Bachelor Thesis

Facilitating the Research, Curation, and Administration of Pharmacogenomic Information to Provide Multilingual Access for Non-Professionals

Unterstützung der Recherche, Kuratierung und Verwaltung pharmakogenomischer Informationen, um mehrsprachigen Zugang für Nicht-Fachleute zu ermöglichen

Jannis Baum

Supervisors

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

Hasso Plattner Institute at University of Potsdam

July 23 2022

Abstract

PharMe is a service that aims to bring PGx and its potential advantages to the treatment of patients by providing them with personalized pharmacogenomic insights. To be useful to patients, these insights need to be communicated comprehensively. Researching, curating and administering the necessary information is a time intensive process conducted by PGx experts. This process is facilitated by the Annotation Interface researched and implemented in this thesis. The primary used method modularizes information to be made up of predefined textual components rather than free text. This increases consistency, maintainability, and curation efficiency of information, while also laying the foundation for support of multilingualism: By merely translating its components, all current and future information is available in the respective language. The presented methods attend PharMe's need for administration of information and provide insights towards establishing systems to solve similar problems for other health-related informative resources.

Zusammenfassung

PharMe ist ein Service mit dem Ziel, PGx und seine potenziellen Vorteile in die Behandlung von Patient*innen einzubringen, indem er ihnen personalisierte pharmakogenomische Erkenntnisse liefert. Um für die Patient*innen nützlich zu sein, müssen diese Erkenntnisse verständlich vermittelt werden. Das Recherchieren, Zusammenstellen und Verwalten der erforderlichen Informationen ist ein zeitintensiver Prozess, der von PGx-Expert*innen durchgeführt wird. Dieser Prozess wird durch das Annotation Interface, das mit dieser Arbeit erforscht und implementiert wird, erleichtert. Die primär verwendete Methode modularisiert die Informationen, so dass sie aus vordefinierten Textkomponenten anstatt aus freiem Text bestehen. Dies erhöht die Konsistenz und Effizienz der Informationspflege und schafft gleichzeitig die Grundlage für die Unterstützung mehrerer Sprachen: Bereits die Übersetzung der Komponenten genügt, um alle aktuellen und zukünftigen Informationen in der jeweiligen Sprache zur Verfügung zu stellen. Die vorgestellten Methoden erfüllen PharMe's Erfordernis an die Verwaltung von Informationen und bieten Einblicke in die Entwicklung von Systemen zur Lösung ähnlicher Probleme für andere gesundheitsbezogene Informationsressourcen.

Contents

1 Introduction

Pharmacogenetics and pharmacogenomics (PGx), the studies of how individuals respond to drugs based on their personal genetics, have been an ongoing research topic since the 1950s [1]. 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 treatment of patients, we have worked on the service and research project *PharMe* in a team of eight students from Hasso Plattner Institute at Prof. Böttinger's chair Digital Health - Personalized Medicine. The implementation of PharMe I refer to in this thesis can be found in its GitHub repository.[1]

PharMe is directed specifically at patients, i.e. people without professional medical education, and aims to (1) be an educational resource by introducing its 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, the app needs to retrieve a collection of readily available data given by multiple sources and display it to its users. The cumulation and provision of this data is the main task of our backend system, the Annotation Server.

The Annotation Server fetches data that is directed at field professionals from existing APIs of external medical organizations. Aside from this data, it also needs to provide guidelines that are comprehensible to patients. These guidelines are created by PGx experts at the Icahn School of Medicine at Mount Sinai who manually curate them.

In the current implementation of the Annotation Server as we have built it in shared effort, some problems have remained:

- The data the Annotation Server fetches from external sources has to be matched into a common database. Inconsistencies between sources can lead to mismatches, which creates a need for human supervision.
- The PGx experts need an efficient way to research, curate and administer the guidelines they provide without relying on communication with the Annotation Server's technical maintainers. The guidelines they administer in turn also need to be matched with the external data.
- There is no infrastructure to provide guidelines in multiple languages.

To attend these problems, this thesis researches a new component to the PharMe system: the Annotation Interface. The Annotation Interface is a web application directed at PGx

[1] `https://github.com/hpi-dhc/PharMe/tree/bbb9595`

experts and aims to facilitate the process of researching, curating and administering patient-oriented, multilingual PGx information.

In this thesis, I give an overview of the Annotation Server's implementation and explain the before mentioned problems in more detail. Subsequently, I propose, implement and test the Annotation Interface as a solution to these problems. Finally, I discuss future work on administration of information for PharMe based on the presented findings.

2 Annotation Server

The Annotation Server is PharMe's data providing backend. The data it provides consists of two main groups of information:

1. Drugs – users should be able to search for and find any drug they want to consult PharMe about; regardless 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 a database called DrugBank, which is one of the world's most used resources for drug information [2]. With their academic license, we can download an XML file from DrugBank that contains all the information we use from them. Additionally, PGx experts annotate drugs that have pharmacogenomic relevance with descriptive texts that are 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" [3] 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 guidelines with patient-friendly wording.

