A Medical Information System for Personalized Rehabilitation after Ankle Inversion Trauma

Jonathan Neugebauer[1][a], Rosemary Dubbeldam[2][b], My Linh Pham[1], Lokman Beser[1], Luka Gerlach[1], Yu Yuan Lee[2] and Herbert Kuchen[1][c]

[1]*Department for Information Systems, University of Münster, Münster, Germany*

[2]*Institute of Sport and Exercise Sciences, University of Münster, Münster, Germany*

Keywords: Medical Informatics, Clinical Decision Support System, Mobile App, Domain Specific Language, Model-driven Software Development.

Abstract: We have developed FEAL, a mobile app and a corresponding server component supporting personalized rehabilitation after an ankle inversion trauma. In order to enable the maintenance of the essential parts of the overall system by health professionals, an easy to understand domain specific language (DSL) has been designed enabling them to adapt the questionnaires which are essential parts of the app. For the same reason, the included medical knowledge is not hard coded in a programming language but provided by rules of a business rules management system. A DSL specification is automatically transformed by a correspondingly developed generator to platform-independent React Native code such that the resulting app can be used on the relevant platforms iOS and Android.

1 INTRODUCTION

Particularly physically active people frequently suffer from ankle sprains. Up to 40 % of these cases can eventually escalate to chronic ankle instability (Hertel and Corbett, 2019). In order to prevent this, patients need to execute appropriate exercises which should be adapted to their current state and patients need to be informed about their injury and the expected rehabilitation process. Thus, we have developed FEAL, a mobile app suggesting and explaining personalized exercises to the patient and providing injury and rehabilitation related information. The required medical knowledge cannot be stored on the mobile device running the app. It rather has to be provided by a server to which the app connects if needed. The medical knowledge is on the state of the art. Nevertheless, it has to be updated, if new insights have been found. This maintenance should ideally be performed by a health professional rather than by a software developer. Thus, the medical knowledge has to be provided in a form which can be maintained by health professionals. Hence, it cannot be expressed in a classic programming language. Instead, we use when-then rules of a business rules management system (BRMS) (Boyer and Mili, 2011). In our case, we use the BRMS Drools[1]. In addition to the rules, the server also stores data about the rehabilitation of the patients, which can later on be used for scientific studies.

The mobile app needs to ask patients about the current state of their ankle impairments and their experiences when performing the suggested exercises. Based on the answers, new exercises are suggested after involving the rules engine on the server side. A large part of the app is a *questionnaire*. Again, this questionnaire is also subject to changes which should ideally be performed by a health professional rather than by a software developer. Thus, a questionnaire should be expressed on a high level of abstraction and in a way health professionals can understand. Therefore, we have developed a *domain specific language* (DSL) (Voelter et al., 2013) which allows to design such a questionnaire. Since both, the medical knowledge and the questionnaire, are not hard coded in a programming language but provided in a form where they can easily be adapted, our approach is not limited to the rehabilitation of ankle inversion trauma but

[a] https://orcid.org/0000-0001-5865-7118
[b] https://orcid.org/0000-0001-7471-9737
[c] https://orcid.org/0000-0002-6057-3551

[1] https://www.drools.org

it can also be used in other medical areas with a similar combination of app, questionnaires, and medical knowledge expressed by rules.

Finally, the app needs to be platform-independent, since patients are using different mobile devices based on the operating systems iOS and Android. Common techniques for reaching such independence include the hybrid, interpreted, and cross-compiled approach (Biørn-Hansen et al., 2018). In the *hybrid approach*, apps are implemented using web technologies and displayed within a native web component. By relying on web technologies, the interface can only imitate the look of native interface elements. The *interpreted approach*, on the other hand, allows for using native interface components which are rendered based on, e.g., JavaScript code interpreted during run-time. In contrast, the *cross-compiled approach* does not perform such transformations to native code during run-time but during compile-time.

We have deliberately chosen React Native[2] which follows the interpreted approach. Thus, with the same code base, apps for iOS and Android can be developed. Following the model-driven software development approach (Stahl and Völter, 2006), DSL specifications are automatically transformed into React Native code. This is done by a corresponding *generator* which we have developed, too.

In our work, we followed the design science research methodology (Peffers et al., 2007). From a software engineering point of view, the main contributions of this application paper are:

- development of an *architecture* for a *mobile app* and a corresponding *server* supporting the rehabilitation after an ankle inversion trauma,
- design of a *DSL* for expressing corresponding *questionnaires* on a high level of abstraction,
- development of a *generator* transforming DSL specifications into React Native code,
- development of a set of *rules* expressing the corresponding *medical knowledge*.

The rest of this paper is organized as follows. In the next section, we will provide medical background for our app FEAL. Section 3 presents the overall architecture of our approach. More details about the development will be given in Section 4. In Section 5, we will discuss our approach. Related work is mentioned in Section 6. Finally in Section 7, we conclude and point out future work.

