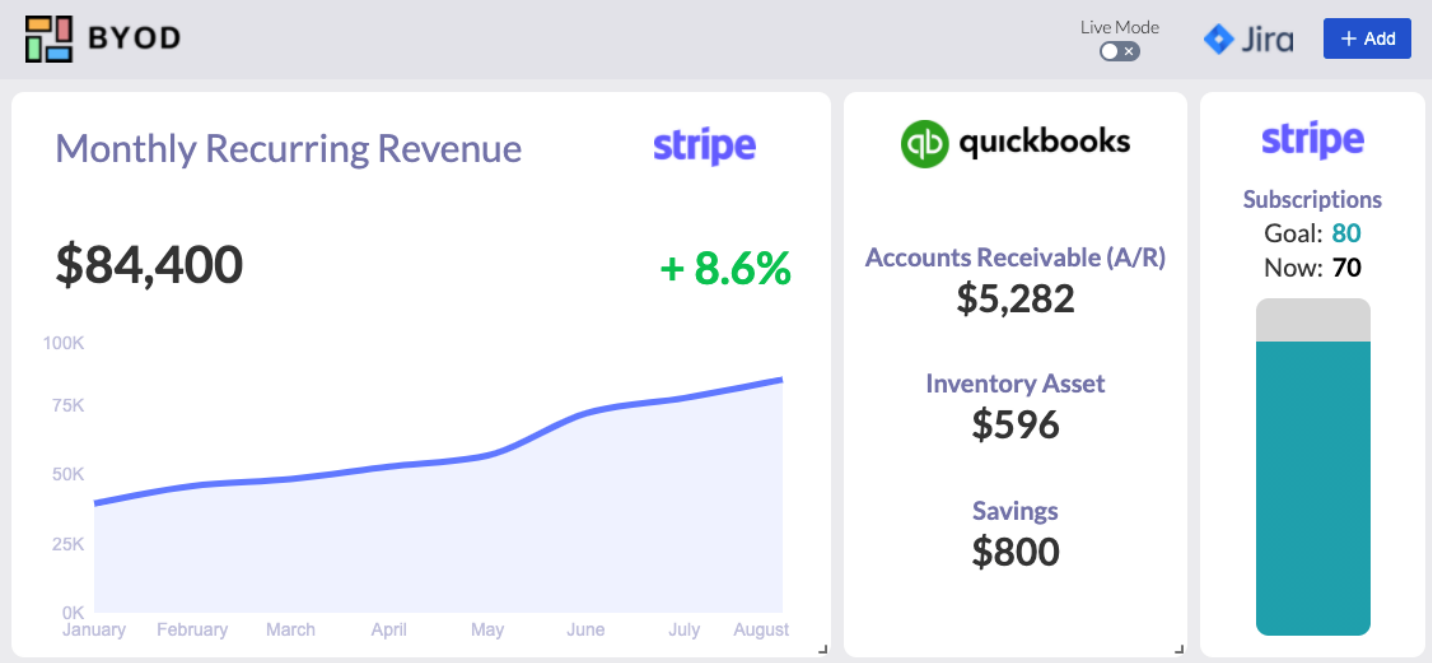
How we used Forge to build a no-code dashboard builder for Confluence

