Engineering Design - Multiplex Testing

System design to support multiplex (flu + covid) reporting in SimpleReport.

Author: *Emma Stephenson*
Last Updated: Feb. 22, 2022

Problem Statement

The PRIME mission is to get better, faster, complete and accurate data to STLT public health departments so they can take timely appropriate action, with an emphasis on flexibility and scale for pandemic readiness. SimpleReport is currently a COVID-19 specific tool. It is unlikely that the next pandemic will be COVID-19 and in order to become more flexible we are looking to expand SimpleReport's capabilities to start recording Flu results in the app.

We want to support the growing number of SimpleReport users testing on multiplex devices that test for COVID-19, Flu A & B with a single patient sample. As of writing, SimpleReport supports four different multiplex devices:

- Abbott Alinity m (SARSCoV2, Flu A, Flu B, RSV)
- Princeton BioMeditech Corp Status COVID-19/Flu (SARSCoV2 N antigen, Influenza A antigen, Influenza B antigen)
- Quidel Corporation Sofia 2 Flu + SARS antigen FIA* (SARS antigen result, Influenza A antigen result, Influenza B antigen result)
- CDC Influenza SARS-CoV-2 Multiplex Assay*

Over 65,000 tests have been recorded in SimpleReport using these devices. We want users to be able to report flu results from these devices in tandem with COVID-19 results for three reasons:

1. To better support users as they track disease outbreaks in their facilities and organizations.
2. To provide a proof-of-concept to the CDC and other public health partners that flu data can be collected and is useful.
3. To begin the engineering work required to add additional diseases.

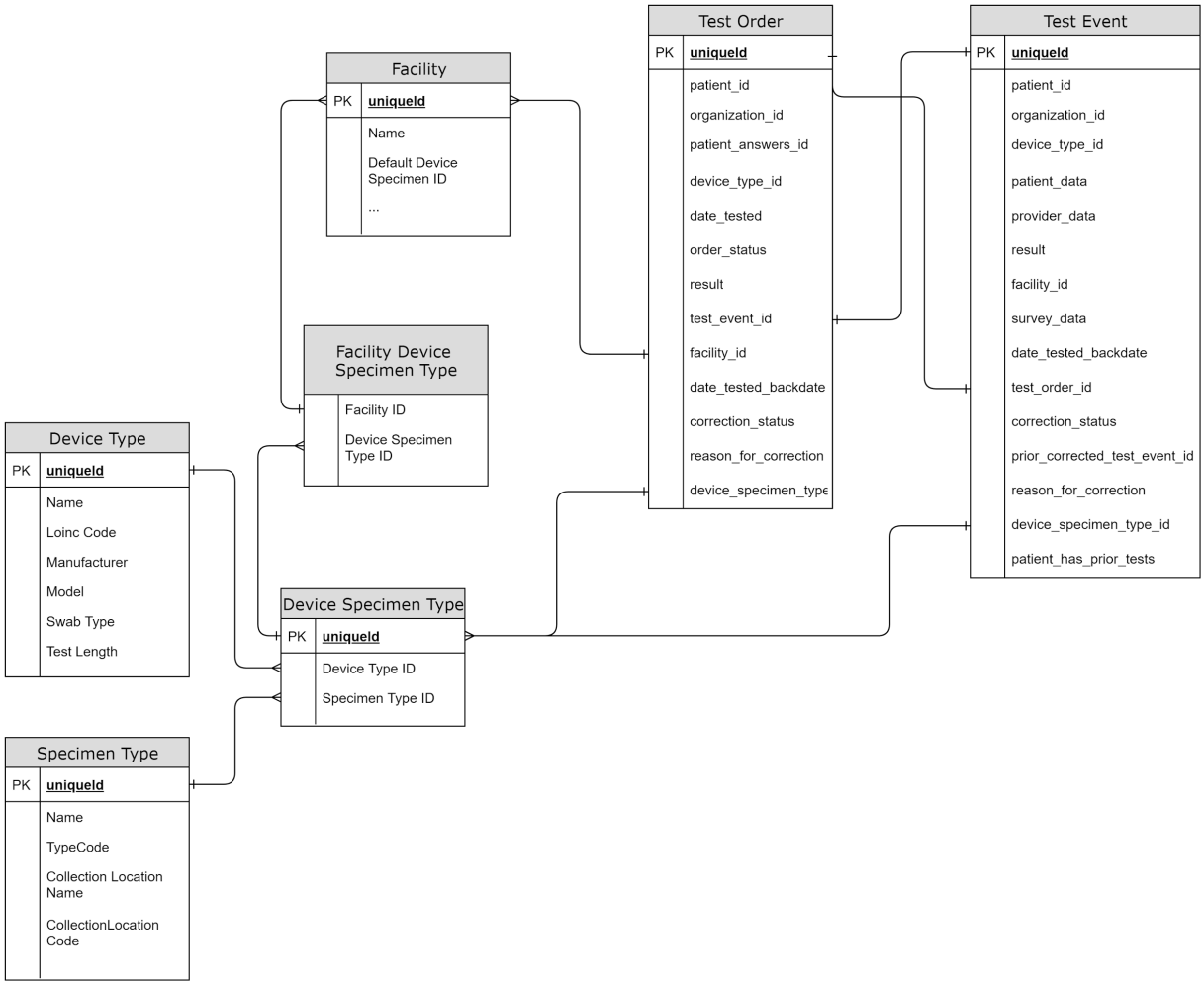
For this iteration of multiplex testing, we will not focus on how to report these results to public health departments through ReportStream, as many departments do not want this data yet. Instead, we'll focus on how to collect and store these results within the SimpleReport system. We believe this will provide value to our end users by allowing them to review the test results and provide a more complete test result to the patient via SimpleReport's SMS and SMTP test result delivery.