[2]https://reactnative.dev

2 MEDICAL BACKGROUND

An ankle inversion trauma occurs when the ankle joints are twisted into a too extreme inverted position, frequently combined with extension of the ankle joints. As a result, the ankle ligaments can be partially damaged or completely torn, depending on the severity of the trauma. Below, first the injury prevalence and participation consequences are presented. Then, measures which influence injury rehabilitation are briefly discussed.

2.1 Prevalence and Consequences

Ankle sprains are common, and the potential consequences are unfortunately often downplayed. Most patients and even health professionals consider an ankle sprain as a minor injury, much less severe than for example a strained or torn cruciate ligament of the knee. Such opinions are not justified if one considers the 35% to 70% recurrence rate. Even 6-12 months after the injury, 50% of the patients are still suffering from minor to major chronic impairments, and about 10% of them cannot return to sports or work (Hertel and Corbett, 2019; van Putte-Katier et al., 2015). Hence, the consequences of this underrated ankle injury are severe. Medical practitioners have very limited time to provide information about the injury and rehabilitation process and, in general, only prescribe an ankle brace for the more severe cases and advise rest for the less severe cases. Physical therapy is rarely prescribed even though studies report a 60% reduced risk of re-injury after balance training (Eils and Rosenbaum, 2001). About half of the patients may seek help in the form of information in the available media or online. However, such information is usually quite general, mostly not scientifically based, and certainly not adapted to the individual suffering from the ankle injury. Bystanders, such as trainers or coaches, are generally not educated to support injury rehabilitation.

2.2 Rehabilitation Process

Deviations from the normal regeneration of balance, gait and jumping or landing abilities predict chronification of the ankle injury (Doherty et al., 2016). Hence, it is important to restore impaired functions. Several typical impairments can be addressed by the patient. For example, static stretching has a positive effect on the ankle dorsiflexion range of motion (Terada et al., 2013); Balance training and muscle strengthening yield improves balance and reduces recurrent injury risk (Eils and Rosenbaum, 2001). Val-

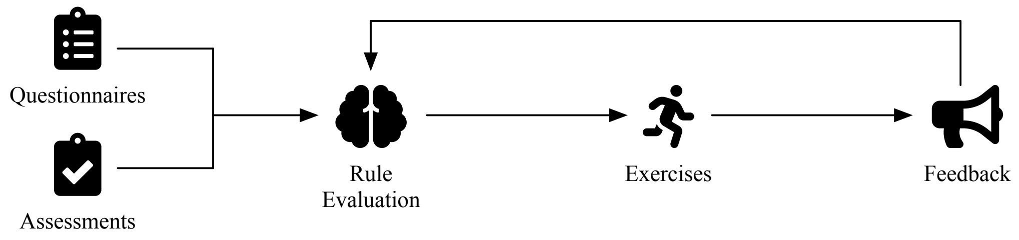


Figure 1: Main phases of the rehabilitation process supported by the medical information system.

idated tools which have been related to the rehabilitation process include the Foot and Ankle Ability Measure (FAAM), the Cumberland Ankle Instability Tool (CAIT) and Fear-Avoidance-Beliefs Questionnaire (FABQ) (Hertel and Corbett, 2019; Houston et al., 2015). Such tools and simple assessments of structural function and activities should be used to monitor the patient's progress. Thus, our idea was to develop a mobile app and a corresponding server backend in order to support the rehabilitation of patients by suggesting exercises which are tailored to the current state of the patient.

3 SYSTEM DESIGN

In this section, the medical information system supporting personalized rehabilitation after an ankle inversion trauma is designed. First, the requirements are discussed in Subsection 3.1. Based on this, Subsection 3.2 outlines the software architecture.

3.1 Requirements Analysis

There are two main stakeholders involved in the rehabilitation process: the *patient* and the *health professional*. Throughout the rehabilitation process, the patient continuously provides information concerning his or her health condition through the system. The health professionals, on the other hand, use this information to monitor and evaluate the user's impairments in structural function and activities and adapt the rehabilitation accordingly. Some parts of this adaption can be done based on rules expressing the current medical knowledge. Such rules are intended to be evaluated automatically using a rules engine.

In Figure 1, the main phases of the rehabilitation process, which the system is intended to support, are illustrated. The following paragraphs elaborate on each of these phases.

Questionnaires. At first, the patient answers different questionnaires, e.g., concerning types of physical activity the patient has done lately or the pain sensation in certain situations. Some questionnaires are more general while others specifically target ankle impairments. Examples for such questionnaires are the international physical activity questionnaire (IPAQ) (Craig et al., 2003), the pain self-efficacy questionnaire (PSEQ) (Nicholas, 2007), or the CAIT (Hiller et al., 2006) mentioned previously. Some questionnaires allow calculating scores from the given answers which are used by the health professionals to monitor and adapt the rehabilitation process.

Assessments. In addition to questionnaires, a prior assessment based on exercises or tests performed by the patient is required. One example is the weight-bearing lunge test (WBLT) (Bennell et al., 1998) assessing the mobility of the ankle. After doing the assessment, the patient is asked to answer some questions. For example, a question could ask for sensation of pain during the assessment or whether it was too difficult to perform. Furthermore, the patient could be asked to do a measurement (e.g., distance or time) indicating how well the patient was able to perform an exercise or a test.

