Bachelor Thesis

wip

deutscher Titel

Jannis Baum

Supervisors
Prof. Dr. Erwin Böttinger, Tamara Slosarek
Digital Health: Personalized Medicine

Hasso Plattner Institute at University of Potsdam

July 23 2022

Abstract

Reliably and responsibly maintaining health-related informative resources requires human supervision in data administration. One such resource is PharMe: a service aiming at bringing pharmacogenomic findings closer to patients to accelerate its adoption in their treatment. In this thesis, I examine PharMe's approach to data administration and discuss the related complications, namely its high communication complexity, susceptibility to human error, and inaccessibility to the field experts who supervise it. Subsequently, I conceptualize, implement and test a method of solving these complications, and to create the fundamental infrastructure towards providing information with support for multiple languages. The presented method achieves this by interactively modularizing information, and thereby also increases its consistency and efficiency in maintenance and initial curation. The resulting findings facilitate the process of data administration for PharMe while providing relevant insights into its future development and towards establishing solutions to similar problems in other health-related informative resources.

Zusammenfassung

to do.

Contents

1 Introduction

Pharmacogenetics, the study of how individuals respond to drugs based on their personal makeup of specific genes, has been an ongoing research topic since the 1950s. This study was later joined by pharmacogenomics, which, instead of focusing only on specific genes, considers the genome as a whole [1]. Since the difference between these two fields is not relevant to this thesis' research topic, I will be using pharmacogenomics, abbreviated as PGx, as a general term for both fields.

With significant advances in genotyping and genome sequencing technologies, findings in the field of PGx have been rapidly increasing in both numbers as well as evidence, most notably over the last 20 years. Yet, despite the prevalence of applicable PGx related research that could help prevent severe adverse drug reactions, adoption in real-world treatment of patients has been slow [1].

With the goal of accelerating the adoption of PGx into general physicians' consulting rooms, I have worked on a service and research project coined PharMe in a team with seven other students from Hasso Plattner Institute at Prof. Böttinger's chair Digital Health - Personalized Medicine.

PharMe is directed specifically at patients and aims to

1. be an educational resource by introducing users to the existence and relevance of PGx and
2. provide individuals that already have genotyping or genome sequencing results with access to latest PGx research personalized to them through an app on their own smartphone.

To facilitate the latter, we have implemented a backend system, the Annotation Server, as the provider of data the smartphone application displays for users. The Annotation Server's main task is to cumulate data from multiple sources and process it to make it readily available to be retrieved by the app.

Aside from data the Annotation Server fetches from preexisting APIs of external medical organizations, it needs to provide guidelines that are comprehensible to users without professional medical education, such as patients; PharMe's primary target group. To provide these guidelines, we work with PGx experts from the Icahn School of Medicine at Mount Sinai who manually curate them.

With how we have built the Annotation Server, some unresolved issues have arisen. The data the Annotation Server fetches from external sources has to be matched into a common database, which can fail in ways that can't be resolved automatically. This creates a need for human supervision. On top of this, the PGx experts need an efficient way to curate and administer the guidelines they provide, and these in turn also have to be matched with the external data.

In this thesis, I will give an overview of the Annotation Server we have built in shared effort over the course of our project. After explaining the before mentioned issues in more detail, I will propose, implement and test a solution towards facilitating the process of curating and administering patient-oriented PGx guideline data. Aside from attempting to solve this issue, the presented solution will also look into how PharMe's guidelines can be provided with multi-language support in a sustainable and efficient manner the future.

2 Annotation Server

As the data providing backend, the Annotation Server is one of PharMe's core components. With how we have conceptualized PharMe's patient-oriented frontend, the smartphone application, the Annotation Server needs to provide two main groups of information:

1. Drugs - users should be able to search for and find any drug they want to consult PharMe about; independent of whether there actually are any PGx findings for this drug.
2. Guidelines - for drugs that do have PGx findings, users should be presented with
    a) simplified information that is comprehensible for them and
    b) more detailed and complete information they can show to their doctor.

To provide information on drugs (1.), we use the academic license to a database called DrugBank. DrugBank is one of the world's most used resources for drug information [2]. DrugBank has provided us with a large XML file that contains all the information we use from them, namely drugs' names, descriptions, synonyms and their RxCUIs: unique identifiers given by the National Library of Medicine's standardized drug nomenclature RxNorm [3]. Additionally, the PGx experts working with us annotate drugs that have pharmacogenomic relevance with descriptive texts made to be easily comprehensible for patients.

Guideline information (2.) is first fetched from the public API of the Clinical Pharmacogenetics Implementation Consortium (CPIC). CPIC provides "peer-reviewed, updated, evidence-based, freely accessible guidelines for drug/gene-pairs" [4] targeted at professionals in the field. This data is used in the case of 2.b) and directed at doctors. For 2.a), we again rely on the PGx experts working with us to annotate CPIC's guidelines with patient-friendly wording.

2.1 Technical overview

The Annotation Server is a web application made with the framework NestJS, which is widely known for its efficiency, scalability and full support of TypeScript [5] - the programming language the majority of PharMe's backend is built with. For storage of data, the Annotation Server relies on the reliable and performant relational database PostgreSQL [6] and the object relational mapping TypeORM to integrate PostgreSQL into the NestJS application.

There are three main modules to the Annotation Server: `medications`, `phenotypes` and `guidelines`.

The `medications` module is responsible for all data regarding drugs. It stores this data by maintaining a drug repository in the Annotation Server's database. This repository

is initialized by loading and saving all relevant data from DrugBank's XML file upon invocation of a POST endpoint the module provides. To simplify processing this large XML file's data, it is first transformed JSON format with the help of a Python script. The JSON format makes it easier for the TypeScript-based web application to process the data. Once the drug repository is initialized, the Annotation Server provides GET endpoints to retrieve information for one or multiple drugs along with the option of applying filters.