All the data the Annotation Server is able to provide in its current implementation is English.

2.1 Technical overview

The Annotation Server is a web application built with NestJS, "a framework for building efficient, scalable [...] server-side applications [with support for TypeScript]" [4], the programming language the majority of PharMe's backend is built with. For storage of data, the Annotation Server relies on PostgreSQL, "the world's most advanced open source relational database" [5] and TypeORM [6] to interact with PostgreSQL from the NestJS application.

The Annotation Server provides POST endpoints that trigger loading the data it uses from CPIC and DrugBank. Before loading is initialized, all previously existing data is deleted from its database.

There are three main modules to the Annotation Server: `medications`, `phenotypes` and `guidelines`. Figure 2.1 shows a simplified ER-Diagram of the database structure corresponding with these modules.

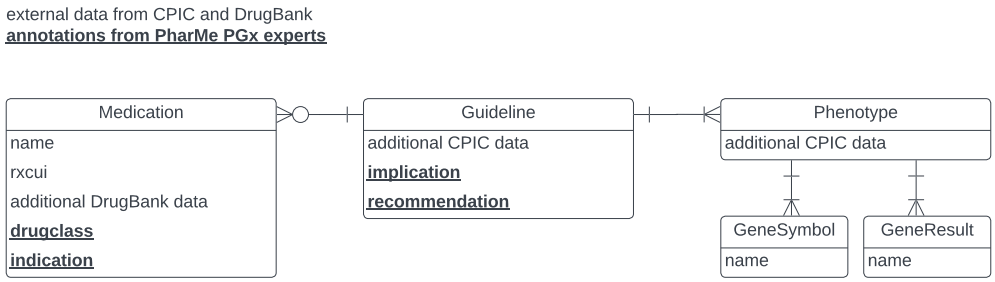

Figure 2.1: Simplified ER-diagram of Annotation Server database

The `medications` module is responsible for all data regarding drugs. It stores drug data by maintaining a table in the Annotation Server's database. This table is initialized by loading and saving all relevant data from the DrugBank XML file. The saved data consists of drug names, descriptions, synonyms and RxCUIs, which are unique identifiers given by the National Library of Medicine's standardized drug nomenclature RxNorm [7]. Once the drug table 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 table. 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 CPIC data about phenotypes is also stored. The `phenotypes` module exposes no dedicated endpoints as it is only used in relation to the `guidelines` module.

The `guidelines` module keeps guidelines in relation to phenotypes and drugs in its guidelines table. This data is initialized by loading all of CPIC's guidelines from its API. The relations to corresponding drugs are created based on the RxCUI CPIC provides with their guidelines. Since inconsistencies in assigned RxCUIs between CPIC and DrugBank may occur, or drugs referenced by CPIC may be missing from DrugBank, these errors are tracked in a separate `GuidelineError` table 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.

Both the `medications` and `guidelines` modules expose PATCH endpoints to annotate the additional patient-oriented data provided by the PGx experts. These annotations consist of

- a drug class and an indication for drugs,
- 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.

In the Annotation Server's current implementation, 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. Here, a data match is given when the drug names, gene symbols and gene results the PGx experts manually write into the Google Sheet are equal to the analogous data from CPIC and DrugBank stored on the Annotation Server. Errors resulting from mismatches are, again, stored as `GuidelineErrors`.

2.2 Data administration process and shortcomings

The 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 research from sources such as CPIC. This party manually writes their annotations into the Google Sheet, initially without any feedback regarding if and how well they match the Annotation Server's existing data from external sources.
- **A maintaining party** with sufficient technical knowledge to invoke the requests to fetch 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 operating the Annotation Server and creates an overhead of communication between the two parties.

Users experience a downtime of the service during the process of updating data. Therefore, updating of data should occur only when new CPIC or DrugBank data is available or when the curating party has modified the Google Sheet. 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 data from CPIC – the same data the Annotation Server had already stored and that the manually curated data later had to match with again.

PharMe is desired to support additional languages besides English, e.g. German for testing conducted by HPI in Germany and Spanish for testing by the Icahn School of Medicine

at Mount Sinai in New York, where it is also necessary to target the Hispanic population. To extend PharMe to support multiple languages in the future, the discussed process would most likely need to be repeated with multiple curating parties and multiple Google Sheets, which would further increase complexity of the process of data administration.

3 Annotation Interface

This chapter describes the conceptualization, implementation, and expert testing of a new component of the PharMe system: the Annotation Interface. The Annotation Interface aims to solve the issues discussed in Chapter 2.2 by improving the overall experience and efficiency of the process of researching and 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 data maintenance. Moreover, the Annotation Interface conceptualizes and partly implements automation into the curating party's research and curation process to further increase efficiency. Finally, it implements an approach towards operating PharMe with support for multiple languages by modularizing annotations.

