Mailer Refactor

Author: Vic Woeltjen
Approvers: @tommychheng, @kevinvan, @jeremykwa
Date: 2/5/2019

What is this feature?

The goal of this feature is to refactor the “mailer” service to increase confidence in correctness and to improve maintainability. Notably, we’ve had issues with unclear, incomplete, or poorly-formatted details in mailings, and difficulty in debugging and testing our mailer has become evident.

Goals for this refactor:

- Reduce the amount of boilerplate in templates (deduplicate markup, code)
- Simplify interface (provide clear, consistent inputs)
- Abide by code style standards in use elsewhere
- Make it easy to add new templates
- Decouple from other services
    - In particular, mailer.suchAndSuch calls appear inline during flows implemented by other services, making it easy to break these during unrelated maintenance.
- Simplify debugging
    - Would like to be able to exercise mailer (and specific mailings) without exercising surrounding behavior (e.g. trigger a “guest requested to book” without actually making a new booking)

Who will use this?

This is a refactor; primary users for the related set of changes will be internal developers.

Refactored component (mailer) is used to interact with host and guests at critical points in the booking flow.

Implementation Overview

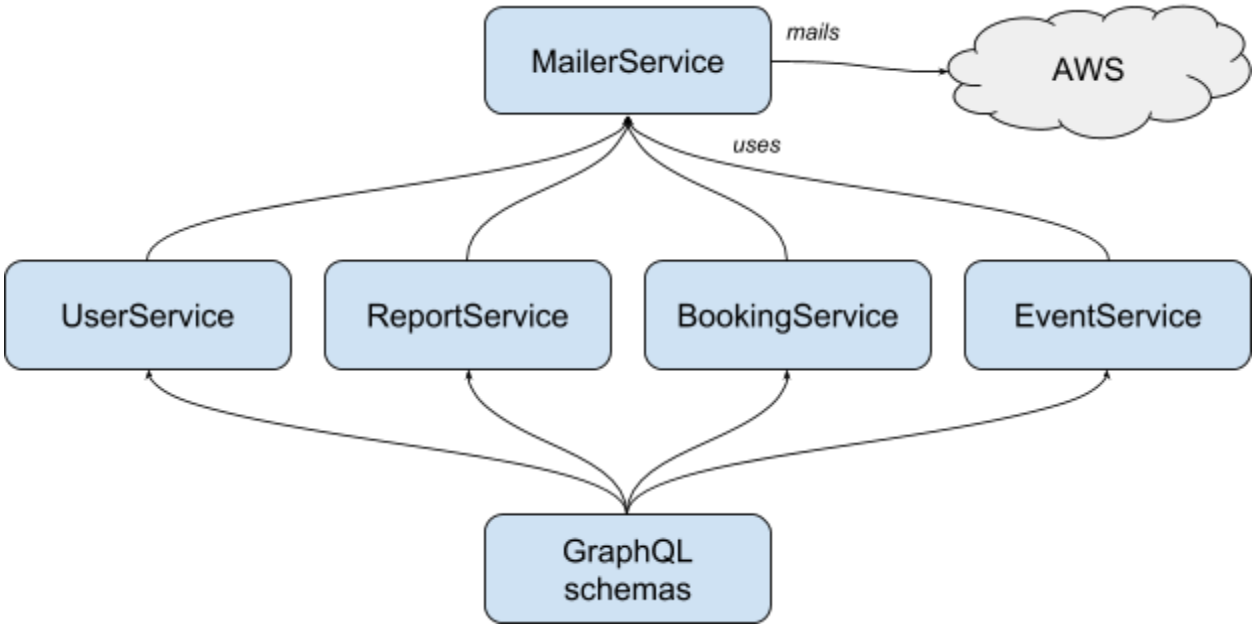
Existing mailings, as exposed from mailer:

Mailing	**When?**	**Recipient**	**Dependent**
`contactUser`	User contacts host	Host	User GraphQL
`sendEmail`	Main Weekly Report	Internal	ReportService
`reportOnboarding`	Onboarding Report	Internal	ReportService
`reportFailedAutopilotHostSignup`	Unused?	Internal	None?
`guestRequestsBooking`	Guest confirm	Guest	BookingService
`guestBookingAcceptedByHost`	Host accept	Guest	BookingService
`guestRequestRejectedByHost`	Host reject	Guest	BookingService
`guestRequestCancelledByHost`	Host cancel	Guest	BookingService
`hostNotificationOfGuestRequest`	Guest confirm	Host	BookingService
`hostNotificationOfGuestCancellationWithoutPenalty`	Guest cancel	Host	BookingService
`hostNotificationOfGuestCancellationBeforeDeadline`	Guest cancel	Host	BookingService
`hostNotificationOfGuestCancellationAfterDeadline`	Guest cancel	Host	BookingService
`contractEventMismatch`	Invoice	Internal	ContractEventService