The `phenotypes` module maintains all the phenotypes CPIC offers guidelines for in its phenotype repository. These phenotypes are defined by a gene symbol such as *CYP2D6* and the effect variants with this phenotype have on the gene, i.e. a gene result such as *Normal metabolizer*. Aside from these identifying properties, some additional data CPIC provides about phenotypes is also stored. The `phenotypes` module exposes no dedicated endpoints as it is only used in relation to the `guidelines` module. The loading of the phenotype repository's data from CPIC's API is invoked by the `guidelines` module when it initializes its own data.

The `guidelines` module keeps guidelines in relation to phenotypes and drugs in its guidelines repository. This repository's data is initialized by invoking the POST endpoint the module provides, which triggers loading all of CPIC's guidelines from its API. Since inconsistencies between CPIC and DrugBank may occur, matching errors are tracked in a separate `GuidelineError` repository to be resolved by the maintainer. The `guidelines` module provides GET endpoints to retrieve guidelines and is connected to the `medications` module to allow fetching guidelines along with the drugs they describe.

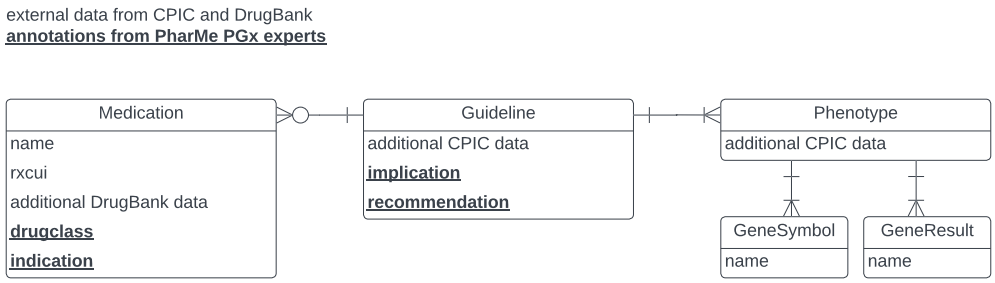

Figure 2.1: Simplified ER-diagram of Annotation Server database

Both the `medications` and `guidelines` modules expose PATCH endpoints to annotate additional data provided by the PGx experts working with us. These annotations are made to be easily comprehensible for patients, i.e. people without professional medical education, and consist of

- a drug class and an indication for drugs and
- an implication, explaining the effect a phenotype has on an individual's response to a drug, a recommendation, giving a suggestion based on the implication's consequences, and a warning level, expressing the severity of the recommendation as one of three tiers, for guidelines.

With how the Annotation Server has been set up by us within the context of our team-based project, the PGx experts provide this data through a shared online Google Sheet.

On request, the Annotation Server automatically downloads and processes this Google Sheet to annotate all data that matches the existing external data.

See figure 2.1 for a simplified overview of the Annotation Server's database and its sources.

2.2 Evaluation and discussion

The illustrated implementation of the Annotation Server relies on two parties to initialize its data and keep it up-to-date:

- **A curating party** with sufficient pharmacogenomic expertise to curate patient-oriented annotations from data they manually research from sources such as CPIC. This party manually writes their annotations into the Google Sheet, initially without any feedback of if and how well they match the Annotation Server's existing external data.
- **A maintaining party** with sufficient technical knowledge to invoke the requests that trigger fetching data from external sources and the Google Sheet. This party also oversees the before mentioned `GuidelineErrors` and acts accordingly, which usually results in notifying the curating party to make necessary adjustments.

This separation leads to some difficulties in the Annotation Server's operation and creates an overhead of communication between the two parties.

Initial loading or reloading (updating) of data should occur when and only when new CPIC or DrugBank data is available or when the curating party has modified the Google Sheet, since there is a downtime users will experience during this process. Triggering the process is therefore always initiated by the curating party, who however can't trigger it themselves and have to notify the maintaining party to do it.

The Google Sheet is a one-way interface to the Annotation Server, yet it is the curating party's only interface other than communication with the maintaining party. Since the Sheet is set up completely manually, it is subject to human error, such as misspelling of a drug's name, which leads to a mismatch with the Annotation Server's existing data upon import. These human errors of the curating party have to then be detected by the maintaining party, who have to notify the curating party to resolve them, only for the maintaining party to trigger a reload that, again, leads to downtime users will experience. This communication-heavy process needs to be repeated until all errors are resolved.

During our development phase, we were working as the maintaining party whilst the curating party was formed by pharmacogenomic expert Dr. Aniwaa Owusu Obeng, who tasked two of her students with curating annotations for roughly 100 drug-gene pairs and oversaw the process herself. According to Dr. Owusu Obeng, her students spent two days curating these annotations which equated to around 25 to 30 hours of work for the initial curation process, without accounting for the subsequent feedback loop. A significant part of the time spent during this process was to research of data from CPIC - the same data the Annotation Server already had stored and that later had to match the manually curated data with again.

Should PharMe be extended to support multiple languages in the future, this process would most likely need to be repeated with multiple curating parties.

3 Annotation Interface

Concerning the main research aim of this thesis, I will propose, implement and test a new component of the PharMe system: the Annotation Interface. With the Annotation Interface I will attempt to solve the issues discussed in chapter 2.2 by improving the overall experience and efficiency of the process of curating annotations and maintaining the Annotation Server's data.

The core idea of the Annotation Interface is to give full control over data to the curating party, eliminating the communication overhead and the need for a second party involved in maintenance of data. On top of this, I will conceptualize and partly implement automation into the curating party's research and curation process to further increase efficiency. Finally, I will implement an approach towards operating PharMe with support for multiple languages by modularizing annotations.

3.1 Conceptualization

On its surface, the concept I have devised for the Annotation Interface closes the gap between the curating party and the Annotation Server by providing the missing infrastructure to allow them to create, view, update and delete their annotations live on the Annotation Server through a web interface. This moves the curated annotations from being external data on a Google Sheet to becoming internal data that is inherently created on top of the data from external sources. The discussed problem of matching manually curated annotations with external data from CPIC and DrugBank therefore effectively disappears.

The problem of matching data between external sources, i.e. matching CPIC's guidelines with DrugBank's drugs, does persist, however it is not susceptible to human error from PharMe's side and the resulting matching errors can be shown directly to the curating party through the Annotation Interface instead of first going through a second involved party.