Rule Evaluation. Based on information collected through the questionnaires and assessments, rules expressing the current medical knowledge are evaluated to determine appropriate exercises the patient should do as a further training. These exercises are organized in exercise groups. An exercise group is a collection of exercises with different difficulty levels regarding a specific biomechanical aspect (e.g., balance or forefoot stance). The rules are used to determine the starting point within an exercise group, i.e., the exercise with the appropriate difficulty level. Depending on the assessment, certain exercise groups might be irrelevant and, thus, are not recommended for the patient (e.g., because the patient is able to do the most difficulty exercise variant without any problems). On the other hand, an exercise group might not be recommended if a required questionnaire or assessment was not done yet.

After the starting points within the relevant exercise groups have been determined, the exercises are prioritized. This prioritization is done based on a list of exercises which are ordered from most easy and most vital exercise to most difficult exercise or least vital for daily life activities. For instance, the prioritization lists starts with two limb weight-bearing and ankle mobilization exercises and the list ends with single-leg jumps and running exercises. A collection of the three exercises with the highest priorities is then given to the patient as homework. In our use case, the priorities of the exercises are fixed and do not depend on the patient's input. Instead, the priorities are only used to rank the exercises that have been determined by the rules.

Exercises. At this stage, the patient performs the suggested personalized exercises from the homework collection daily.

Feedback. After each exercise, the patient evaluates whether the exercise was "too easy", "too difficult", or "exactly right". This feedback is used as an additional input for the rule evaluation. Depending on the patient's answers the exercises are adapted and a new collection of three exercises is assembled for the next day. If the performed exercise was too easy, a more difficult variant is recommended next (analogously if the exercise was too difficult). If the exercise's difficulty was exactly right, it stays in the homework collection. Depending on the patient's progress, an exercise group may be removed from the homework collection given the order of the exercises on the exercise prioritization list.

In addition to the daily feedback on the exercises, the patient is asked regularly to repeat specific questionnaires and assessments throughout the training phase to allow for a continuous monitoring of the rehabilitation progress.

Based on this description of the rehabilitation process several requirements can be identified. They can be classified according to which user group they concern.

Patients. First, patients should be able to fill out questionnaires and do assessments using a mobile app. Additionally, the mobile app should display the homework collection and provide the means to give feedback after having done a homework exercise. Lastly, the app should present a collection of useful information to the patient (e.g., general information on the ankle inversion trauma, advice, or support contacts).

Health Professionals. First, health professionals require a dashboard to monitor the rehabilitation progress of their patients. For this purpose, plots providing a basic analysis of the scores calculated for the answered questionnaires should be presented. To facilitate a more detailed statistical analysis, the information collected through questionnaires and assessments and with the feedback on the exercises should be saved in a database. Personalized and appropriate exercises should be determined automatically based on user input from the questionnaires and assessment and on the rules specified by the health professionals. Lastly, health professionals should be able to adapt the questionnaires.

There are different question types which the system needs to handle. Firstly, single-choice as well as multiple-choice questions should be supported. Additionally, open-ended questions are required. Lastly, for the purposes of our use case it should be possible that patients can answer certain questions using a visual analogue scale (VAS) (Hayes and Patterson, 1921). Here, the patient can choose an appropriate value for a rating scale between two opposing categories (e.g., "not at all confident" vs. "completely confident"). An exemplary question from the previously mentioned PSEQ involving a VAS is depicted in Figure 3.

3.2 Software Architecture

Based on the requirements analysis conducted in the previous subsection, this subsection introduces a suitable software architecture. In Figure 2, several core components are illustrated. On the one hand, there are parts colored in gray involved in code generation prior to running the app, while the components colored in black are needed at runtime.

There are two components providing a user interface (UI) for the two main stakeholders: the *mobile app* and the *dashboard*. Both receive data from and send data to the *backend* through its application programming interface (API). Furthermore, the backend uses a *database* for persistent storage of information. We chose MongoDB[3] due to the flexibility such a document-oriented database offers for the schema design. The rule evaluation is done by a separate component representing the *rules engine*. Also this component features an API enabling the communication to the backend. The rules are saved in specific *rule* files which are loaded into the rules engine upon application start. This way, the rule logic can be adapted by health professionals without the need of touching the program code of the rules engine. Drools offers a DSL called drools rule language (DRL) for the purpose of specifying rules. In addition to using DRL for defining rules, we implemented a DSL named QuApp allowing to describe questionnaires and mobile apps within *models*. Such models can be fed into a *code generator* to generate artifacts corresponding to the models.