3.1 Conceptualization

The devised concept for the Annotation Interface closes the gap between the curating party and the Annotation Server by providing the missing infrastructure to allow curators to administer external data, as well as to create, view, update and delete their annotations live on the Annotation Server through a web interface. Additionally, external data such as CPIC guidelines that curators use in their research are made accessible right in the same interfaces that are used to create and maintain annotations.

The second major feature of the concept for the Annotation Interface revolves around modularizing annotations. This modularization is achieved by strictly limiting the creation of annotations to combinations of Text Bricks: predefined textual components or templates that adjust to the annotation they are used in. One such Text Brick might be

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

where `#drug.name` is a placeholder that will be resolved to the name of the drug the Brick is annotating. Aside from placeholders, Bricks offer an interface to define their texts in multiple languages, allowing the same Bricks to be used to create annotations for all supported languages.

Finally, the concept proposes employing techniques of natural language processing for picking suitable Text Bricks based on the corresponding CPIC guidelines. A model trained to infer Text Bricks accordingly presents curators with suggestions for which Bricks to use in creating an annotation. 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 [8]. In this example, the user clicks on the first Text Brick to get a visual explanation of why it was suggested.

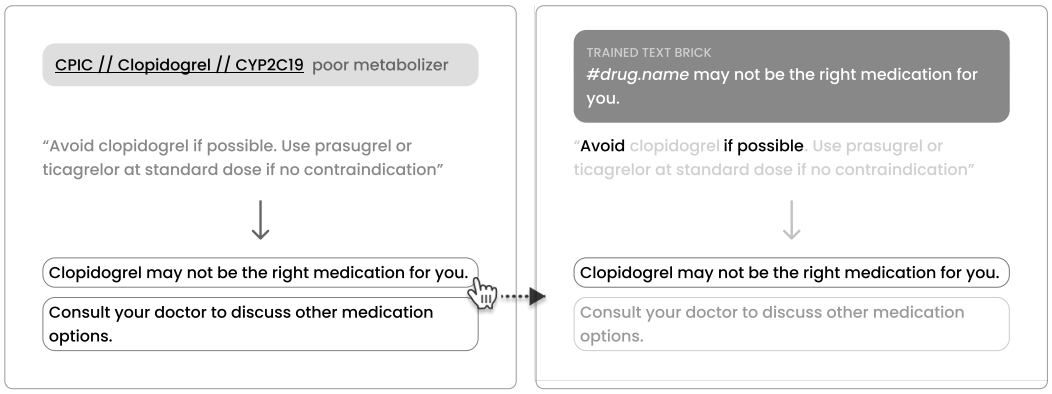

Figure 3.1: Conceptualized suggestion of Text Bricks based on CPIC guideline (left) and visual explanation of why one of the Text Bricks was suggested (right)

3.2 Implementation

In the scope of this thesis I implemented a minimal viable version of the conceptualized Annotation Interface. The implemented features were selected in consultation with PGx expert Dr. Aniwaa Owusu Obeng and include

- infrastructure that eliminates the need for the Google Sheet and gives control over data updates to the curating party,
- presenting CPIC findings on the Annotation Interface during the curation process,
- modularizing annotations by constraining them to be made up of combinations of defined Text Bricks.

The Annotation Interface is implemented as its own web application that is distinct from the Annotation Server. This separation is kept to allow for exploration and research of *an* Annotation Interface as a more general concept beyond PharMe, and to avoid reimplementing large parts of the existing Annotation Server.

Like the Annotation Server, the Annotation Interface uses TypeScript as its main programming language. Contrary to the Annotation Server's framework NestJS, the Annotation Interface is built using Next.js, a framework for creating web applications based on React [9], which is a "JavaScript library for building user interfaces" [10]. To store Text Bricks and the annotations they make up, the Annotation Interface has its own MongoDB document database [11] and uses the library Mongoose to interact with the database from TypeScript [12].

The Annotation Interface has primarily been tested using Apple's web browser Safari on macOS, and, in its minimal implementation, does not strive to provide responsiveness for screens smaller than those of 13-inch laptops.

3.2.1 Usage flow and technical overview

The implementation of the Annotation Interface can be found in PharMe's GitHub repository 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 sections, *Home*, *Annotations* and *Bricks*, which are accessible through a navigation bar on the left side of the interface. Navigation between these sections 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 section and manages persistent state within and between sections with the help of React Contexts defined in the `contexts` directory. This automation is based on regular expressions that are matched against the page routes and is defined by `tabDefinitions` in the same file making it easily adjustable and extensible in case more pages or sections are added.