 Ashwin Kumar



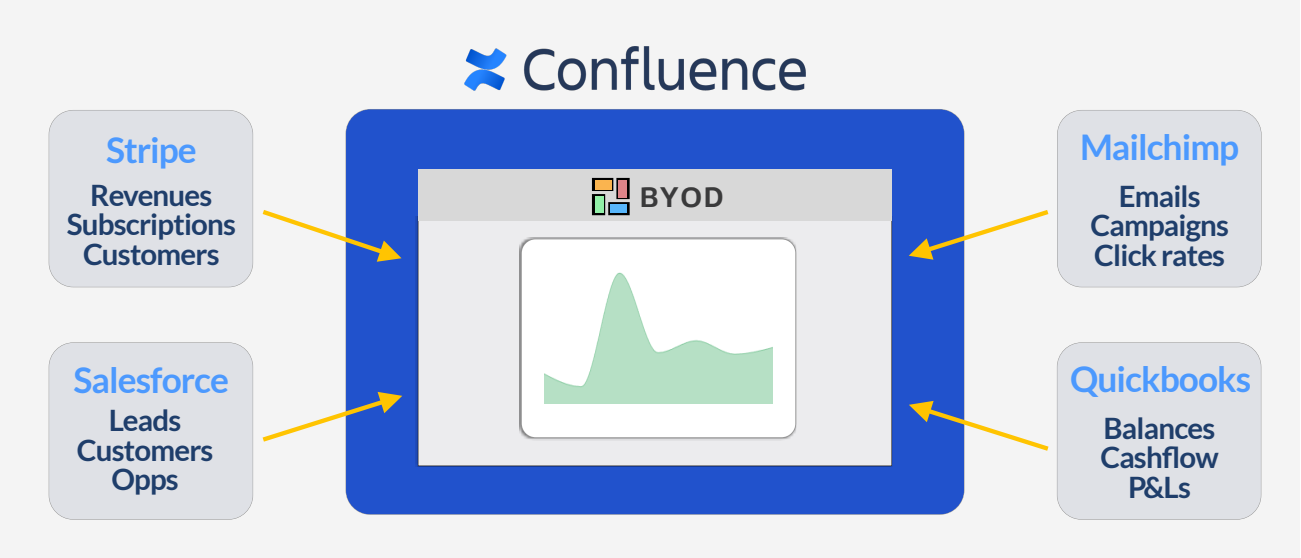
We've been avid users of Confluence at the companies we've worked at, so we were *very* excited to build a Confluence macro for the **Codegeist 2021** hackathon this year!

To determine what to build, we talked to dozens of business users of Confluence to understand their pain points, and we heard the same thing over and over:

> *"Our business data is siloed in external sources (e.g. revenue data in Stripe, cashflow data in Quickbooks, leads data in Salesforce). We really need a way to see that data in Confluence so our team can be on the same page and collaborate more effectively."*

What we built: BYOD

To solve the "disparate data source" problem, we built **BYOD** *("Build Your Own Dashboard")*, a macro that business teams can include on any Confluence page, that allows them to directly integrate with their external data accounts and pull that data into Confluence, without writing any code!



Our goals in creating BYOD were to:

- **Reduce login overhead:** Since a single team member can set up the integrations for BYOD, teams won't need to get login access for each of those external accounts for each team member anymore.
- **Eliminate stale data:** Since the BYOD macro updates the data from external accounts in real-time through API calls, the data in the BYOD dashboard is always up to date, so all team members with access to the Confluence page will be on the same page with the latest data.
- **Allow deep personalization:** Every team on Confluence has different needs, and we wanted BYOD's user interface to make it extremely simple for Confluence users to build *customizable* dashboards using whatever integrations and modules that made sense for them.

How we used Forge to build BYOD

Though BYOD is simple for the end-user to use, it was a complex endeavor to build. To build it in the timeframe of the hackathon, we leveraged *multiple* features of Forge to speed up our development.

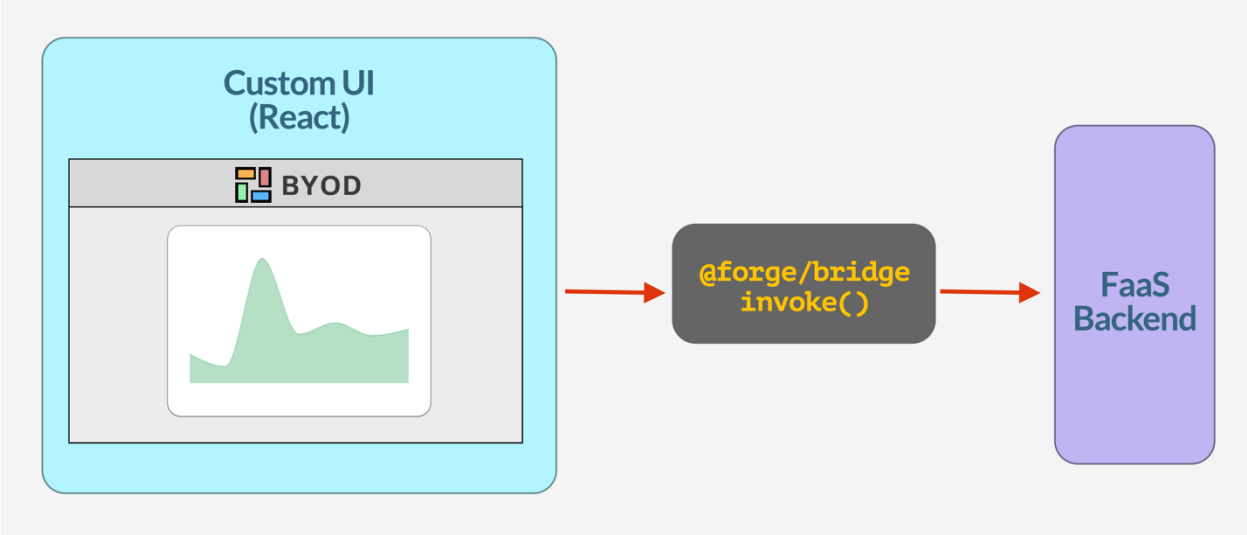
Step 1: Building the frontend with React and Custom UI