With the Annotation Interface displaying the Annotation Server's data, my concept also includes showing the data that is already stored from CPIC and linking to their additional resources. Having this information accessible right alongside the annotations the curating party manages has the potential to reduce the time they have to spend on research making the overall process more efficient.

The second major feature of my concept for the Annotation Interface revolves around modularizing annotations. I have discussed this idea with Dr. Aniwaa Owusu Obeng, an expert in the field of pharmacogenomics, who has shown high enthusiasm for the concept.

This modularization is achieved by strictly limiting the creation of annotations to combinations of Text Bricks: predefined textual prototypes or templates that adjust to the

annotation they are used in by replacing placeholders with data matching the given annotation. One such Text Brick might be

> `#drug.name` may not be the right medication for you.

where `#drug.name` will be replaced with the name of the drug the Brick is annotating. Constraining annotations to be made up of combinations of Text Bricks brings an array of benefits.

Having a finite number of options in creating annotation texts ensures consistency between annotations without needing to copy and paste text or risking human errors such as misspelling. This is particularly relevant when there is more than one person curating annotations. The limited options also make the curation process more straightforward once the initial Text Bricks have been defined which has the potential to reduce time requirements.

The full body of annotation texts being made up of a finite number of Text Bricks also simplifies support for multiple languages. A language expert can be consulted once to translate all Bricks into the respective language. With Bricks defined in this language, the full body of annotation texts is consequently also translated and the curating party can create and update annotations for this language without having to know it themselves; simply by creating them in their own language.

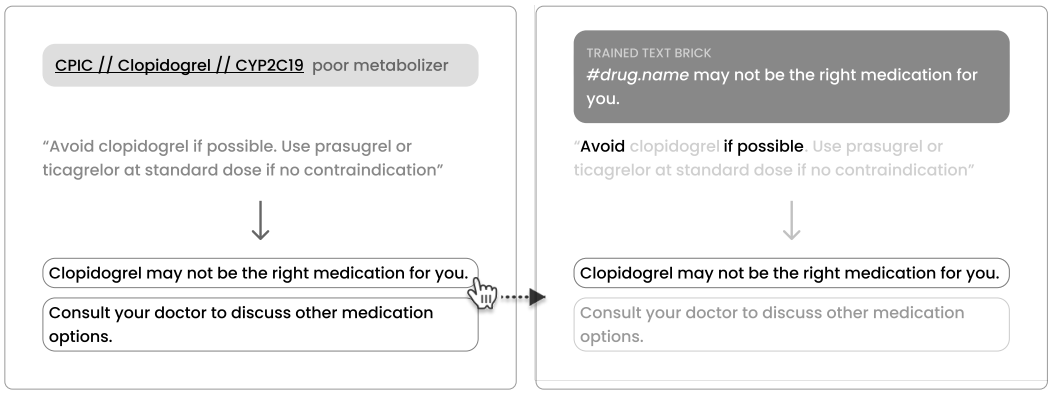

Figure 3.1: Conceptualized suggestion of Text Bricks based on CPIC guideline [7]

Finally, my concept proposes employing techniques of natural language processing for the picking of suitable Text Bricks based on the corresponding CPIC guidelines. With a model trained to infer matching Text Bricks based on external guidelines, curation time requirements can be reduced further by providing the curators with automated suggestions. This reduces their task to accepting or editing suggestions. Figure 3.1 shows a mock-up of a user interface with the natural language processing model's suggestion of trained Text Bricks based on a CPIC guideline. In this example, the user clicks on the first Text Brick to get a visual explanation of why it was suggested.

3.2 Implementation

In consultation with Dr. Aniwaa Owusu Obeng I prioritized the presented concept's features and implemented a minimal viable version of the Annotation Interface in the context of this thesis.

To keep separation of concerns and between our group project and my thesis project, the Annotation Interface is implemented as its own web application that is distinct from the Annotation Server. Like the Annotation Server, it uses TypeScript as its main programming language, but relies on a different technology stack due to its different requirements.

The Annotation Interface is built using Next.js, a framework for creating web applications based on the frontend JavaScript library React. It has primarily been tested using Apple's web browser Safari on macOS, and, in it's minimal implementation, does not strive to provide responsiveness for small screens.

To store Text Bricks and the annotations they make up, the Annotation Interface has its own MongoDB document database and uses the library Mongoose for object modeling.

3.2.1 Usage flow and technical overview

The Annotation Interface's implementation can be found in PharMe's GitHub repository[1] under the root directory `annotation-interface`. All mentions of source code files will assume this directory as their base.

The Annotation Interface is made up of three main groups of web pages: *Home*, *Annotations* and *Bricks*, which are accessible through a navigation bar on the left side of the interface. Navigation between these groups is handled by the `Layout` React component defined in `components/common/Layout.tsx`. The `Layout` component wraps all pages of the Annotation Interface and automatically builds the navigation bar, highlights the currently active page group and manages persistent state within and between page groups with the help of React Contexts defined in the `contexts` directory. This automation is based on regular expressions that are matched against the pages' routes and is defined by `tabDefinitions` in the same file making it easily adjustable and extensible in case more pages or page groups are added.

All main pages of the Annotation Interface include a small description in the top of the page giving general information or hints about what this page does. These descriptions are meant to speed up the user's process getting familiar with the interface and with PharMe's terminology.

3.2.1.1 *Home* page group

Upon first opening the Annotation Interface, the user is presented with an overview of the status of the Annotation Server's external data, i.e. data from CPIC and DrugBank, on the *Home* page. This overview includes information on when the data has most recently been updated. The dates for these updates are retrieved from the Annotation Server through the Annotation Interface's API and with the cache control technique stale-while-revalidate [8]. Using this technique, the dates are revalidated to be kept up-to-date with the Annotation Server in case changes occur.

[1] github.com/hpi-dhc/PharMe

Aside from checking when the last update has occurred, the user has the option to trigger a new update, which sends the necessary request to the Annotation Server. Since triggering the update essentially resets all data on the Annotation Server, the Annotation Interface afterwards uses the Annotation Server's described PATCH API to (re-)upload all of the annotations curated and stored here.