Candidate Approaches

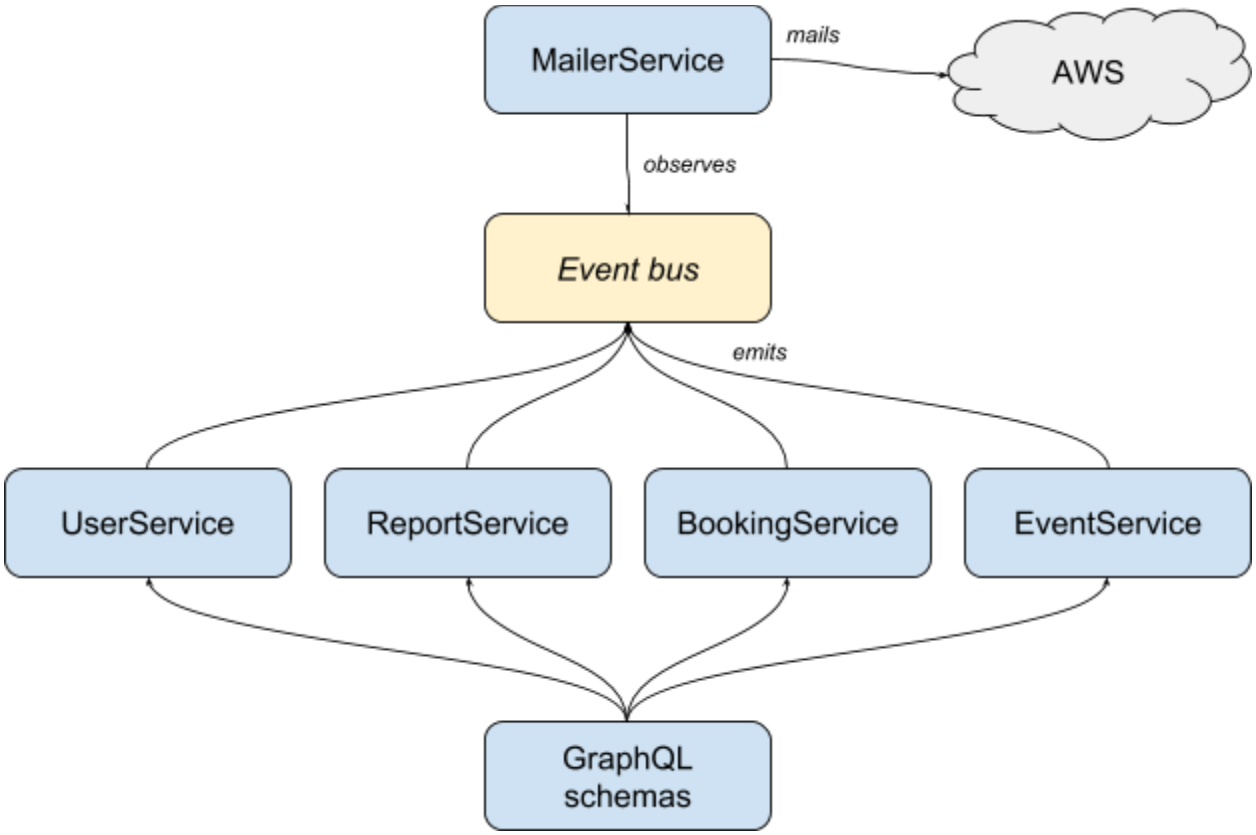
Simplified Interface Only

In this solution, MailerService interface is revised for simplicity. Instead of exposing separate methods to build parameters and send mailings, take domain objects (e.g. Booking objects) as arguments and utilize those at the service level. Update usages.