While the app model is transformed to JavaScript code for React Native, a questionnaire model results in a JavaScript Object Notation (JSON) document that

[3]https://www.mongodb.com

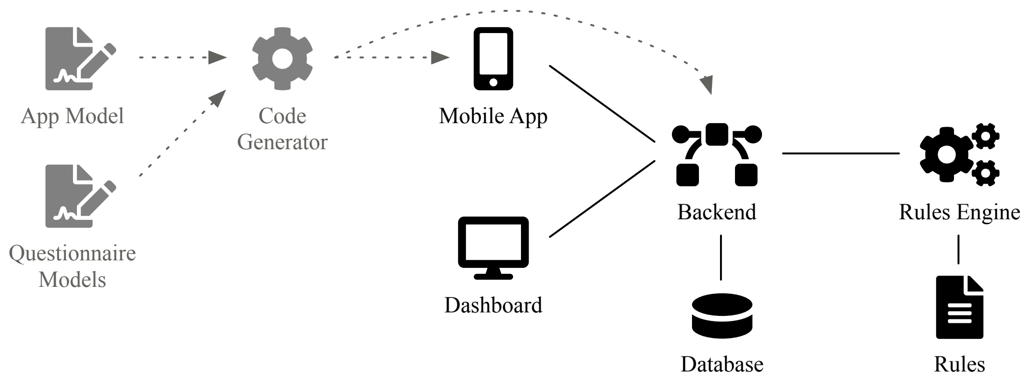


Figure 2: Architecture of the medical information system.

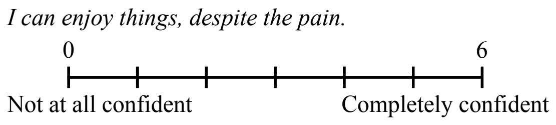


Figure 3: Exemplary question involving a visual analogue scale (cf. PSEQ).

can be loaded into the backend. This document encompasses the meta information about the questionnaire. Such meta information is used in the mobile app and dashboard for displaying purposes and in the rules for the rules engine to express logic. The code generator is able to cover the requirements for the questionnaire part of the app well. However, there are also custom parts which have been realized by custom code next to the generated code. For instance, the implementation for the assessments and exercises were too specific to our use case, thus, preventing a meaningful abstraction into language elements for the QuApp language. Nevertheless, the custom implementations are also based on the mentioned questionnaire metadata and rules logic. Thus, they do not pose a limitation towards our goal of making changes easy for health professionals.

4 PROTOTYPE DEVELOPMENT

Given the requirements and software architecture outlined in the previous section, this section gives more details on the development of the different system components. First, Subsections 4.1 to 4.4 deal with the two UI components (mobile app, dashboard) and the components realizing the API service (backend, rules engine). Based on this understanding of the prototype, in the last Subsection 4.5, we describe how we used our QuApp DSL to generate certain aspects of the system automatically. Exemplary code showcasing our use of the DSL and DRL rules can be found in a dedicated GitHub repository[4].

4.1 Mobile App

By using the mentioned JavaScript framework React Native, the mobile app could be realized as a cross-platform app. Hence, it can be run on both Android and iOS devices. React Native leans on the JavaScript library React[5] and, thus, the UI appearance and behavior of an app is implemented through React components. Here, a component represents a bundle of code that can be reused to compose the overall app (e.g., an input element that is needed on different screens). At runtime, React Native translates the React components to native view elements for the respective mobile platform.

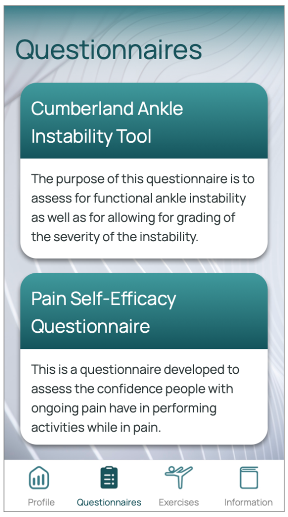
Once logged in, the user is presented the app's main screen. As depicted in Figure 4a, this screen contains a tab bar at the bottom. Here, the user is able to switch between four different sub-screens.

Profile. This screen contains options to view and edit the information of the user profile. Additionally, the user can logout from the app.

Questionnaires. Here, a list of available questionnaires is shown. After choosing a questionnaire, the patient is presented its content step-by-step. A questionnaire is divided into sections where each section contains questions and optionally information pages. An information page can be used to display instructions before questions. For instance, we use information pages to explain when a following block of questions refers to the same overall topic (e.g., about the physical activity done in the last seven days).

All required question types mentioned previously have been implemented. An example can be seen in Figure 4b. Here, the input for a VAS has been realized as a slider. The other question types look sim-

[4]https://github.com/wwu-pi/ankle-rehab-examples
[5]https://reactjs.org


Questionnaires

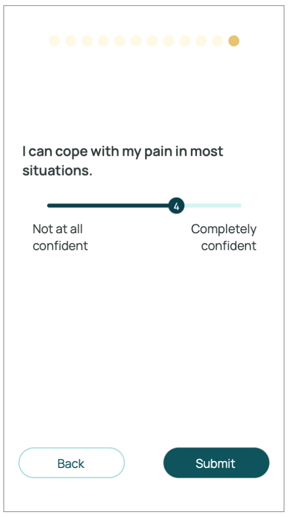
Cumberland Ankle Instability Tool

The purpose of this questionnaire is to assess for functional ankle instability as well as for allowing for grading of the severity of the instability.