Before the buttons to trigger updates become visible, the interface's visual design radically changes by having a dark themed overlay placed on top of its usual light theme. This aims to regain the user's active attention and signal that they can now make modifications to the live data on the Annotation Server, which may affect users. The dark themed overlay is used in all parts of the Annotation Interface that have the ability to make such modifications. Figure 3.2 shows the Annotation Interface's *Home* page along with the dark themed overlay that allows to trigger data updates.

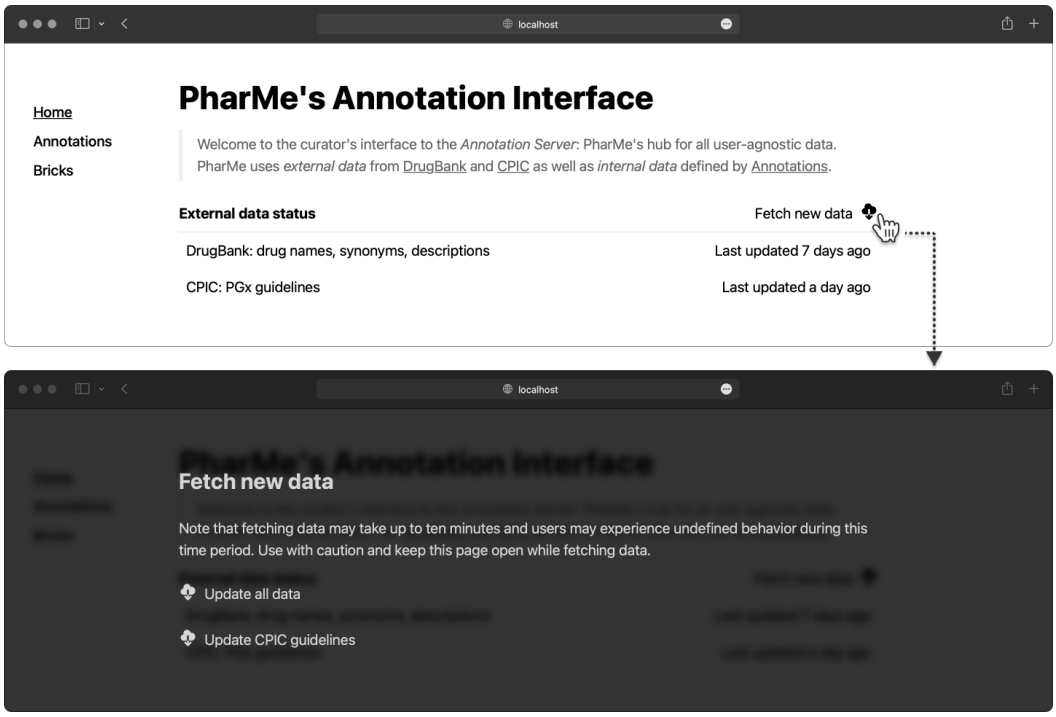

Figure 3.2: The Annotation Interface's *Home* page

3.2.1.2 *Bricks* page group

By navigating to the *Bricks* page group, the user can view, create, modify and delete Text Bricks in multiple languages. The initially visible Brick overview offers filtering by the Bricks' category, and setting the language they are displayed in. The category filter and selected display language states are stored using the before mentioned React Contexts provided through the `Layout` component and are thereby persistent across navigation within the *Bricks* page group.

When clicking on an existing Brick to edit it, and when adding a new one, the user can define translations for the Brick by writing into a text field. Alongside of free text, Bricks

feature placeholders: textual components that adjust to the annotation the Brick is used in, such as the respective drug's name. To facilitate working with placeholders when editing or creating Bricks, the text fields used here feature automatic completion, suggestion, and highlighting of placeholders; similar to what one might expect from modern code editors: Valid placeholders within the Brick are highlighted with an underline, and a menu listing matching placeholders is displayed right under the user's cursor when detecting it is within the context of a placeholder or the user is about to add one. The menu is navigable with the arrow keys, and the selected option can be picked using the return key.

Figure 3.3 shows a Brick's English translation being edited with placeholders. The finished version of this Text Brick might, for example, be resolved to

> As a CYP2C9 normal metabolizer you can use clopidogrel at standard doses.

when used in the context of a guideline for the drug *clopidogrel* and the gene *CYP2C9* with the result *normal metabolizer*.

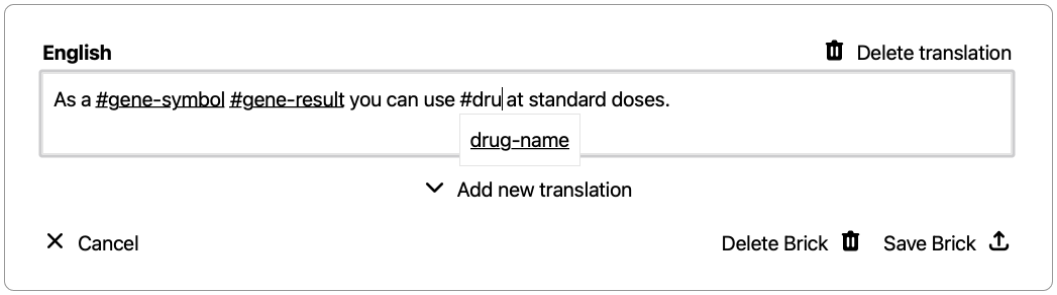

Figure 3.3: Editing a Brick with placeholders

The highlighting, automatic completion, and suggestion are implemented without any additional external dependencies in the components `AutocompleteArea` and `AutocompleteMenu` found in the `components/bricks` directory. The underlying strategy is an HTML `<div>` element that is aligned, synchronized with and placed underneath a transparent `<textarea>` element. The `<div>` element provides highlighting and the completion menu as children, while the `<textarea>` makes the text editable.

3.2.1.3 *Annotations* page group

The Annotation Interface's final page group, *Annotations*, provides the ability to manage PharMe's internal patient-oriented annotations made out of Text Bricks. Through this page group, the user is given access to all the data present on the Annotation Server along with filtering options. The initial data overview is split up into entries for drugs, and entries for guidelines. The entries presented here are akin to the guidelines CPIC offers and the drugs corresponding with them, and are shown with labels indicating which entries are missing which types of annotations.

