User Onboarding Tour TDD

User Onboarding Tour TDD 1
Administrativia 2
Review Tracker 2
Background 2
Motivation 2
Goals 3
Non-Goals 3
Solutions 4
- Solution 1 5
  - High Level Design 6
    - Diagram 6
    - Description 7
  - Low Level Design 9
  - Testing Plan 13
  - Tasks To Do 13
  - Release Plan 14
  - Benefits 14
  - Drawbacks 14
- Solution 2 14
  - Benefits 14
  - Drawbacks 14
- Solution 3 14
  - Benefits 15
  - Drawbacks 15

Conclusions 15
Future Next Steps 15

Administrativia

Authors: Reynold Morel

Approvers: Enzo Martellucci (enzo.martellucci@preset.io)

Reviewers: enzo.martellucci@preset.io, luis.sanchezm86@gmail.com

Status: Draft / **In Review** / Approved

Last Review: Feb 3, 2026

Review Tracker

Role / Team	Name / Email	Approved ✅	Conditional Approval ✳	Not Approved ❌
Software Engineer	Enzo Martellucci (enzo.martellucci@preset.io)	✅		
Software Engineer	Luis Sánchez (luis.sanchezm86@gmail.com)	✅		

Background

New users entering Apache Superset for the first time may find the application challenging to navigate. Due to the breadth of features and configuration options, key actions and workflows may not be immediately discoverable. As a result, first-time users often require additional time and experimentation before becoming comfortable with the user interface and overall user experience.

Motivation

The goal of this proposal is to introduce a guided onboarding experience that helps users understand how to navigate Superset and complete common tasks. This experience should provide contextual guidance through interactive tours that explain relevant UI elements and workflows.

In addition to supporting first-time users, the guided tours should be accessible after the initial experience, allowing users to revisit them when needed.

Goals

Given the potentially large scope of a guided onboarding system, this proposal intentionally limits its scope to ensure feasibility within a constrained development timeline (approximately 10 hours).

The goals of this effort are:

- Establish a reusable foundation for guided tours that can support additional workflows in the future.
- Automatically present a guided tour to users who have not previously completed it.
- Allow users to re-run guided tours on demand.
- Implement a single, end-to-end guided workflow:
    - Creating a new dashboard starting from a state where no charts currently exist.

Non-Goals

The following items are explicitly out of scope for this proposal:

- Implementing guided tours for all Superset workflows.
- Building a dedicated UI page or management interface for creating, editing, or administering tours.

Solutions

Regarding the FE library to use, we found the following:

Criterion	React Joyride	Shepherd.js	Intro.js
Primary Model	React-specific guided tour component	Framework-agnostic core with React wrapper	Framework-agnostic JS tour library
React Integration	Native React component API	React via wrapper (react-shepherd) (GitHub)	React wrappers available but not React-native
Custom Component Support	Supports custom React components for tooltips	Support via config; less React-idiomatic	Limited; imperative API
Maintenance / Activity	Some recent maintenance, but original repo hasn't seen major updates for a while (similar to being stable) (js.libhunt.com)	Active core Shepherd library with recent commits; React wrapper archived (but core still maintained) (GitHub)	Mature and popular, frequent releases on main repo; React wrapper may lag
Popularity / Community	Moderate GitHub stars, widely used in React apps (js.libhunt.com)	Large community for core Shepherd; smaller React wrapper	Very large and established, many users and examples (OnboardJS)
License	MIT	MIT	AGPL-v3 (commercial license available) (introjs.com)
Bundle Size / Dependencies	Larger due to React and floater positioning	Moderate; requires CSS and popper logic	Lightweight core; CSS + JS
Feature Set	Flexible tours, callbacks, continuous flows	Highly customizable, modal overlay, async hooks	Simple tours with progress bar, hint steps