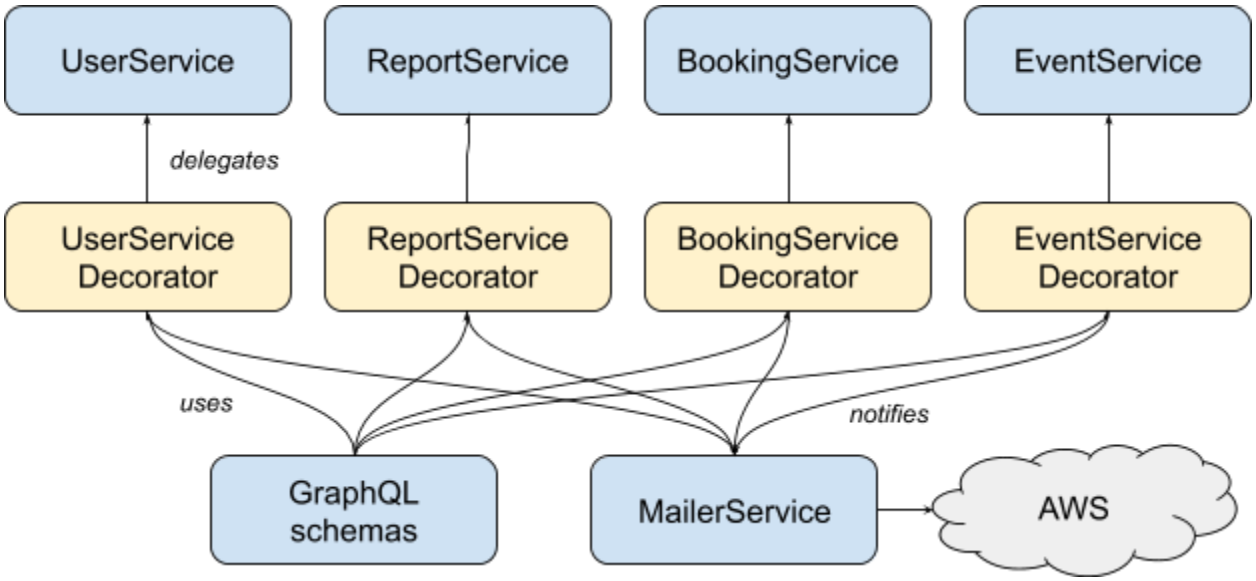
- MailerService with simplified interface (remove buildParams)
    - Most explicit solution
        - Clear what sends emails and when (when viewing implementation)
        - ...but unclear when viewing service interfaces
            - Do I expect BookingService.foo to send an email?
    - Error-prone due to mixing of concerns
        - Sending emails become concerns of the booking service, which wants to be focused on managing bookings

Event-Driven



- Event listener, event dispatcher
    - Introduces event broadcasting as a ubiquitous responsibility
        - More general than sending emails, but still out of the narrow scope a service desires
    - Provides clear semantics for when and why emails get issued
        - When this happens, send that email

Service Decoration



- Service decoration
    - Hides mailing responsibilities entirely from service implementation
    - Provides clear semantics for when and why emails get issued
        - When this service call succeeds, send that email
    - Intrudes in broader architecture

Recommendation

Though it addresses the fewest concerns, the “service interface only” approach offers an incremental improvement and will be the smallest deviation from our current architecture. Other approaches introduce trade-offs and it is unclear if the benefit will justify the cost.

Instead of revisiting application-level architecture with respect to the mailer, focus on internal architecture of the mailer and improvement of its external interfaces. This is sufficient to address most of the priorities itemized for this issue, and leaves open the possibility of future architectural changes.

Internal Refactor

Goals for internally refactoring the mailer component:

- Move boilerplate and common markup into one place
    - Already partially done with header/footer
    - Consolidate CSS into one place
- External interface should accept domain objects (e.g. Booking instances)
- Provide a clear pattern for adding new (or maintaining existing) templates
    - Find this through iterative simplification of existing calls

List the security concerns and attack vectors

Using domain objects from the mailer means we have access to potentially sensitive information at the time a mailing is being generated. Care should be taken to avoid leaking sensitive information as a consequence (arguably, this reduces the likelihood of such a leak by colocating the relevant code in an auditable place.)

How is this an improvement over the existing system?

Currently, our mailer exposes a set of “method pairs” per kind of email (one to build parameters for the next), resulting in complicated usage throughout the application. Internally, mailings themselves currently replicate a lot of HTML from template to template. Additionally, multiple steps and parts are involved for each mailing, increasing maintenance effort, as well as the cost of adding new mailings which follow the same pattern.

The interface refactor will remove the “build” half of the “method pairs”, simplifying usage; instead, methods will take a small set of domain objects as arguments and handle the retrieval of relevant properties. The internal refactor will focus on deduplication of code and simplification of flows.

Implementation Details

Backend

Implementation plan:

1. Add a wrapper (as services/mailer/mailer.js, normalizing directory structure a bit) which accepts domain objects and maps those to mailer calls.
2. Refactor existing components to use the wrapper

3. Reimplement mailer with interface from the wrapper

Data Format

No changes required to data models. Interface change in mailer will accept domain objects (e.g. Booking models) instead of a call-specific parameter object.

Frontend

This change is front-end neutral.

Maintenance/Support Details

Backend

No new dependencies acquired. Maintenance of non-mailer components should be marginally simplified (mailer interactions are simpler) and maintenance of mailer itself should be significantly improved (reduction of work to add new templates)

Frontend

No frontend impact expected