Supplementary to selecting the category *drugs* or *guidelines*, the user can choose to filter for entries that are missing annotations or entries that already have all patient-friendly annotations defined, i.e. are fully curated. On top of this, there is a search bar to filter for entries matching the search query. All filtering as well as the selected category are,

analogously to filtering states in the *Bricks* page group, stored using the React Contexts provided through the `Layout` component, which makes them persistent across navigation within the *Annotations* page group.

By clicking on one of the entries, the user is taken to a page detailing its corresponding annotations, i.e. drug class and indication for a drug entry, and implication, recommendation and warning level for a guideline entry. Detail pages for guidelines additionally feature the data CPIC provides for the corresponding drug-phenotype pairing as well as a link to CPIC's respective publication. This aims to streamline the research and curation process.

When pressing the button to edit an annotation, the before mentioned dark themed overlay is used again to signal that changes to data on the Annotation Server can be made here. This overlay provides a drag-and-drop based interface to modify, create or delete the given annotation using the Text Bricks defined for this type of annotation. Figure 3.4 shows a recommendation annotation being edited in the overlay by dragging a second Brick onto the annotation canvas.

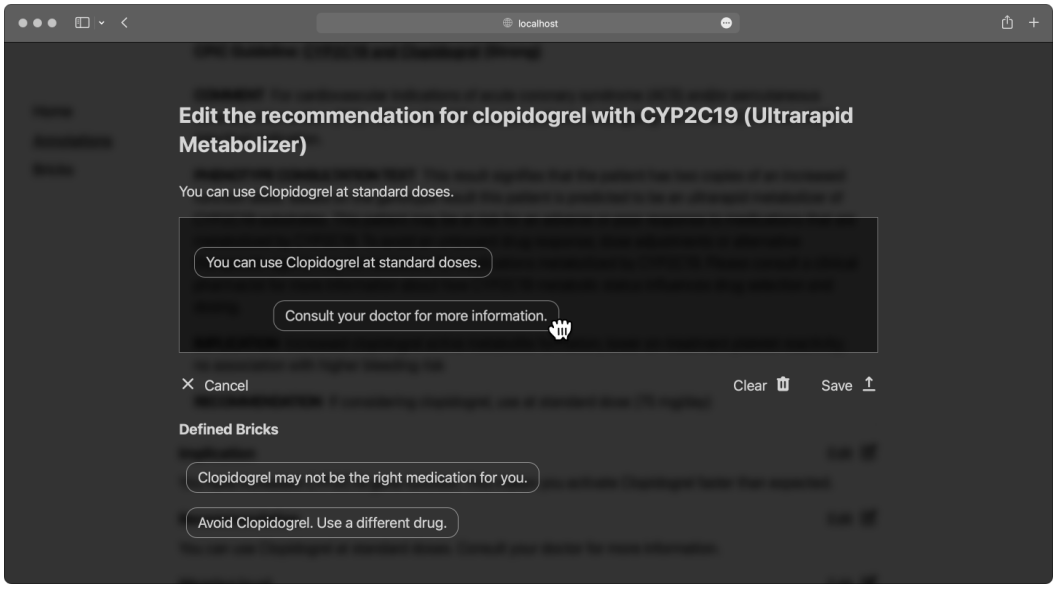

Figure 3.4: Editing a recommendation annotation

Saving an annotation from this overlay calls the Annotation Interface's API which

- saves the annotation in the Annotation Interface's database as a combination of references to saved Text Bricks and
- calls the Annotation Server's API to save the text the Bricks resolve to, visible above the annotation canvas in figure 3.4.

This enables live editing of Annotation Server data and keeps it synchronized with the annotations stored on the Annotation Interface.

3.2.2 Data structure

The Annotation Interface uses a MongoDB document database along with Mongoose for object modeling in TypeScript. All documents the Annotation Interface stores in the database are typed with the help of TypeScript interfaces that extend the base interface `IBaseModel` found in `database/helpers/types.ts`. This base interface provides an optional property `_id` that is used to reference documents in the database. The `_id` property is generically typed to be a Mongoose `ObjectId`, a `string` or `undefined`, which allows the same document interfaces to be used

- when a document is created or retrieved on server side, i.e. having an `ObjectId`,
- when a document is sent to the client side with a serializable `string` instead of an `ObjectId` as an `_id`, and
- when data for a new document is created on the client side, i.e. without an `_id`.

The Annotation Interface's database is used to store Text Bricks with support for multiple languages, and annotations as combinations of references to Bricks. The Mongoose models for all database documents are implemented in the `database/models` directory and use Mongoose's builtin techniques for validation.

To ensure unambiguous matching, the annotations stored on the Annotation Interface need to injectively map to data on the Annotation Server. This is achieved by storing the corresponding drug's RxCUI in the documents for all annotations and additionally storing the corresponding gene symbol and gene results for guideline annotations.

The documents that describe annotations for drugs and guidelines are modeled by `MedAnnotation` and `GuidelineAnnotation` respectively. These models inherit from `AbstractAnnotation` for their common properties and validation functions.

Since this is a commonly used operation, both `MedAnnotation` as well as `GuidelineAnnotation` implement a static method that finds the document matching with the given Annotation Server entity data. Aside from the discussed properties that are necessary for this matching to be unique, `MedAnnotation` models the properties `drugclass` and `indication`, and `GuidelineAnnotation` models `implication` and `recommendation` as arrays of references to `TextBricks` for the Brick-based annotations. These references are again generically typed as `ObjectIds` or `strings` to be usable on both server and client side. Additionally, `GuidelineAnnotation` models the three-tier warning level, the only non-Brick-based annotation, as a `string` with three possible values.

Documents for Text Bricks are modeled by `TextBrick`. Each Brick document has an assigned `BrickUsage`, specifying what type of annotation it can be used in, as well as an array of translations defined by their language and text.

