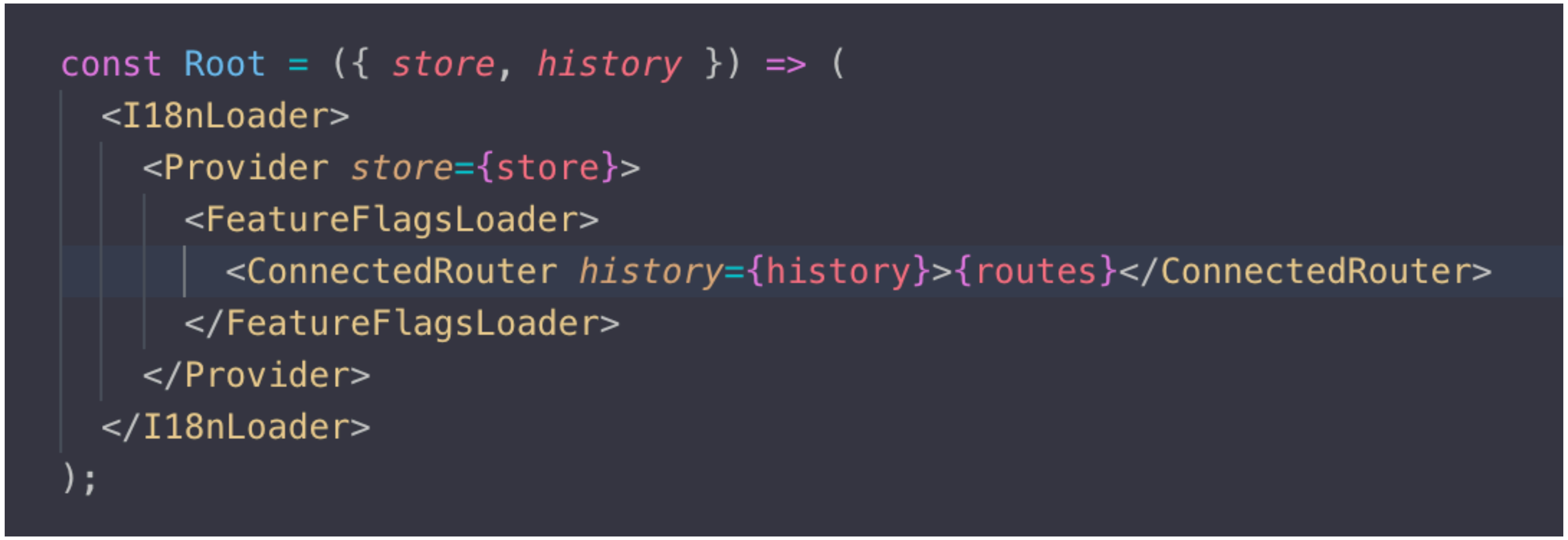
React.js

Our implementation..



```jsx
<FeatureToggle featureName={toggleNames.PASSWORD_RECOVERY_FEATURE}>
  <div>
    <ForgotCommonPass>
      <Trans
        id="Have you forgotten your password? No worries,"
        key="login.forgotPasswordText.Text"
      />
      <ForgotPassLink onClick={this.PasswordRecovery}>
        <Trans
          id=" get a new one!"
          key="login.forgotPasswordLink.span"
        />
      </ForgotPassLink>
    </ForgotCommonPass>
  </div>
</FeatureToggle>
```

Our implementation..



```
const Root = ({ store, history }) => (
  <I18nLoader>
    <Provider store={store}>
      <FeatureFlagsLoader>
        <ConnectedRouter history={history}>{routes}</ConnectedRouter>
      </FeatureFlagsLoader>
    </Provider>
  </I18nLoader>
);
```

Technical Explanation

We will have a back-office from where you can maintain a list of features in database along with their status.

Further Reading

https://medium.com/@dehora/feature-flags-smaller-better-faster-software-development-f2eab58df0f9

https://www.atlassian.com/blog/jira-software/tips-for-feature-flagging

https://code.fb.com/web/rapid-release-at-massive-scale/

https://rollout.io/blog/top-5-use-cases-feature-flags/