The frontend we wanted for our app was complex: we needed a way to let users have a GUI to move modules around freely and resize them within their dashboard. To do this, we used the **React Grid Layout** library and the **Custom UI** framework of Forge.

The Custom UI framework allowed us to build our entire frontend in React, and use Forge's **resolver** API to communicate with the FaaS backend hosted by Atlassian (more on that below).

Step 2: Connect the React frontend to the FaaS backend

To leverage the hosted infrastructure superpower of Forge, we used the resolver API and **Forge bridge** to let our frontend communicate with our FaaS backend.



The FaaS backend handles multiple features of BYOD:

- **Maintains state for the dashboard:** Every time a user interacts with the dashboard on the React frontend (e.g. adding, resizing, removing modules), we needed a way to save the configuration so we could restore it on new page loads. Our FaaS backend uses Forge's **Storage API** to save and load the dashboard state inside the Confluence instance, so we didn't need to set up an external database!
- **Handle callbacks for our OAuth flows:** To work with the OAuth 2.0 framework (more on that in Step 3), we needed to handle callback URLs from external integrations like Stripe and Quickbooks. We used Forge's **webTrigger API** to handle the OAuth authorization and token exchange flows.
- **Call external APIs:** The whole point of BYOD is to update its dashboard data in realtime, so we used our FaaS backend to make those API calls on behalf of the authenticated Confluence user (more on that in Step 4).

Step 3: Handle external integrations and OAuth

Since the basis of BYOD is about pulling in data from external integrations, we needed a way to let our Confluence users give BYOD permission to read that data from those external sites, using the OAuth 2.0 framework.

We saw a post about **External Auth** on the Atlassian developer forum, which will make the OAuth process *much* easier, but since it wasn't available for the hackathon, here's how we built it (using our Stripe integration as an example).

Add the integration's OAuth URLs to *manifest.yml*

manifest.yml

```
permissions:
 external:
  fetch:
   client:
     - 'https://connect.stripe.com/oauth/'
     - 'https://appcenter.intuit.com/connect/'
```

Custom UI (React)

```
import { router } from '@forge/bridge';

router.open(OAUTH_URL)
```

The first step in the OAuth 2.0 flow is to send the user to the external site to login and give BYOD permission to get access to their data for the dashboard. Since we'd be navigating the user away from Confluence for this, we included these URLs in the *client permissions* section of our manifest.yml file.

Set up webtriggers to handle authorization callbacks

manifest.yml

```
modules:
 webtrigger:
   - key: stripe-web-trigger-key
     function: stripe-oauth-webtrigger
 function:
   - key: stripe-oauth-webtrigger
     handler: index.stripeOauthWebTrigger
```

FaaS backend (index.js)

```
function stripeOauthWebTrigger (request) {
    // handle OAuth token exchange
};
```

When the user authorizes BYOD for an external integration, that site (Stripe, in this example) sends an authorization code to an endpoint that we need to exchange for an access token.

To do this, we created a Forge webtrigger URL that we registered with the external API and did the access token exchange within the webtrigger function.