All main pages of the Annotation Interface include a short description at 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 of getting familiar with the interface and with PharMe's terminology.

3.2.1.1 *Home* section

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 can be seen in the upper part of Figure 3.2.

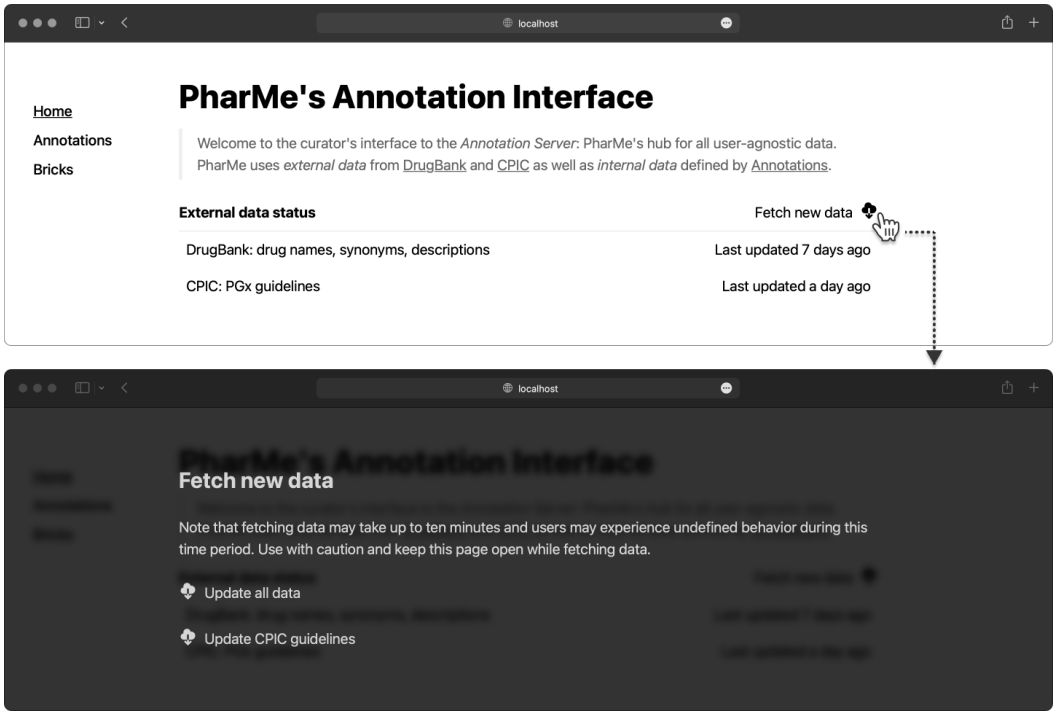

Figure 3.2: The *Home* page (top) and its interface to update external data (bottom)

The overview includes information on when the data has most recently been updated. The dates for these updates are retrieved from the Annotation Server with the cache control

technique stale-while-revalidate [13]. Using this technique, the dates are revalidated to be kept up-to-date with the Annotation Server in case changes occur. Stale-while-revalidate makes the revalidation invisible to the user by displaying the previously cached (stale) date while fetching the update [13].

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 resets all data on the Annotation Server, the Annotation Interface afterwards uses the Annotation Server's PATCH API to 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 as visible in the lower part of Figure 3.2. This change in design 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 PharMe users. The dark themed overlay is used in all parts of the Annotation Interface that have the ability to make such modifications.

3.2.1.2 *Bricks* section

By navigating to the *Bricks* section, the user can view, create, modify and delete Text Bricks in multiple languages. The initially visible Brick overview is shown in Figure 3.3.

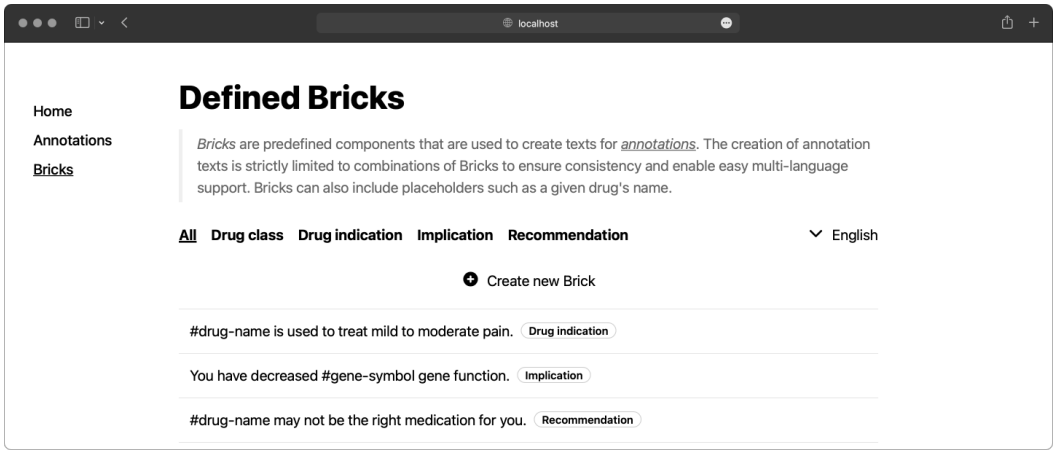

Figure 3.3: Overview of defined Text Bricks

This 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* section.