Detailed Design

Background

SimpleReport was designed and built to support recording and reporting of COVID-19 results. There are many assumptions baked into the application that will need to change if we add multiplex testing, or indeed any additional diseases.

SimpleReport's existing data model is predicated on the assumption that there are devices which record test results. These results can either be positive, negative, or inconclusive. We also have an organization-facility model, where organizations are the broad groups doing testing and facilities are the specific sites where the tests are performed. People (patients) are the ones being tested, while api users are the ones doing the testing (or at least the reporting.) Patients can be tested multiple times for the same disease, and post different results. All these elements of our data model will still hold true for recording flu results. The major change is that now our devices will not only be testing for COVID-19, but also for flu.



The ERD above describes the existing database as it relates to devices and test events. Test orders are created when a patient is added to the testing queue, and test events are created once the result is recorded. Both test orders and test events hold result and device data, among others.[1] The test event is what is ultimately passed to ReportStream.

Specimens include data about the specific type of swab/sample collected (i.e. nasal swab, saliva, etc) while devices include information about the name, model, and type of device. Together these pieces of data combine to make a device specimen type; something like Abbott BinaxNow Nasal Swab. Facilities have a list of device specimen types that can be used to conduct tests at that facility.[2]

[1] We may soon remove test orders as they largely duplicate the information in test_event after a test is completed. Historically TestEvents were created to enforce immutability, so that even if data like patient address or name changed, the test event would preserve the original recorded data.
[2] Historically facilities had a single default device and a list of other devices, and no specimen types at all. This has recently changed to the facility device specimen type model. The concept of a default

We'll also need to make sure patient results are updated to record the results of all tests run on the multiplex device, not just COVID-19 results.

Suggested Solutions

We don't yet have frontend designs for reporting flu results, nor will we be passing flu data along to ReportStream yet. As such, this system design will focus on the data model changes required to add multiplex testing to SimpleReport. I've outlined a few potential solutions below:

Option 1: Add new device specimen types specifically for recording Flu A/B.

In this option, we would add new device/specimen types to record Flu A and B results, with the understanding that all prior device-specimen types recorded COVID-19 results. This option is reasonable for adding additional diseases that will have their own testing devices, but the data model breaks down for multiplex testing because there's a single device/specimen collection and test for multiple diseases.

There's more discussion around how additional diseases relate to swab types here: Notes from 2nd Disease

Option 2: Allow a single test event/order to record multiple results.