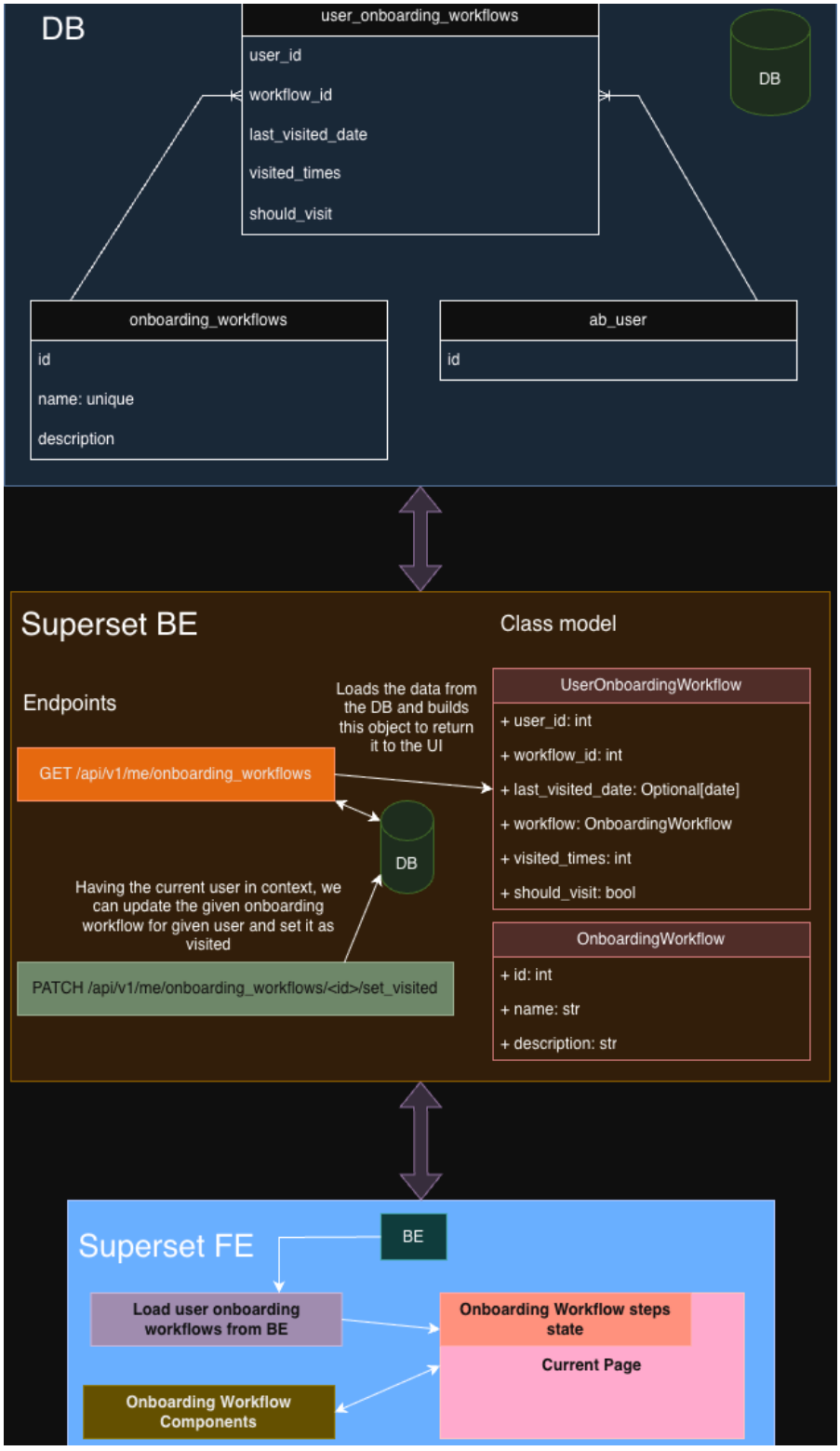
Typical Use Case	React apps needing deep component control	Complex multi-framework apps	Simple declarative tours, broad browser compatibility

Solution 1

Consists on supporting a structure in the BE to store the workflows a user can see or go through and store, if it was visited or not, last visited date and the times the user has gone through an onboarding workflow. Finally, in the FE we can load / control the onboarding workflow state depending on if the user has gone through it or not. We are planning to use https://docs.react-joyride.com/ for the FE, this library would be in charge of adding the tour experience and connecting each component for each step.

High Level Design

Diagram



Description

This proposal introduces a backend-backed onboarding framework to support guided workflows and track user progress across sessions.

Data Model

Two new database tables will be introduced to support onboarding workflows and per-user progress tracking:

1. **Onboarding Workflows**
   A table to define the available onboarding workflows supported by Superset.

   Example fields:
   a. id (integer, primary key)
   b. name (string, unique)
   c. description (string)

2. **User Onboarding Workflows**
   A mapping table to track onboarding workflow usage on a per-user basis. This table will be used to persist whether a user has completed or interacted with a given onboarding workflow, as well as basic usage metadata.