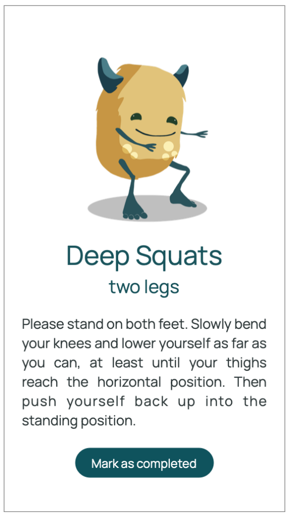
Pain Self-Efficacy Questionnaire

This is a questionnaire developed to assess the confidence people with ongoing pain have in performing activities while in pain.

(a) Overview list with available questionnaires.


I can cope with my pain in most situations.

(b) VAS input within a questionnaire realized as a slider.


Deep Squats

two legs

Please stand on both feet. Slowly bend your knees and lower yourself as far as you can, at least until your thighs reach the horizontal position. Then push yourself back up into the standing position.

(c) Exercise instructions.

Figure 4: Exemplary features of the mobile app.

ilarly. While for single-choice questions radio buttons are displayed, we use check-boxes for multiple-choice questions. Finally, for open-ended questions, a text input field is provided.

Exercises. Analogously to the aforementioned questionnaires list, the patient is presented a list of available exercises from the homework collection. Once an exercise has been selected, a corresponding exercise screen is shown (cf. Figure 4c). Here, instructions on how to do the exercise correctly are given. After doing the exercise, the patient can mark it as “completed” and evaluate whether it was too difficult, too easy, or exactly right.

From the exercise list, the patient also has the option to access the assessments screen (e.g., in case the exercise list is still empty because pending assessments have to be done). The patient can choose from a list of assessments and is given instructions on how to do the exercise or test that is going to be assessed. Furthermore, questions are shown similar to questionnaires (cf. Figure 4b). The question types used for assessments are single- and multiple-choice as well as open-ended.

Information. The final screen provides general information for the user. Again, a list is used to display multiple entries from which the patient can choose. Entries are fetched from the backend as static HTML code and are displayed accordingly when selected.

4.2 Dashboard

The UI for the dashboard is also implemented using React components. After logging in, health profes-

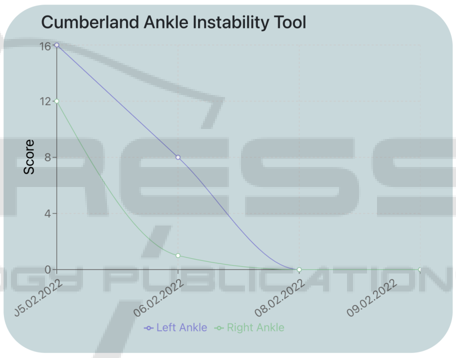


Figure 5: Dashboard plot visualizing the CAIT scores for the left and right ankle over time.

sionals get access to a list of participants. After selecting a participant, plots visualizing the questionnaire scores over time are displayed. A plot with exemplary data for the CAIT score is depicted in Figure 5. In this case, the line plot contains two data series for the left and right ankle. These two series correspond to two sections in the questionnaire allowing the backend to compute the scores individually. For other questionnaires such as the PSEQ, only a single score is computed. The score calculation method is specified in the questionnaire’s metadata.

4.3 Backend

The dashboard is bundled with the backend within a single web application. For the backend implementation, we used the web framework Express[6] which is

[6]https://expressjs.com

based on the JavaScript runtime Node.js[7].

The main responsibility of the backend is to provide an API for the other system components. This includes endpoints for authentication and data management.

Endpoints for data management provide the means to read, insert, update, or delete entities in the database. For instance, questionnaire metadata is retrieved by the mobile app for data collection and by the dashboard for data visualization. Similarly, the mobile app sends answers to questionnaires to the backend via a corresponding endpoint.

In addition to obtaining data from the database, certain endpoints require evaluating rules. For instance, this is the case after a questionnaire or an assessment has been submitted. Additionally, the rules engine is consulted when the user evaluates the difficulty of an exercise. Finally, the rules engine determines which exercises are returned by the backend for the homework collection.

4.4 Rules Engine

As mentioned earlier, the rules engine was implemented based on the BRMS Drools. It offers an API for the backend to trigger a rule evaluation based on the user's inputs. Hereby, the rules engine is completely stateless and only depends on the rules loaded upon start and information provided in a particular request. Since Drools is implemented in Java, we also realized the API service of the rules engine with Java based on the Spring Framework[8].

The rules are organized in several rule files each containing a set of rules. Firstly, the assignment of exercises to groups, the difficulty levels as well as the priorities of exercises are defined using rules. Secondly, there are rule sets for evaluating which exercise to add to the homework collection given a specific questionnaire or assessment input.

In Listing 1, an exemplary rule in DRL syntax is shown. After the definition of the rule's name in line 3, there are two main blocks. While the `when`-block (cf. line 4 ff) specifies the conditions for the rule to fire, the `then`-block (cf. line 10 ff) contains the logic to be executed when the conditions are met. In this example, the rule is only fired when the VAS input for the fifth question of the CAIT questionnaire has a value higher than zero. In this case, an exercise with ID `B2` contained in exercise group `G1` is added to the list of exercises defined in line 1.