When clicking on an existing Brick or when adding a new one, the user is presented with a page that is dedicated to editing Text Bricks and their translations. Figure 3.4 shows a Brick's English translation being edited.

On this page 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 name. To facilitate

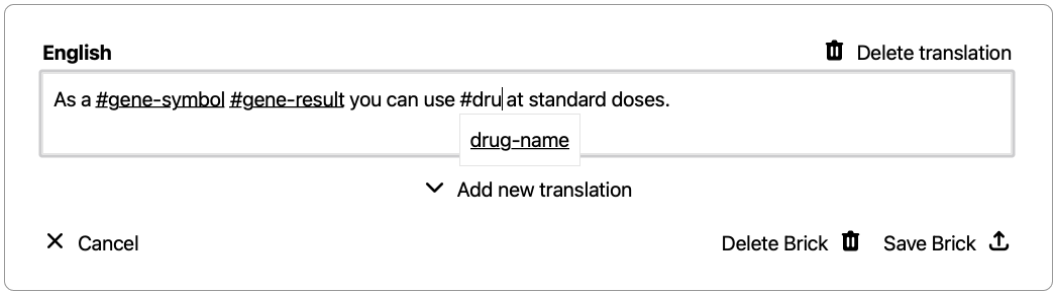

Figure 3.4: Editing a Text Brick with placeholders

working with placeholders when editing or creating Bricks, the text fields used here feature automatic completion, suggestion, and highlighting of placeholders; similar to that of modern code editors.

As visible in Figure 3.4, valid placeholders within the Brick are highlighted with an underline, and a suggestion menu is displayed right below 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.

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*.

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 using an HTML `<div>` element along with the `<textarea>` element that enables the text editing. The `<div>` is aligned to have the same bounding box as the `<textarea>` and placed behind it, while the `<textarea>` itself is given a transparent background to make the `<div>` visible. With this layout, the same text that the user is writing into the `<textarea>` is simultaneously added to the `<div>`, which then provides the highlighting and completion menu as children. Hence, the text visible in Figure 3.4 stems from the `<textarea>`, while the underlines and completion menu are children of the `<div>` behind it.

3.2.1.3 *Annotations* section

The Annotation Interface's final section, *Annotations*, provides the ability to manage PharMe's internal patient-oriented annotations made out of Text Bricks. Through this section, the user is given access to all the data present on the Annotation Server along with filtering options. The initial overview in this section is depicted in Figure 3.5.

This overview is split up into entries for drugs and entries for guidelines. All entries presented here are associated with a guideline CPIC offers, or with a drug corresponding with one of these guidelines. Additionally, each entry that is missing one or more types of annotations has a label indicating this. The first listed entry in Figure 3.5 for example is missing its warning level annotation.

Supplementary to selecting the category *drugs* or *guidelines*, the user can choose to filter for entries with missing annotations or entries that already have all patient-friendly annotations defined, i.e. are fully curated. In addition to this, curators can filter entries

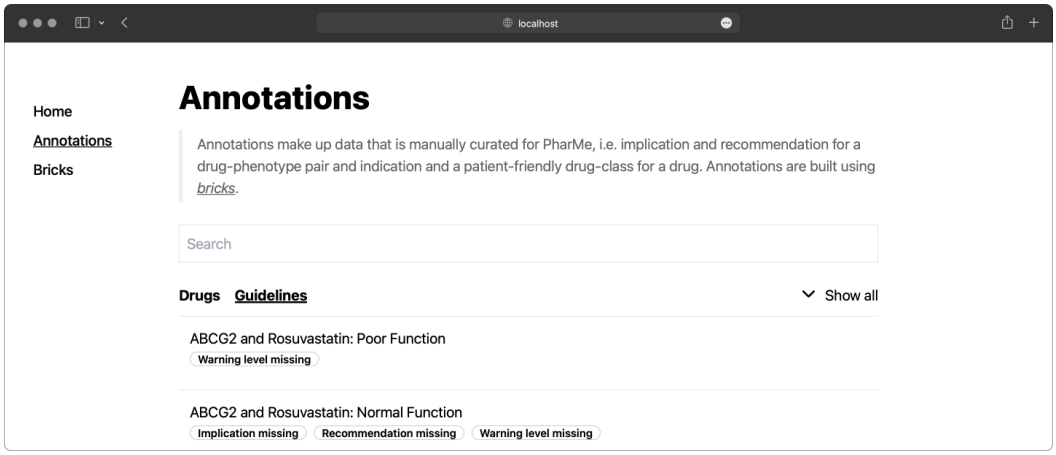

Figure 3.5: Overview of guideline annotations

more specifically by using the search bar. To preserve the filtering states and selected entry category across navigation within the *Annotations* section, these are stored using the React Contexts provided through the `Layout` component. This is done analogously to the filtering states in the *Bricks* section.