   Example fields:
   a. user_id (integer, foreign key to the user table)
   b. workflow_id (integer, foreign key to onboarding workflows)
   c. last_visited_date (nullable timestamp)
   d. visited_times (integer)
   e. should_visit (boolean)

Backend API

Two new **REST API** endpoints will be added under the current user namespace to expose onboarding workflow data to the frontend:

- Retrieve onboarding workflows for the current user
  GET /api/v1/me/onboarding_workflows

  Returns the list of onboarding workflows along with user-specific visitation state.

- **Mark an onboarding workflow as visited**
  PATCH /api/v1/me/onboarding_workflows/<workflow_id>/set_visited

  This endpoint will update the corresponding user-workflow record by:
  - Incrementing ***visited_times***
  - Updating ***last_visited_date*** to the current timestamp.

Frontend Integration

The frontend will load onboarding workflow metadata during user login to determine which workflows should be presented automatically and which can be manually re-triggered.

A dedicated Superset component will encapsulate all onboarding workflows and their configuration. Individual pages or components will be responsible for defining and managing the state and progression of their associated onboarding steps.

Migration Strategy

A database migration will be introduced to initialize onboarding workflow records for existing users. This ensures that all users have a consistent baseline state for onboarding workflows without requiring manual intervention.

Notes for Future Extension

The proposed design establishes a reusable foundation that can support additional onboarding workflows over time without requiring changes to the core architecture.

Roles system impact

According to the current implementation only **Gamma and Public** users will not be able to mark their tour as visited since they have very limited access, mostly read access permissions to the API resources. This means the tour will always show up and these users will need to skip it.

Maintenance

For the first version of this feature, the dev process to add a new workflow will be:

1. Dev adds a new migration using Alembic to add the new onboarding workflow and assign it to users.
2. Dev adds the new onboarding workflow in the FE.

Low Level Design

We could create the 2 tables mentioned in the previous section by leveraging the existing ORM:

```
class OnboardingWorkflow(Model, AuditMixinNullable):

    __tablename__ = "onboarding_workflows"

    id = Column(Integer, primary_key=True)

    name = Column(String(100), unique=True)

    description= Column(String(255))



class UserOnboardingWorkflow(Model, AuditMixinNullable):

    __tablename__ = "user_onboarding_workflows"

    user_id = Column(Integer, ForeignKey("ab_user.id"))

    workflow_id = Column(Integer, ForeignKey("onboarging_workflows.id"))

    workflow = relationship("OnboardingWorkflow")

    visited_times = Column(Integer, default=0)

    should_visit = Column(Boolean, default=False)
```

This will create both the model and the tables we want at once.

It would be a good idea to create an ***UserOnboardingWorkflowDAO***, that way we can abstract the way we access onboarding workflow data and also we don't expose db connection instances outside of the DAO layer:

```
class UserOnboardingWorkflowDAO(BaseDAO[UserOnboardingWorkflow]):

    @staticmethod

    def get_by_user_id(user_id: int) -> List[UserOnboardingWorkflow]:
```

```
        # add code here



    @staticmethod

    def set_visited(user_id: int, onboarding_workflow_id: int) -> None:

        # add code here
```

Then we will add the 2 new endpoints mentioned above probably within the ***CurrentUserRestApi***:

```
    @expose("/onboarding_workflows/", methods=("GET",))

    @protect()

    @permission_name("read")

    @safe

    def get_my_onboarding_workflows(self) -> Response:

      # add your code here



@expose("onboarding_workflows/<int:onboarding_workflow_id>/set_visited",
methods=("PATCH",))

    @protect()

    @permission_name("write")

    @safe

    @statsd_metrics

    @event_logger.log_this_with_context(
```

```
        action=lambda self, *args, **kwargs:
f"{self.__class__.__name__}.patch",

        log_to_statsd=False,

    )

    def set_onboarding_workflow_visited(self, onboarding_workflow_id: int)
-> Response:

      # add your code here
```

At this point we should have exposed the endpoints to the FE.