The common operation of *resolving* a Brick, i.e. filling its placeholders with information for a specific annotation, is implemented by the `resolveBricks` function defined in `database/helpers/resolve-bricks.ts`. This function takes an array of `TextBricks` of any `_id` type, a target language, and a generic source `BrickResolver`. The `BrickResolver` can be drug or guideline data from either the Annotation Interface or the Annotation Server, and is used to fill out the `TextBricks`' placeholders accordingly to return their resolved texts. This function's flexibility makes it usable on both client and server side, with any type of source resolving data and makes it easily extensible for new placeholders as well as new types of resolving data.

Since finding `TextBricks` and then resolving them is also a common operation, the `TextBrick` model implements this as a static method by wrapping the `resolveBricks` function.

3.2.3 Abstraction and extensibility

To illustrate some of the Annotation Interface's architectural abstraction and its resulting extensibility, as well as to give deeper insights into its code base, I will use this section to walk through how

1. a new category of Text Bricks with new placeholders, and
2. the visual interface to display and edit a new type of annotation

could be implemented. These steps, along with adjustments to the API and database models analogous to what is already implemented, are what would make up the requirements towards implementing a new type of annotation on the Annotation Interface.

3.2.3.1 Text Bricks and placeholders

The first step towards adding the new category of Text Bricks is handled by adding the new category's displayed name to the array of Brick categories (*usages*) in `common/constants.ts`. This will automatically enable the `TextBrick` database model and corresponding API to process the new Bricks, and adjust the interfaces in the *Bricks* page group to allow filtering for the new category, as well as adding and editing Text Bricks that use it.

The only aspect that is left to be handled now is defining what placeholders the new Bricks can use. This is achieved by adjusting the function `placeholdersForBrick` found in `database/helpers/resolve-bricks.ts` to return the possible placeholder strings accordingly. This file is also where new placeholders that are resolved with new or existing types of source data can be defined. To enable resolving placeholders from a new type of source data, its type is added to the type union `BrickResolver` and its mapping to the placeholders is defined in the function `getPlaceholders`.

With placeholders defined, the newly implemented Text Bricks can be created, edited, used and resolved analogously to the existing ones.

3.2.3.2 Visual annotation components

The interfaces that display an annotation's current value and provide the editing overlay shown in figure 3.4 are built with multiple levels of abstraction. This abstraction makes the underlying React components reusable for flexible applications. All components referred to in this section are implemented in the `components/annotations` directory.

The most abstract of these components is `AbstractAnnotation`. This component is responsible for displaying an annotation's current value, or, if appropriate, a label stating it is missing, as well as for implementing showing and hiding the editing overlay visible in 3.4, and creating the overlay's basic structure. `AbstractAnnotation` provides a React reference that enables its parent component to hide the overlay as well and places its own children on the overlay.

This component is used to implement the interface for a new type of annotations that is not based on Bricks, such as the existing warning level annotation in `WarningLevelAnnotation.ts`.

The components for Text Brick-based annotations feature additional levels of abstraction. `AbstractBrickAnnotation` component uses `AbstractAnnotation` to implement the interface seen in 3.4 by adding the drag-and-drop functionality based on information it is given about the Bricks to use. It also adds buttons to confirm or cancel changes using a given callback function.

The next level of abstraction is implemented by the `BrickAnnotation` component, which sets up `AbstractBrickAnnotation` based on given Annotation Server data and an Annotation Interface API endpoint, by making assumptions about the structure of both. `BrickAnnotation` is the component used by the final `MedAnnotation` and `GuidelineAnnotation` components, which are used to create the interfaces for all other existing types of annotations.

Finally, the new type of Brick-based annotation is implemented using `BrickAnnotation` or `AbstractBrickAnnotation` depending on whether its data structure and API follow the assumptions made by `BrickAnnotation`.

3.3 Testing with an expert

To test the Annotation Interface's usability and how it meets its goals of improving the process of curating and researching annotations for PharMe, I set up a second consultation with an expert of pharmacogenomics. For this consultation, I again talked to Dr. Aniwaa Owusu Obeng, who I also interviewed for the Annotation Interface's initial conceptualization and feature prioritization.

3.3.1 Setup

The consultation was set up as a roughly hour-long video call and initially accompanied by presentation slides. To start off, I briefly reintroductioned what the Annotation Interface is and aims to accomplish, and explained how it fits into the PharMe system.

The main part of the consultation revolved around use case testing. Instead of merely presenting the application to them, I chose to have the expert use the Annotation Interface in real scenarios to get more comprehensive feedback and explore aspects that might otherwise have been missed.

Before introducing the testing scenarios, I explained the aims, focus and process of the testing and asked the expert to

- think out loud and mention anything that is unclear, and
- focus on fundamental functionality, rather than suggestions to improve the visual interface

during the testing process. In addition, I reminded them that the testing is not meant to test them, but the Annotation Interface as an application, to ensure a comfortable environment.

The testing consisted of the expert working through the following four scenarios that I chose to reflect real use cases with increasing complexity and to progressively disclose the Annotation Interface's features:

1. Check when CPIC's guidelines have last been updated on the Annotation Server and update them.
2. Find the existing patient-friendly drug class annotation for a specific drug.
3. Create a new recommendation Text Brick that uses a placeholder.
4. Create an annotation with the new Text Brick.

During and after the testing we discussed various aspects of the Annotation Interface and how it might evolve in future versions.

3.3.2 Results

The PGx expert perceived the Annotation Interface as overall "very user-friendly" and a "great annotation tool", and learned the user interface quickly and without apparent confusion. They mentioned liking the display of CPIC's data, explicitly stating this would avoid the discomfort of "having to use two screens" in the curation and research process, i.e. one with the annotation tool and one with the corresponding sources. In this context, the expert also expressed it would be useful to additionally include findings from sources other than CPIC, such as publications of the Dutch Pharmacogenetics Working Group (DPWG), another resource for pharmacogenomic guidelines [9].