Given that multiplex tests are really a single test with multiple results, we could update test events/orders to follow the same model. This has the benefit of hewing closest to reality, but would require a lot of changes to our database and backend more generally. It also doesn't scale particularly well to additional diseases if we customize the test events table to have results for COVID-19 and flu specifically.

Option 3: Add a "disease" column to devices and test orders/events.

In this option, we add a disease type column to devices and test orders. Our multiplex devices would have a list of three diseases in the new column (COVID-19, Flu A, and Flu B), and a test performed on a multiplex device would generate three different test events, one per disease. This approach has the advantage of being easily extensible to future diseases SimpleReport might cover.

---

device still perpetuates in the database, but it's not a true default - rather it's the most recent device/specimen used by the facility.

I propose that we implement option 3. This solution is relatively lightweight and the data model changes can be implemented quickly. It also allows us to easily filter out only COVID-19 results to send to public health departments at this time. We will need to update patient results to display test results for three different diseases when applicable, which will likely require us to link the three test events together in the database. This could be achieved by either repurposing the test order object to contain multiple test events, or by creating a new index table that links together results by patient and time of test.

Final Solution

After discussion with the larger engineering team, we will implement a combination of proposed solutions 2 and 3 above. Under this model, we would add a disease table to the database that links to all the diseases SimpleReport supports (for now, COVID-19 and Flu A&B.) Test events and test orders will also be changed to store multiple results. We believe that this solution will allow us to expand to multiplex tests, with extensibility to test for other second diseases in the future. It should also preserve some core app functionality that we want to avoid touching too much in this change if possible, particularly the patient experience and our ReportStream functionality. Patients will still receive a single text, it will just contain all their results instead of only their covid result. We can also filter test event results to only send COVID-19 results to ReportStream.

Disease Table & LIVD

To better reflect the different diseases a device can support, we'll need to add a new disease table. The source of truth for this data is the CDC's LIVD table, which maps all available and approved testing devices to the diseases they test for, the swab types they use, the potential results, etc. Today this data is stored in a single Excel spreadsheet, but the ReportStream team has recently built an API that supports lookups to the table. Reading from this API on every SimpleReport device interaction is unsustainable (we'd need to call the API for almost every data interaction a user has) but given that the information in the table doesn't update very often, we instead will use it as the source of truth for a daily disease table update. We'll create a supported disease table that lists our supported diseases and their potential results, and link this table to our device table. We'll run a daily job that checks the LIVD API with the devices stored in our table to make sure all the data is up to date.

Changes to Test Event/Order

Changing test orders and events to store multiple results hews closest to the realities of multiplex testing. A single test is performed, and multiple results are generated. Much of the data between flu tests and covid tests will be shared (patient demographic information, symptoms, facility, ordering provider, etc.) with only the results of the tests differing.

Test orders and test events currently store a test result, which is simply an enum of POSITIVE, NEGATIVE, or UNDETERMINED. We will need to instead store multiple results per test event/order, not just one.

The team originally discussed storing results as a json, as this is a fairly logical representation of the test (a key for each test performed, and the result stored as the value for that key.) However, there are some drawbacks to this approach - it's not best practice for relational databases, and would make it very expensive to query by test result (i.e., to get all the positive tests in a facility to show users.) Instead, we will store results in a separate table, linked to test events/orders and diseases by foreign key. We will no longer store anything in the test event/row result column, but instead perform a lookup within the results table when necessary.

Storing results in an external table allows us to continue filtering and sorting by test result, and should still be performant for looking up individual test event results.