[7]https://nodejs.org

[8]https://spring.io

```
1  global List<Exercise> exercises;
2
3  rule "R1"
4  when
5    Questionnaire(
6      questionnaireId.equals("CAIT"),
7      questionnaireAnswers.get("Q5")
8        > 0
9    )
10 then
11   Exercise e = new Exercise();
12   e.setExerciseId("B2");
13   e.setGroupId("G1");
14   exercises.add(e);
15 end
```

Listing 1: Exemplary rule in DRL syntax defining which exercise to add to the homework collection based on questionnaire inputs (simplified).

When the backend makes a request to the rules engine providing the questionnaire input, the API controller feeds the supplied data as facts to the working memory of Drools. Additionally, the global variable `exercise` (cf. line 1) is initialized with an empty list. Then, all rules are fired which causes Drools to evaluate the rules based on the provided facts and execute the rule logic where conditions are met. At this stage, all applicable rules write their corresponding exercises to the global variable. Subsequently, the controller can return this list as a result to the request. The backend, on the other hand, can then use the specified IDs to obtain the exercise entities from the database to provide them to the mobile app.

4.5 Domain-specific Language

In this subsection, the main concepts of the QuApp DSL are presented. We developed the DSL using the Xtext[9] language workbench. After defining the language elements with a grammar, Xtext automatically generates a basic language infrastructure. For instance, this includes a parser, basic validation, and a plug-in integrating the language into Eclipse IDE[10]. Additionally, Xtext provides an infrastructure to implement custom code generators transforming the DSL specifications to general-purpose code (e.g., JavaScript).

There are two concerns the DSL targets: specifying mobile apps and questionnaires. Both aspects will be discussed subsequently.

[9]https://www.eclipse.org/Xtext

[10]https://www.eclipse.org/eclipseide

4.5.1 Mobile App Specification

To specify a mobile app, first, general properties have to be defined. This includes the app's name and version. Furthermore, the URL to a data provider must be set. In our case, this is the API provided by the backend. Every generated mobile app will have a questionnaire screen. Next to this basic feature, custom sub-screens for the tab bar (cf. Figure 4a) can be created using the keyword `CustomTab`.

Listing 2 contains an exemplary specification of a custom tab. Here, the functionalities of the information screen mentioned in Subsection 4.1 are defined. First, a data model for the static information fetched from the specified API endpoint is created. Based on the defined attributes, an index screen for a list of multiple entities and a content screen shown when selecting a single entity is specified. Here, the model's attributes are referenced. The `name` field is used as a title for the list entries and the data contained in the `text` field is handled as HTML code and rendered accordingly. Lastly, using the keyword `InitialScreen` the first screen shown after selecting the corresponding tab bar button is defined. Besides the shown `HTMLBlock`, there are other components that can be used to display the data of an attribute on a content screen (e.g., `Text`, `Image`, or `Video`).

```
CustomTab Information ... {
  Model StaticInfo {
    id
    name
    text
  }
  FetchPath /"staticInfo"
    fetchAsList from
    /"staticInfo/list"
    identifier
    StaticInfo.id
  IndexScreen InfoIndex ... {
    Title StaticInfo.name
    DetailScreen InfoConent
  }
  ContentScreen InfoConent ... {
    HTMLBlock bind StaticInfo.text
  }
  InitialScreen InfoIndex
}
```

Listing 2: Exemplary specification of a custom tab (simplified).

4.5.2 Questionnaire Specification

The implemented system expects the metadata for questionnaires to be defined in a specific JSON schema. While this format is well suited for implementing application logic, it is not easy and concise to write for a user. Consequently, we use the DSL to specify questionnaires and generate the corresponding JSON document automatically.

In Listing 3, the main structure of a questionnaire specification is shown. After defining the questionnaire's ID and name, a rule for the score calculation must be set. With `sumSection`, the score corresponds to the sum of the scores within a section. When multiple sections are defined, multiple scores are calculated. Another calculation method is `sumAll` where only one score for all sections is determined. For each section, an ID and name must be set and within the section, questions and information pages are defined.

```
Questionnaire CAIT "Cumberland Ankle
                    Instability Tool"
CalculationRule sumSection
Description "Dear patient..."

Section left "Left ankle"
Description "The following questions
          concern your LEFT ankle..."
   Question(s) / information page(s)
 Additional section(s)
```

Listing 3: Exemplary questionnaire specification.

To define an information page, only an ID and the text to display need to be specified. Question definitions, on the other hand, entail more properties. For each of the question types implemented in the mobile app (cf. Subsection 4.1), a keyword is defined in the DSL. In Listing 4, a multiple-choice question is specified. Here, each option is assigned an individual score. Single-choice questions can be defined analogously using the `SCQuestion` keyword. However, for single-choice options, the user can define cross-references to a follow-up question. For instance, one could append the expression "`=> S2.Q11`" to line 4 to indicate that when choosing this option the next question will be question 11 from section 2. Without such a cross-reference, questions are shown according to the order in which they are defined within a section.

```
1 MCQuestion Q1 "I have pain in my
2                ankle."
3   Choice "Never"
4     Score 5
5   Choice "During sport"
6     Score 4
7   ...
```

Listing 4: Exemplary multiple-choice question (cf. CAIT).