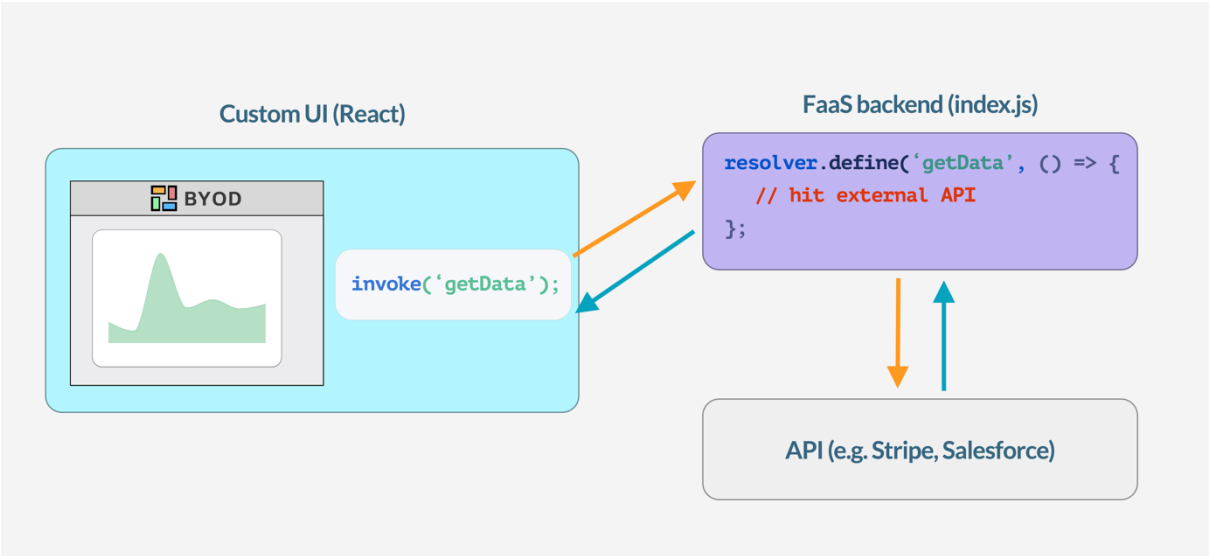
Using API secrets and saving access tokens

Since BYOD has developer accounts for each integration, we used Forge CLI's **environment variables** capability to encrypt and securely pass API keys and secrets to the application.

Storing the user's access tokens provided a bit of a challenge: we needed a way to save them to use in API calls, but we didn't want to save them to an external database for security concerns. For the hackathon we decided to encrypt the tokens and save them using the Storage API (and decrypt them when making API calls). We're excited for the External Auth feature to more securely handle the access tokens!

Step 4: Calling APIs to update data in Confluence

Finally, once we had the Confluence user authenticated with an integration through OAuth 2.0, we were able to do the fun stuff: update their BYOD dashboard data with API calls!



We were especially impressed with how we could use the manifest.yml file to "whitelist" the external API endpoints we'd be consuming, which was a huge help in maintaining the security of the macro for our Confluence users.

For the hackathon, we updated the data in near-real-time by calling each API endpoint on an interval, but our next step here would be to setup webhooks with each integration to update the data as soon as it changes.

Forge gave us development superpowers

There is no way we could have built such a complex application this quickly without leveraging Forge!

Here are just a few ways we supercharged our development speed using Forge:

- **Skip worrying about infrastructure:** We didn't have to think about hosting our app or performing compute, and with the Storage API we didn't even have to worry about setting up an external database!
- **Reduce security vulnerabilities:** Since our app ran in the Forge platform, we could rest assured that much of the security for our app would be taken care of by Atlassian. By utilizing the manifest.yml file and its permissions framework, we were able to whitelist only the URLs that we needed and nothing more.
- **Quick iteration speed:** By using Forge's **tunnel** feature, we didn't need to redeploy the app after each change, so we could develop the app very quickly before shipping to production.
- **Tight integration with the Atlassian ecosystem:** We wanted to include a "Create Jira issue" feature for BYOD, and it was insanely simple given Forge's **requestJira** API. We're excited to see how Forge's extensibility capabilities grow going forward.

What's next for BYOD?

The Codegeist hackathon was an inspiration for us and we're eager to get BYOD ready for the Atlassian marketplace so Confluence users can use the app and develop powerful dashboards for their teams!

To that end, we're focusing on:

- **Building more integrations:** For the hackathon we were able to build out integrations with Stripe and Quickbooks, but we're focused on building even more.
- **Write access for modules:** BYOD is currently a read-only dashboard, but since it's an OAuth-based app, we want to allow Confluence users to ***write*** to their integrations as well. For example, teams can respond to Zendesk support tickets right inside the BYOD macro, or they can build internal tools to authorize expense reports through Expensify.
- **Locking down security:** Security is top-of-mind for us, and we want to build out the infrastructure to safely store and use API secrets and OAuth access tokens for our users.

As mentioned before, the upcoming External Auth feature for Forge will be a *huge* help for us in achieving these goals quickly and securely, and we can't wait to see how Confluence users utilize BYOD going forward!