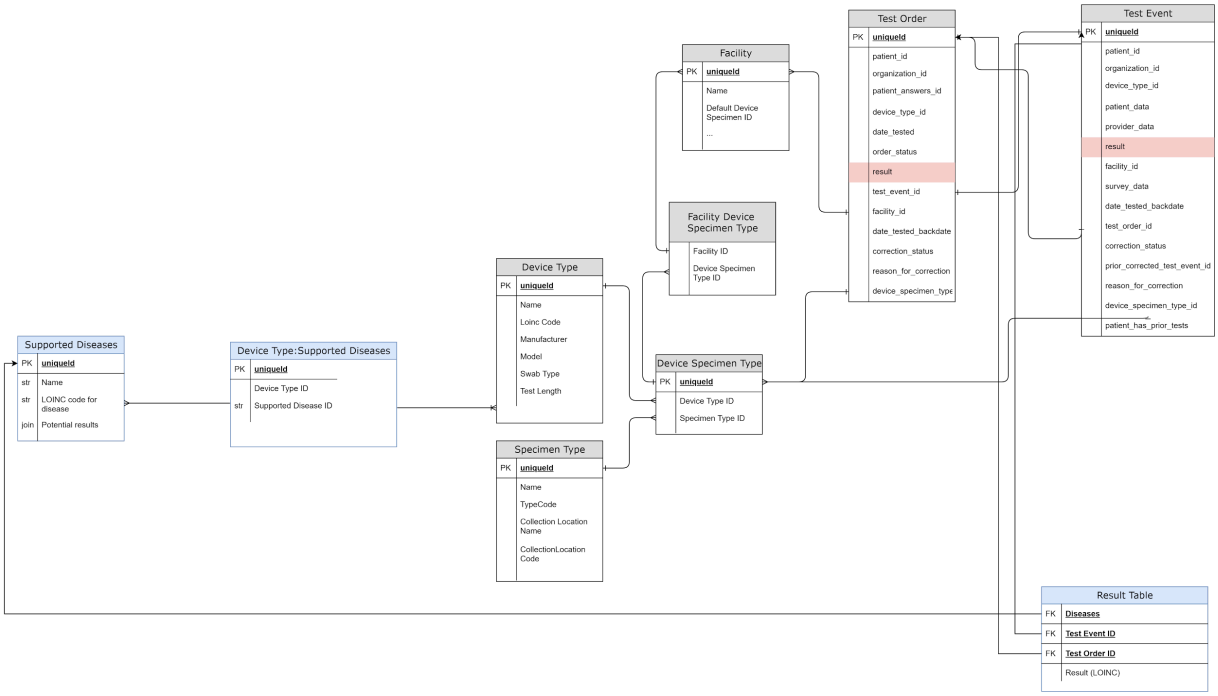
To enforce consistency between the results and the devices used, the results data in a test event/test order will depend on the device used for the test. The supported diseases table, linked to the device table, will limit the results available to the tests performed by the device. This will ensure regular COVID-19 tests can't enter Flu results.

Backfilling

One assumption baked into our existing database is that all devices, tests, and results are for COVID-19. We'll need to backfill several spots in our database to indicate that they were COVID-19 data, or else assume nulls in the database equate to COVID-19 tests.

New Schema Diagram

Blue indicates new/changed fields or tables. The image is clickable if you need to zoom in.



Implementation Plan

Database Changes

- Add a new table with diseases and their potential results, using the LIVD table as a guide. For now, this table only needs to include COVID-19, Flu A, and Flu B. For now, we assume that all diseases have the same potential results (the LOINC codes for positive, negative, or inconclusive.)
- Add a new join table between supported diseases and devices to record a device's supported diseases. A join table is necessary as each device can potentially support multiple diseases.
- Add a new result table, with foreign keys for disease, test event id, test order id, and result (stored as a LOINC).
- Once all the new columns are in place, backfill existing devices and test orders/events to mark them as COVID-19 results. It may be possible to backfill some with a default value instead of a separate SQL execution.

Backend Changes

- Update all the database models and service calls to reflect the new database columns
    - Add a new database model class for the results table
    - Add a new database model class for the supported diseases table
    - Add a new supported diseases column to DeviceType.java
    - Mark the results column in TestEvent.java and TestOrder.java as deprecated
- Once our LIVD-backed disease table is in place, begin reading from ReportStream's LIVD API to update the table.
    - Once the API calls are in place, set up a daily job to read from the API and update the table accordingly.
- Add a lookup table for LOINC codes -> human readable translations. (e.g., 260415000 means "Negative")
- Filter our ReportStream sending logic to only pass along COVID-19 test results from TestEvents
- Update the patient link to show multiple results from the same test order, not just COVID-19
- (Depending on frontend changes) update mutations to pass multiplex results from the backend
- (Depending on frontend changes) update mutations to read multiplex results from the database when necessary, maintaining backwards compatibility.