Listing 5 contains an exemplary slider question, i.e., a question with a VAS input. Besides defining the titles for the two opposing categories, the minimum and maximum values as well as the interval to use for the slider must be set.

```
SliderQuestion Q1 "I can enjoy things
                    despite the pain."
  MinValue 0 : "Not at all confident"
  MaxValue 6 : "Completely confident"
  Interval 1
```

Listing 5: Exemplary slider question (cf. PSEQ).

Finally, an example for an open-ended question is shown in Listing 6. The input provided to this question type is handled as plain text and, thus, is not considered for the score calculation.

```
OpenEndedQuestion Q26 "During the
     last 7 days, how much time did
     you usually spend sitting on a
     weekday?"
  InputText "Hours per day"
```

Listing 6: Exemplary open-ended question (cf. IPAQ).

5 DISCUSSION

In this section, we evaluate and discuss the results of our work. Subsection 5.1 deals with unit and performance tests evaluating the system's conformance to the functional requirements and its responsiveness. Then, we assess the usability of the dashboard and the approach of expressing expert knowledge using DRL rules. In Subsection 5.3, we evaluate the effectiveness of the DSL-based approach by comparing how much lines of code (LOC) could be generated automatically based on manually written DSL code. Lastly, we discuss limitations of the prototype.

5.1 Unit and Performance Tests

During development of the prototype, we did some functional testing via unit tests. On the other hand, we also did non-functional testing evaluating whether the interaction of the different components introduces an unacceptable latency. For instance, we measured the mean response time of the rules engine's API endpoints given exemplary data. Furthermore, we tested whether large amounts of documents (up to 100,000) in the database lead to an unacceptable increase in latency when querying data within the backend's implementation. In all cases, we did not observe unacceptable response times. The API endpoints responded within less than 11 ms, while the queries took less than 200 ms. All measurements were done on consumer-grade hardware which suggests that running the software on a server in production should result in even lower values. Also, we checked how long it takes to display the dashboard plots for large amounts of data points (up to 400) which did involve rending times of less than 34 ms.

Table 1: LOC comparison between manually written and automatically generated code for different types of models.

Model Type	Written	Generated
PSEQ questionnaire	44	136
IPAQ questionnaire	79	328
CAIT questionnaire	191	597
App with two custom tabs	93	1,706
Sum	**407**	**2,767**

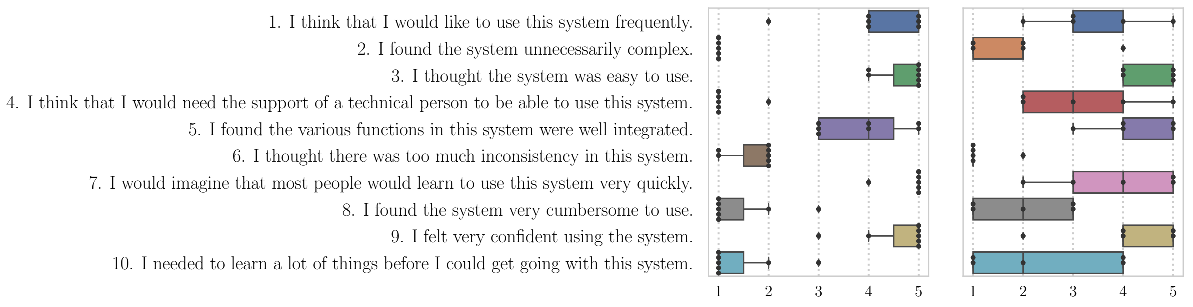
5.2 Usability Tests

For assessing the usability of the system we conducted two usability tests based on the system usability scale (SUS) (Brooke, 1996). While the first test concerned the dashboard's usability (5 participants), the second test assessed the usability of DRL rules format (7 participants). The SUS is a questionnaire comprised of ten statements which the participants rate after having used the software under evaluation. Each statement is rated on a scale between 1 (strong disagreement) and 5 (strong agreement). In the end, a score between 0 and 100 can be calculated after converting and scaling the ratings. Here, higher scores indicate a superior usability.

In Figure 6, the ratings for the ten statements and for both tests can be seen. The final scores for each participant are visualized in Figure 7. All participants had a background in medicine or physical therapy and did not have prior knowledge about programming. On average, the first test concerning the dashboard resulted in a mean score of 88.57 and the second test concerning the DRL format yielded a mean score of 73.00. These results can be interpreted based on the adjective rating scale suggested by Bangor et al. (2009). The dashboard's mean SUS score indicates an "excellent" usability and the DRL format almost reached the rating "good".