On their first contact with Text Bricks, the expert quickly made me aware of their perceived similarity between Bricks and a system they were already familiar with. This system is made by the company Epic and is, for example, used by physicians to create clinical notes based on so called *SmartPhrases*: templates with placeholders for data such as a patient's information, that are automatically filled out with respect to the patient they are used for. According to the expert, it is likely that a user of the Annotation Interface would already be familiar with Epic's concept of templates.

Additionally, the expert mentioned they would like having a way of creating names for Text Bricks or having an automatic identifying number assigned to them for easier referencing, explaining that someone working with the Annotation Interface will memorize the numbers over time. This reference could then be used to efficiently find Bricks when creating annotations.

While working with the interface to edit annotations (as depicted in figure 3.4), the expert remarked that a search bar should be incorporated where the draggable Text Bricks are listed. They explained this would to make it easier to find Bricks by their content or the name or number introduced above, saying that a search bar would be essential in keeping the work flow efficient once a large number of Bricks has been defined.

During the final part of the video call, the expert and I discussed various features that would be needed if PharMe was to be taken into a real production setting with real users. These features include

- Seeing a log of what CPIC guidelines have changed before updating them and having the option to update only specific guidelines.

- Keeping a revision history of all data that is presented to users. The expert emphasized the need for this by explaining the scenario of receiving a complaint about data from a specific day.
- Implementing staging and reviewing phases into the administration of annotations. This would allow for staged data to be reviewed by other curators before making it onto the live application where it is presented to users.
- Providing curators with a summary page of all data that will be presented to users.

Concluding our discussion, the expert emphasized the Annotation Interface's value to the curation and research process of a service like PharMe, emphasizing on its efficiency and support of multiple languages, and strongly encouraged its further development.

4 Discussion

According to PGx expert Dr. Aniwaa Owusu Obeng, the Annotation Interface in its current state meets its research aim of facilitating the research and curation process of patient-oriented annotations for PharMe. In addition, it provides crucial infrastructure towards making PharMe sustainably available in multiple languages. I have implemented the Annotation Interface for modularity and flexibility, making it provide relevant requirements towards maintainability and extensibility for future use and development.

Beyond the scope of what I have implemented in the context of this thesis, there is sizable room for further development of both the Annotation Interface, as well as PharMe as a whole. A list of the next minor features and changes to implement into the Annotation Interface can be found within PharMe's GitHub issues labeled `P: Annotation Interface`. This list, for example, includes the naming of and searching for Text Bricks that was suggested by the expert while testing the application in chapter 3.3. In the following, I will go into detail about future work beyond these minor features and changes.

4.1 Future work

A major feature I conceptualized for the Annotation Interface in chapter 3.1 is to employ techniques of natural language processing to provide suggestions for which Text Bricks to use for an annotation based on data from an external source such as CPIC. This feature has been left out of the scope of this thesis, since creating the now existing infrastructure was necessary first to make the automated suggestions viable and useful. The PGx expert I consulted to conceptualize, prioritize, and test features of the Annotation Interface has expressed high interest in this feature, yet prioritized it lower than the other features I will explain hereafter. Should automated suggestions be implemented in the future, related work such as PGxCorpus [10] may be useful to leverage for training the underlying natural language processing model.

Regarding future work around the Annotation Interface's user experience, the similarity between Text Bricks and Epic's *SmartPhrases* that was recognized in chapter 3.3 should be taken into consideration. A user who is already familiar with *SmartPhrases* may benefit from this similarity while learning the Annotation Interface and its concepts.

With the current state of PharMe as a system, the infrastructure the Annotation Interface provides towards supporting multiple languages does not yet contribute value to PharMe, since it is only available in the creation and administration of Text Bricks. Implementing the support of multiple languages into the other parts of the PharMe system, and connecting them to what the Annotation Interface already provides, is a feature highly prioritized by the consulted PGx expert.

Similarly to how PharMe currently supports only one language, it leverages only one external pharmacogenomic resource. To further assist the research process of curating

annotations, implementing additional sources such as DPWG is another high priority feature for the Annotation Interface as well as PharMe as a whole. With PharMe offering additional data from more sources, the Annotation Interface also gains additional value as the system that is used to manage possible conflicts or inconsistencies between the sources.

Lastly, as presented in the final section of chapter 3.3, there is a gap in functionality that needs to be bridged before PharMe could be taken into a real production setting with real users relying on it. Currently, all data modifications made through the Annotation Interface immediately appear on the Annotation Server and are thereby available for users. This is, as also brought up by the consulted PGx expert, not viable in a production setting - especially for a health service like PharMe. To avoid human error and provide the crucial layer of certainty in correctness of the information that is presented to users, the production-related features explained in 3.3 are of highest priority if PharMe is to be taken into such a setting.

4.2 Boundary between Annotation Server and Interface

During our project, we worked on the Annotation Server throughout various changes of requirements and of how PharMe provides its users with data. Development of the Annotation Server began long before the need for human data administration that the Annotation Interface now handles was recognized.

To preserve the team's existing knowledge about its architecture as well as the Annotation Server's ability to work without the Annotation Interface, and to allow myself to work independently for the context of this thesis, I developed the Annotation Interface as a second and entirely distinct web application. This enabled me to explore the various features and concepts I examined in this thesis.

Now that requirements and features for PharMe's data providing backend are much more clear, I strongly encourage considering rebuilding the Annotation Server and Annotation Interface as one joint web application if PharMe is to be developed further in the future. The argument for this becomes especially clear when considering the features I discussed in the previous section. Implementing them would, with the current boundary between the Annotation Server and Interface, inherently lead to building the same concepts, such as the data structure to support multiple languages, into both systems. I regard avoiding the resulting repetition and complexity as vital in keeping maintainability and extensibility of the system as a whole.

5 Conclusion

This thesis explored how to support field experts in their process of researching, curating and maintaining informative pharmacogenomic data, as well as how to provide this data in multiple languages. The presented methods focus on the service PharMe: a patient-oriented resource for PGx information.

Within this context, I explained PharMe's preexisting approach to data administration and discussed the related complications, namely its high communication complexity, susceptibility to human error, and inaccessibility to the field experts responsible for it. Subsequently, I conceptualized, implemented and tested the Annotation Interface as a method of solving these complications while creating the fundamental infrastructure towards multi-language support. I conducted this work in consultation with an expert in the field of pharmacogenomics.