In the FE, we should create the 2 calls to the new endpoints above:

```
async loadOnboardingWorkflows(): Promise<QueryData[]> {

      // add code here

      const requestConfig: RequestConfig = {

            endpoint: '/api/v1/me/onboarding_workflows',

      };

}



async setOnboardingWorkflowVisited(onboardingWorkflowId: number):
Promise<QueryData[]> {

      // add code here

      const requestConfig: RequestConfig = {

            endpoint:
`/api/v1/me/onboarding_workflows/${onboardingWorkflowId}`,

      };

}
```

We can add a new plugging to wrap https://docs.react-joyride.com/ and then add some configurations to it. These configurations are meant to be used to set **JoyRide** properties and only expose what we are going to use from the library, that should look something like:

```
<OnboardingWorkflow

 name="CREATE_DASHBOARD_TOUR"

 stepsOverrides={{step1: { content: <MyCustomContent />, placement:
'left', target: 'component_class' }}}

 run={run}

 continous

/>
```

Then we could just use this within a page.

For the migrations we could just use ***Alembic***, see other migrations here: ***superset/migrations/versions***, the main idea is to create the onboarding workflows there and assign them to the users.

We can also be jumping between pages for an onboarding workflow, we could use the query params from the browser to pass down any needed flag or state in case the workflow is complex enough that might need jumping between multiple pages.

Testing Plan

Since this needs to work at a fast pace, we should probably add happy path unit tests per each component, endpoints and models added. Probably we can also add at least one single E2E test, one for the FE and one for the BE testing a simple onboarding workflow.

Also we could test the library itself without the backend by using local storage to store the tour steps and visits.

Tasks To Do

Backend ✅

1. Create the models for the onboarding workflows entities ✅
    - PR: https://github.com/apache/superset/pull/37680
2. Add the DAO for the user onboarding workflow entity + tests ✅
    - PR: https://github.com/reynoldmorel/superset/pull/1
3. Add the endpoint to load the onboarding workflows + tests ✅
    - PR: https://github.com/reynoldmorel/superset/pull/2
4. Add the endpoint to set onboarding workflow as visited + tests ✅
    - PR: https://github.com/reynoldmorel/superset/pull/2

Frontend ✅

1. Add the endpoint to load the onboarding workflows + tests ✅
    - PR: https://github.com/reynoldmorel/superset/pull/3
2. Add the endpoint to set onboarding workflow as visited + tests ✅
    - PR: https://github.com/reynoldmorel/superset/pull/3
3. Add new component to wrap the https://docs.react-joyride.com/ + tests ✅
    - PR: https://github.com/reynoldmorel/superset/pull/4
4. Add onboarding workflow for creating dashboard from a none existing chart + tests ✅
    - PR: https://github.com/reynoldmorel/superset/pull/5
5. Implement onboarding workflow for creating dashboard from a none existing chart + tests ✅
    - PR: https://github.com/reynoldmorel/superset/pull/5

PR having all the changes:

https://github.com/apache/superset/pull/37929

Release Plan

- We will need to put the feature behind a feature flag.
- We would need to split each task from above in a PR to avoid huge PRs, each PR will be dependent on each other.

Benefits

- **Scalable solution:** we could add a UI at any point so the user can decide when to revisit the onboarding workflow.
- We could extract important metrics about the behavior of the user for a given workflow in the application. Metrics could let us know:
    - Feature adoption
    - How hard the application from the user perspective
    - If the onboarding workflow is useful or not.
- Could probably help bring more users due to being more user-friendly.
- Centralized and reusable workflows
- Possibility of automating the onboarding in a longer future, not needing to code to build the workflow

Drawbacks

- It is a complex solution
- Involves BE, FE, DB to work
- It will require time and work to be useful: one workflow might not be enough

Solution 2

Same as solution 1 but using https://docs.shepherdjs.dev/guides/install/.

Benefits

- Strong control over tour flow and positioning.
- Supports advanced features (e.g., modal overlays).

Drawbacks

- It has pricing for commercial applications (probably will not apply for superset)

Solution 3

Same as solution 1 but using https://introjs.com/.

Benefits

- Fast setup, minimal config.
- Good for simple tours.

Drawbacks

- Less flexible than Joyride/Shepherd for complex logic.

Conclusions

We decided to go with solution 1 initially, this is because we are opened to move from Joyride library in case it doesn't work

Future Next Steps

- Add a CRUD of onboarding workflows, so we can avoid using Alembic migrations.
- Add a batch workflow assignment tool, so we can avoid using Alembic migrations.
- Support tracking user status in a tour. For example we should know if a workflow never finished, the last step the user went through, the time the user spent in a step…