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 3.6 shows a recommendation annotation being edited in the overlay by dragging a second Brick onto the annotation canvas (box).

Saving an annotation from this overlay calls the Annotation Interface's API which (1) saves the annotation in the Annotation Interface's database as a combination of references to saved Text Bricks and (2) calls the Annotation Server's API to save the text the Bricks resolve to, visible above the annotation canvas in Figure 3.6. 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

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

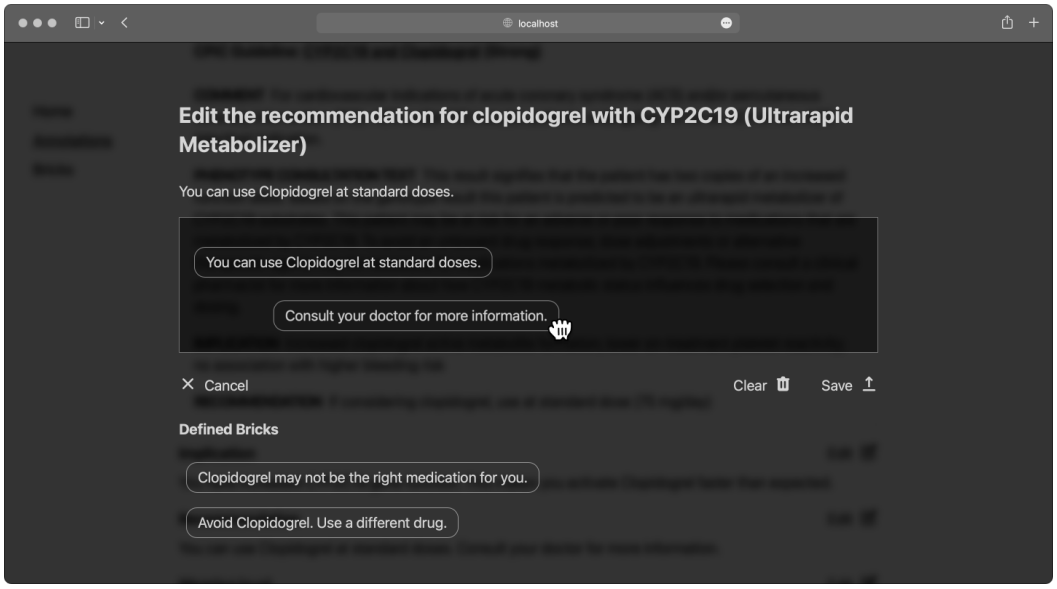

Figure 3.6: Editing a recommendation annotation

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`,
- 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 RxCUI in the documents for all annotations, and 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.

Both `MedAnnotation` and `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 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.

The `TextBrick` model implements finding Brick documents and then returning their resolved texts 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, I will use this section to walk through how to implement the following additions to the Annotation Interface:

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

These additions, 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 a new category of Text Bricks is handled by adding the new category's display 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* section 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

To implement a React component that creates the visual interface for a new type of annotation, a decision needs to be made on which of the existing components is most suitable as a starting point. All components referred to in this section are implemented in the `components/annotations` directory.

The existing interfaces that display an annotation's current value and provide the editing overlay shown in Figure 3.6 are built with multiple levels of abstraction. This abstraction makes the underlying React components reusable for flexible applications. Figure 3.7 gives an overview of all existing components used for displaying and editing annotations and how they are used in composition.

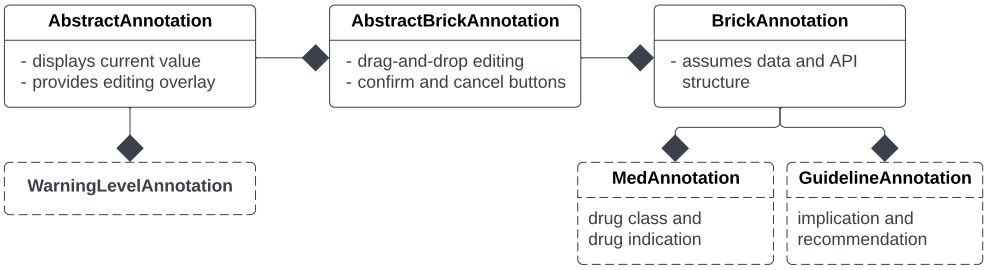

Figure 3.7: Components for existing types of annotations (dotted borders) and their compositional makeup of abstract components (solid borders)

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. Additionally, it implements showing and hiding the editing overlay visible in Figure 3.6, and creating the overlay's basic structure. `AbstractAnnotation` provides a React reference that enables its parent component to hide the overlay and places its own children on the overlay. The components for all existing types of annotations use `AbstractAnnotation` in composition.