Frontend Changes

- These will largely depend on the frontend designs, which have not yet been finalized. It's safe to assume that we'll need to update the test cards, results tab, and patient results view to support multiplex testing.

Associated Tickets:

Notes/Scratch

Implementation plan draft:

Database Changes:

- Add a new table with diseases and their potential results (for now, COVID-19: positive, negative, inconclusive; Flu A: positive, negative, inconclusive; Flu B: positive, negative, inconclusive)
- Add a new column on the device table to record diseases. This will likely be a list, or some data structure that supports multiple entries
- Add a new column for results on test events and orders. This will likely be some kind of map, to record multiple results when necessary. A multiplex result might look like

  `{ COVID-19: positive, Flu A: negative, Flu B: negative }`

  while a regular COVID-19 test would look like `{ COVID-19: positive }`.

- For example the Abbott Alinity m preforms the following tests: Flu A (85477-8), Flu B (85478-6), RSV (85479-4), COVID-19 (94500-6) each having the result Positive (260373001) and Negative (260415000) so covid positive, flu a negative, flu b negative and rsv negative would look like

  {“94500-6”: “260373001”, “85477-8”:”260415000”, “85478-6”: “260415000”, “85479-4”: “260415000”}

  while a covid only test would look like {“94500-6”: “260373001”}

- Once all the new columns are in place, backfill existing devices and test orders/events to mark them as COVID-19 results.

Backend Changes:

- Update all the database models and services to reflect the new database reality (Adding a new `result` column to TestEvent.java, for example)
- Filter our ReportStream sending logic to only pass along COVID-19 test results
- Update the patient link to show multiple results from the same test order

- (depending on frontend changes) update mutations to pass multiplex results to the frontend

Frontend Changes:

- Once design is finalized, trigger multiple results input on given device selection
- Other changes depend on final multiplex design

After discussion with Zedd, we propose a combination of proposed solutions 2 and 3, above. Under this solution, we would add a disease table to the database that linked to all the diseases SimpleReport supports (for now, COVID-19 and Flu A & B). We would further add a column to the device table that lists all diseases a device supports. This column would be many-to-one, meaning that a single device can support multiple diseases.

In addition to the device changes, test events will need to change. Rather than recording multiple test orders/events for a single multiplex test, we propose that test orders and events have a new result column that will record the results of all diseases tested. This could take the form of a json or similar data structure, but would look something like { covid result: positive, flu A result: negative, flu B result: negative }. This is slightly verbose, but does have some advantages over using separate test events to record each result. First, all of the other fields in a multiplex test event/order are the same - time of test, symptoms, patient info, etc. Second, as noted above, this hews closer to the testing reality - a single test is being conducted at the patient level, it just has multiple results instead of one.

We believe that this solution will allow us to expand to multiplex tests, with extensibility to test for other second diseases in the future. It should also preserve some core app functionality that we want to avoid touching too much in this change if possible, particularly the patient experience and our ReportStream functionality. Patients will still receive a single text, it will just contain all their results instead of only their covid result. We can also filter test event results to only send COVID-19 results to ReportStream.

backend changes

- reportstream filtering (only send covid results)
- update results view and patient link to show multiple results from same test order

frontend changes

- once design is finalized, trigger multiple result inputs on given device selection

Emma / Zedd 1/20

- patient links are currently linked to individual test orders, but that wouldn't be true for multiplex - how can we change this?
    - link patient links to a specific patient, rather than test event/order - see all your testing history at that facility
- Do we need to link test events together in the database?
    - can use collection date as an approximation, but this is a little dangerous/unsable
    - provides an avenue to group results in the UI
- still need devices to link to multiple diseases
- need a table to list out all the diseases that we support, and the potential results
- link single test event to multiple results
    - prevents duplication of all the other test event fields (tot, device, symptoms, etc etc) and prevents us from needing to group together multiple test events
    - Will work with current patientLink system