The BrownField Airline is one of the fastest growing, low-cost, regional airlines, flying directly to more than one hundred destinations from its hub. As a start-up airline, BrownField Airline has started its operations with a few destinations and a few aircrafts. The BrownField has developed its home-grown PSS application to handle their passenger sales and services.

Business process view

For discussion purposes, this use case is considerably simplified. The process view in the following diagram shows BrownField Airlineâ€™s end-to-end passenger services operations covered by the current PSS solution:

A picture containing graphical user interface

Description automatically generated

The current solution automates certain customer-facing functions, as well as certain internally-facing functions. There are two internally-facing functions, **Pre-flight** and **Post Flight**. **Pre-flight** is in the planning phase, used for preparing flight schedules, plans, aircrafts, and so on. **Post Flight** is used by the back office for revenue management, accounting, and so on. **Search** and **Reserve** are functions, which are part of the online seat reservation process, and the **Check-In** function is the process of accepting passengers at the airport. The Check-In function is also accessible to the end users over the Internet for online check-in.

The cross marks at the beginning of the arrows in the diagram indicate that they are disconnected, and occur at different timelines. For example, passengers are allowed to book 360 days in advance, whereas, check-in generally happens 24 hours before flight departure.

Functional view

The following diagram shows the functional building blocks of BrownField Airlineâ€™s PSS landscape. Each business process and its related sub-functions are represented in a row:

A picture containing diagram

Description automatically generated

Each sub-function shown in the preceding diagram explains its role in the overall business process. Some sub-functions participate in more than one business process.

Let us assume that we have reproduced the functions and their dependencies as shown in the following diagram:

A picture containing graphical user interface

Description automatically generated

There are many dependencies going back and forth between different modules. The bottom layer shows cross-cutting capabilities, which are used across multiple modules. At this point, the modules are more like a spaghetti rather than autonomous units.

### Orchestrating microservices

The logic of the booking orchestration and the execution of rules sits within the Booking service. The brain is still inside the Booking service in the form of one or more booking business components. Internally, business components orchestrate private APIs exposed by other business components or even external services.

As shown in the preceding diagram, the booking service internally calls to update the inventory of its own component, as well as calls the **Fare** service.

Text

Description automatically generated with low confidence

As shown in the preceding diagram, the booking service internally calls to update the inventory of its own component, as well as calls the **Fare** service.

Handling exceptions

Examine the booking scenario to understand different exception-handling approaches.

In the service sequence diagram below, there are three lines marked with a cross mark. These are potential areas where exceptions could occur.

Timeline

Description automatically generated

There is a synchronous communication between Booking and Fare. What if the fare service is not available? If the Fare service is not available, throwing an error back to the user may cause revenue loss. An alternate thought is to trust the fare which is coming as part of the incoming request. When we serve search, the search results will have the fare as well. When the user selects a flight and submits, the request will have the selected fare. In case the Fare service is not available, we trust the incoming request and accept the Booking. We will use a circuit breaker and a fallback service, which simply creates the booking with a special status, and queues the booking for manual action or a system retry.

What if creating the booking fails? If creating booking fails unexpectedly, a better option is to throw a message back to the user. We could try alternative options, but that would increase the overall complexity of the system. The same is applicable for inventory updates.

In the case of creating a booking and updating the inventory, we avoid a situation where a booking is created and the inventory update somehow fails. As the inventory is critical, it is better to have both create booking and update inventory to be in a local transaction. This is possible, as both components are under the same subsystem.

If we consider the **CheckIn** scenario, **CheckIn** sends an event to **Boarding** and **Booking** as shown in this diagram:

A screenshot of a computer

Description automatically generated with medium confidence

Consider a scenario where the check-in services fail immediately after the check-in complete event is sent out. The other consumers processed this event, but the actual check-in is rolled back. This is because we are not using a two-phase commit. In this case, we need a mechanism for reverting that event. This could be done by catching the exception, and sending another check-in cancelled event.

## **Target implementation**

The following diagram represents the implementation view of the BrownField PSS microservices system:

A picture containing timeline

Description automatically generated

As shown in the preceding diagram, as an example, we are implementing four microservices--**Search**, **Fare**, **Booking**, and **Check-In**. In order to test the application, there is a website application developed using Spring MVC with Thymeleaf templates. The asynchronous messaging is implemented with the help of RabbitMQ. In this sample implementation, the default H2 database is used as the in-memory store for demonstration purposes.

Implementation projects

The basic implementation of the BrownField Airlineâ€™s PSS microservices system has five core projects, as summarized in the following table. The table also shows the port range used for these projects to ensure consistency throughout the book.

|  |  |  |
| --- | --- | --- |
| **Microservices** | **Projects** | **Port Range** |
| Book Microservice | chapter6.book | 8060-8069 |
| Check-in Microservice | chapter6.checkin | 8070-8079 |
| Fare Microservice | chapter6.fares | 8080-8089 |
| Search Microservice | chapter6.search | 8090-8099 |
| Website | chapter6.website | 8001 |

The website is the UI application for testing the PSS microservices.

Diagram

Description automatically generated