The components for Text Brick-based annotations feature additional levels of abstraction. The `AbstractBrickAnnotation` component uses `AbstractAnnotation` to implement the interface seen in Figure 3.6 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.

Finally, the visual interface for the new type of annotation to implement is created as a composition with one of these three abstract components. If the new type of annotation, like the existing warning level annotation, is not based on Bricks, it is directly implemented as a composition with `AbstractAnnotation`. If it is Brick-based, it is implemented with `AbstractBrickAnnotation` or `BrickAnnotation`. This is dependent on whether its data

and API structure follow the patterns of the other existing types of annotations as assumed by `BrickAnnotation`.

3.3 Expert testing

To test the Annotation Interface's usability and how it meets its goals of improving the process of curating, researching and administering annotations for PharMe, I set up a consultation with Dr. Aniwaa Owusu Obeng, an expert of pharmacogenomics.

3.3.1 Setup

The consultation was set up as a roughly hour-long video call and was initially accompanied by presentation slides supporting the introduction and explanation of the testing process. To start off, I briefly reintroduced what the Annotation Interface is and aims to accomplish, and explained how it fits into the PharMe system. The testing was conducted on July 12, 2022.

Before starting, I explained the aims, focus and process of the testing and asked the expert to (1) think out loud and mention anything that is unclear, and (2) 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 main part of the consultation revolved around use case testing. Instead of merely presenting the application, 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.

I chose the following scenarios for increasing complexity and to progressively explore 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 to use the interface quickly and without apparent confusion. They mentioned liking the display of CPIC 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 [14].

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* [15] 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 including 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 3.6), the expert remarked that a search bar should be incorporated where the draggable Text Bricks are listed. They explained this would 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 clinical 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 users will have access to.

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

The implemented Annotation Interface provides the missing infrastructure to allow curators to administer the Annotation Server's external data as well as to create, view, update and delete their annotations through a web interface. This eliminates the need for both the Google Sheet and for a second party involved in data administration, and thereby solves the Annotation Server's problem of high communication complexity.

With annotations inherently created on top of CPIC guidelines, the problem of mismatches between them disappears. Additionally, since CPIC guidelines are accessible to curators right in the Annotation Interface, the time they have to spend on research is reduced, making the overall process more efficient.

The modularity of annotations introduced, by limiting them to be combinations of Text Bricks, brings multiple advantages.

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 further reduces time requirements.

All annotation texts being made up of a finite number of Text Bricks also simplifies future 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, all annotations are subsequently 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.

Finally, the Annotation Interface has been implemented for modularity and flexibility, making it provide relevant requirements towards maintainability and extensibility for future use and development.

Given the above points, the Annotation Interface meets its aim of improving the process of researching, curating and administering patient-oriented, multilingual PGx information for experts, while solving the discussed problems of the Annotation Server. These statements are supported by PGx expert Dr. Aniwaa Owusu Obeng, who emphasized them in the conducted testing.

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. The following sections will discuss topics for future work and their priorities in different scenarios.

4.1 Future work

A list of minor features and changes to implement next into the Annotation Interface can be found within PharMe's GitHub issues labeled `P: Annotation Interface`. This list includes, for example, the naming of and searching for Text Bricks that was suggested by the expert while testing the application in Chapter 3.3, as well as some items regarding configuration and general robustness of the Annotation Interface. In the following, I will go into detail about future work beyond these minor features and changes, i.e. features to consider in implementing PharMe for more extensive testing or clinical environments.

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 was outside of the scope of this thesis, since creating the now existing infrastructure was necessary first to make the automated suggestions viable and useful. Should automated suggestions be implemented in the future, related work such as PGxCorpus [16] 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. Since *SmartPhrases* are a tested concept that has already been implemented and is being used in clinical settings, they may offer improved or additional solutions to the similar problems the Annotation Interface aims to solve. Additionally, a user who is already familiar with *SmartPhrases* may be able to more easily learn the concept of Bricks by being made aware of the similarity.

In PharMe's current implementation, the Annotation Server's data structure only supports one language and the smartphone application's user interface is only available in English. To have the infrastructure the Annotation Interface provides towards supporting multiple languages contribute value to PharMe, these other parts of the system need to be adjusted accordingly.