In its presented state, the Annotation Interface fulfills its aim of facilitating the research and curation process of patient-oriented information for PharMe. This is primarily achieved by making data administration interactive and modularizing the information, which, while increasing its consistency, maintainability and curation efficiency, also lays the foundation for the desired multi-language support.

By testing the Annotation Interface with the consulted PGx expert, I have verified its efficacy at meetings its research aim and gained further insights into what capabilities for data administration would be crucial if PharMe was taken into a production setting with real users relying on it. Finally, I illustrated the focus and key aspects of future work on the Annotation Interface and PharMe's data administration as a whole based on these insights and the previously made findings.

To reliably and responsibly provide users with data through health-related informative resources such as PharMe, the need for human supervision in data administration is apparent. The Annotation Interface as a system and as its broader concept provides an approach to attend this need for PharMe. The findings I have made within the context of research for and around the Annotation Interface comprise relevant insights towards establishing systems to solve similar problems for other health-related informative resources.

References

1. Scott, S.A. (2011). Personalizing medicine with clinical pharmacogenetics. Genetics in Medicine *13*, 987–995. Available at: https://linkinghub.elsevier.com/retrieve/pii/S1098360021036121 [Accessed July 10, 2022].

2. Wishart, D.S., Feunang, Y.D., Guo, A.C., Lo, E.J., Marcu, A., Grant, J.R., Sajed, T., Johnson, D., Li, C., Sayeeda, Z., *et al.* (2018). DrugBank 5.0: A major update to the DrugBank database for 2018. Nucleic Acids Research *46*, D1074–D1082. Available at: http://academic.oup.com/nar/article/46/D1/D1074/4602867 [Accessed July 10, 2022].

3. Liu, S., Wei Ma, Moore, R., Ganesan, V., and Nelson, S. (2005). RxNorm: Prescription for electronic drug information exchange. IT Prof. *7*, 17–23. Available at: http://ieeexplore.ieee.org/document/1516084/ [Accessed July 10, 2022].

4. Relling, M.V., and Klein, T.E. (2011). CPIC: Clinical pharmacogenetics implementation consortium of the pharmacogenomics research network. Clin Pharmacol Ther *89*, 464–467. Available at: http://doi.wiley.com/10.1038/clpt.2010.279 [Accessed July 10, 2022].

5. Mysliwiec, K. (2022). NestJS. NestJS. Available at: https://nestjs.com [Accessed July 11, 2022].

6. Group, T.P.G.D. (2022). PostgreSQL. PostgreSQL. Available at: https://www.postgresql.org/ [Accessed July 11, 2022].

7. Lee, C.R., Luzum, J.A., Sangkuhl, K., Gammal, R.S., Sabatine, M.S., Stein, C.M., Kisor, D.F., Limdi, N.A., Lee, Y.M., Scott, S.A., *et al.* (2022). Clinical pharmacogenetics implementation consortium guideline for *CYP2C19* genotype and clopidogrel therapy: 2022 update. Clin Pharma and Therapeutics, cpt.2526. Available at: https://onlinelibrary.wiley.com/doi/10.1002/cpt.2526 [Accessed July 12, 2022].

8. Nottingham, M. (2010). HTTP cache-control extensions for stale content (Internet Engineering Task Force) Available at: https://datatracker.ietf.org/doc/rfc5861 [Accessed July 13, 2022].

9. University, P. DPWG: Dutch pharmacogenetics working group. PharmGKB. Available at: https://www.pharmgkb.org/page/dpwg [Accessed July 16, 2022].

10. Legrand, J., Gogdemir, R., Bousquet, C., Dalleau, K., Devignes, M.-D., Digan, W., Lee, C.-J., Ndiaye, N.-C., Petitpain, N., Ringot, P., *et al.* (2020). PGxCorpus, a manually annotated corpus for pharmacogenomics. Sci Data *7*, 3. Available at: http://www.nature.com/articles/s41597-019-0342-9 [Accessed July 16, 2022].

List of Figures

Glossary

annotation A piece of medical information curated specifically for PharMe and directed at patients, i.e. people without prior professional medical education. An annotation can be a drug class or indication for a drug or an implication, recommendation or warning level for a guideline. v, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 25

API Application Program Interface. 1, 3, 4, 9, 10, 12, 14, 15

CPIC Clinical Pharmacogenetics Implementation Consortium, PharMe's main resource for pharmacogenomic information. 3, 4, 5, 7, 8, 9, 11, 12, 16, 19, 25

DPWG Dutch Pharmacogenetics Working Group, a resource for pharmacogenomic information. 16, 20

DrugBank PharMe's main resource for drug information. 3, 4, 5, 7, 9

guideline Information on a drug-phenotype pair, consisting of an implication, a recommendation and a warning level. 3, 4, 11, 12, 13, 16, 27

implication The consequence of having a given phenotype with respect to the expected response to a given drug. 4, 12, 27

indication A valid reason to use a drug. 4, 12, 27

pharmacogenomic Relating to pharmacogenomics (PGx), the study of how an individual's genome affects drug response. Used as a hypernym for pharmacogenomic and pharmacogenetic in this thesis since the precise difference between the two is not relevant to its subjects. iii, 1, 2, 3, 4, 5, 7, 15, 16, 19, 20, 21, 27

recommendation A suggestion resulting from an implication. 4, 12, 16, 25, 27

RxCUI A unique drug identifier given by the standardized drug nomenclature RxNorm. 3, 13

Text Brick A predefined textual prototype or template used to create annotations in the Annotation Interface. v, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 19, 25

warning level A three-tier indication for a recommendation's severity, i.e. "green", "yellow", or "red". 4, 12, 13, 15, 27

Eidesstattliche Erklärung

Hiermit versichere ich, dass meine Bachelorarbeit “wip” (“deutscher Titel”) selbständig verfasst wurde und dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt wurden. Diese Aussage trifft auch für alle Implementierungen und Dokumentationen im Rahmen dieses Projektes zu.

Potsdam, den 23.07.2022,

(Jannis Baum)