Moreover, PharMe currently leverages only one external pharmacogenomic resource. To further assist the research process of curating annotations, implementing additional sources such as DPWG is another important 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, there is a gap in PharMe's functionality that needs to be bridged for it to be usable in a clinical setting. 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 clinical 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 features of staging and reviewing changes, keeping a revision history of presented data, and providing change logs for external data updates are essential.

4.2 Prioritization

The PGx expert I consulted to conceptualize, prioritize, and test features of the Annotation Interface has expressed high interest in the feature of employing natural language processing for further automation, yet prioritized it lower than the other features I explained. These other features are prioritized according to the setting PharMe is to be used in.

Should PharMe be used in more extensive user testing and studies, the support for multiple languages is of highest priority. The most important additional languages to support would be German for testing at HPI in Germany and Spanish for testing at the Icahn School of Medicine at Mount Sinai in New York where tests would also be conducted with Hispanic population.

To use PharMe in a clinical setting with users having independent and unsupervised access to the data presented through the smartphone application, the presented clinical use-related features are of highest priority.

Implementing DPWG or other pharmacogenomic resources as additional external data has also been highly prioritized by the consulted PGx expert for any usage setting, though this was regarded as less important than the discussed setting-specific features.

5 Conclusion

In its presented state, the Annotation Interface fulfills its aim of facilitating the process of research, curation and administration of patient-oriented, multilingual information for PharMe. This is primarily achieved by making data administration interactive and modularizing the information to be made up of predefined textual components, Text Bricks, rather than free text.

The modularization has been found to increase the consistency, maintainability and curation efficiency of information, while also laying the foundation for the desired support of multiple languages: With information made up of a finite number of Text Bricks, the task of providing current and future information in an additional language is reduced to translating all Text Bricks once.

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 to be implemented in a clinical setting.

To be used in a clinical setting with users having independent and unsupervised access to the data PharMe provides, higher certainty in correctness needs to be ensured. The most crucial features to achieve this are (1) providing curators with functionality to stage and review data before making it accessible to users, and (2) keeping a revision history of all data presented to users.

The Annotation Interface as a system provides an approach to attend PharMe's need for data administration by pharmacogenomic experts. The explored concepts of modularized information as well as needs for clinical settings 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. 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].
4. Kamil Mysliwiec NestJS documentation. Available at: https://docs.nestjs.com [Accessed July 20, 2022].
5. The PostgreSQL Global Development Group (2022). PostgreSQL homepage. Available at: https://www.postgresql.org/ [Accessed July 11, 2022].
6. TypeORM TypeORM documentation. Available at: https://typeorm.io/ [Accessed July 20, 2022].
7. 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].
8. 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].
9. Vercel, Inc Next.js homepage. Available at: https://nextjs.org [Accessed July 21, 2022].
10. Meta Platforms, Inc React homepage. Available at: https://reactjs.org/ [Accessed July 21, 2022].
11. MongoDB, Inc MongoDB homepage. Available at: https://www.mongodb.com [Accessed July 21, 2022].
12. Automattic, I. Mongoose homepage. Available at: https://mongoosejs.com/ [Accessed July 21, 2022].

13. 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].

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

15. Epic Systems Corporation Epic homepage. Available at: https://www.epic.com/about [Accessed July 21, 2022].

16. 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, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 27

API Application Program Interface. 1, 3, 4, 10, 12, 14, 15, 16

CPIC Clinical Pharmacogenetics Implementation Consortium, PharMe's main resource for pharmacogenomic information. 3, 4, 5, 7, 8, 9, 11, 12, 16, 17, 19, 20, 27

DPWG Dutch Pharmacogenetics Working Group, a resource for pharmacogenomic information. 17, 20, 21

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

guideline Information on a drug-phenotype pair, consisting of an implication, a recommendation and a warning level. 3, 4, 11, 12, 13, 14, 16, 17, 27, 29

implication The consequence of having a given phenotype with respect to the expected response to a given drug. 4, 12, 29

indication A valid reason to use a drug. 4, 12, 29

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, 8, 16, 17, 19, 20, 21, 23, 29

recommendation A suggestion resulting from an implication. 4, 12, 13, 16, 27, 29

RxCUI A unique drug identifier given by the standardized drug nomenclature RxNorm. 4, 13

Text Brick A predefined textual component or template used to create annotations in the Annotation Interface. v, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 23, 27

warning level A three-tier indication for a recommendation's severity, i.e. "green", "yellow", or "red". 4, 11, 12, 14, 15, 29

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich meine Bachelorarbeit “Facilitating the Research, Curation, and Administration of Pharmacogenomic Information to Provide Multilingual Access for Non-Professionals” (“Unterstützung der Recherche, Kuratierung und Verwaltung pharmakogenomischer Informationen, um mehrsprachigen Zugang für Nicht-Fachleute zu ermöglichen”) selbständig verfasst habe 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 24.07.2022,

_____

(Jannis Baum)