5.3 Lines of Code Comparison

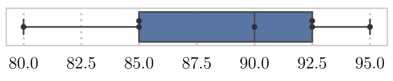
To assess the effectiveness of the approach of defining questionnaires and parts of the mobile app based on the QuApp DSL, we compared the LOC that had to be written in the DSL specifications manually with the LOC that resulted out of the code generation process. In Table 1, the results are summarized. We defined models of different types. Besides different models defining questionnaires, our test also included the specification of a mobile app with two custom tabs. In total, based on 407 manually written LOC 2,767 LOC could be generated.



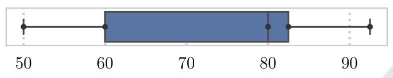
(a) Ratings concerning the dashboard ($n = 7$).

(b) Ratings concerning the DRL format ($n = 5$).

Figure 6: Box plots visualizing the ratings for the SUS statements.



(a) Scores concerning the dashboard ($n = 7$).



(b) Scores concerning the DRL format ($n = 5$).

Figure 7: Box plots visualizing the SUS scores.

5.4 Limitations

There are some limitations to our prototype. First, the SUS score obtained for the DRL format suggests that its usability can still be improved. One reason for this assessment could be that still some technical understanding is needed to implement DRL rules. For instance, in the example in Listing 1, a health professional has to understand that a global variable is to be filled with exercises as part of the rule's logic. One way to overcome this limitation is to abstract away the technical details by defining a simpler language closer to this use case. Drools offers functionalities to define transformations from such custom languages to DRL constructs[11]. Furthermore, only the basic question types required for our use case have been implemented. This could be extended. Also, the generated metadata is specific to our application. One could consider implementing (partial) compatibility with a metadata standard such as ODM-XML[12]. This standard can be used to specify questionnaires and is widely supported in various tools used for clinical research.

[11]https://docs.drools.org/7.67.0.Final/drools-docs/html_single/#_domain_specific_languages

[12]https://www.cdisc.org/standards/data-exchange/odm

6 RELATED WORK

Examples of e-health solutions for musculoskeletal related disorders are the mobile app or web-based systems ViViRa[13], eCovery[14], and Injurymap©[15]. These systems provide a video-based intervention support. Other systems such as MeineReha®[16], SWORD HEALTH[17], Companion® patella[18] and Kaia[19] additionally incorporate technical devices such as the mobile phone's camera and motion sensors, to evaluate and monitor the patient's movements or to provide direct feedback. So far, the entrance level for the exercises and advancement through the exercises is mostly determined by a health professional after personal contact (Companion® patella, MeineReha®) or after a remote consult by means of an interactive platform (SWORD HEALTH). eCovery, Kaia, ViVRa and Injurymap© use a few questions or two-dimensional video analysis with which they are able to perform some form of a remote assessment and monitoring of the patient's health status. In addition, a rules based AI system has been included to advance through the exercises (Injurymap©, ViViRa) or correct exercises (Kaia). Information concerning the disorder such as the rehabilitation process and advice to handle pain, has been provided by a few systems (Injurymap©, eCovery, Kaia, Companion® patella) or can be gained through chatting remotely (SWORD HEALTH). First clinical trials with patients suffering from an acute lateral ankle sprain (with-

[13]https://www.vivira.com/

[14]https://ecovery.de

[15]https://www.injurymap.com

[16]https://www.meinereha.de

[17]https://swordhealth.com

[18]https://www.medi.de/digitale-anwendungen/diga-companion-patella

[19]https://kaiahealth.com

out control groups) and user experience studies indicate that users are satisfied with the app, but exercise adherence may be low if patients need to exercise independently (Injurymap$^{©}$) compared to daily remote biofeedback sessions with a health professional (SWORD HEALTH) (Bak et al., 2022; Correia et al., 2021). Although reminders are offered by most systems, other features such as gamification which enhance exercise motivation have not been included yet (Croon et al., 2021). In summary, many e-health systems still depend on (remote) interaction with a health professional, monitoring and feedback, the system may require costly additional equipment, information concerning the disorder is limited and personalized entrance level and advancement through the exercises is only based on rules engines in a few systems. On the contrary in our system FEAL, the personalized entry level is based on extensive questionnaires as well as an assessment with feedback from the patient. Extensive information concerning the injury is provided. Furthermore, our system does not need additional technology and can be easily adapted by a health professional if needed. Since the App can store health related data such as patient's function and activities as well as answers from questionnaires, it will also be a useful tool for research.

7 CONCLUSION AND OUTLOOK

Considering that ankle sprains are a frequently occurring injury, this paper dealt with developing FEAL, a medical information system for personalized rehabilitation after ankle inversion trauma. After giving some medical background on this use case, we presented requirements and a software architecture for the system. Then, we described how we developed a prototype consisting of several components. A mobile app for the injured person is used to obtain insights on the user's rehabilitation based on questionnaires, exercises, and assessments. For health professionals, on the other hand, we implemented a dashboard helping them to monitor and adapt the rehabilitation progress. As part of the backend components, we developed a rules engine allowing to recommend exercises to the user based on the input to the mobile app. Additionally, we introduced our DSL QuApp allowing to specify questionnaires and mobile apps and generate corresponding code automatically. We evaluated our work regarding different aspects (e.g., functional conformance, performance, usability), discussed limitations, and also pointed out related work.

Future work for FEAL should primarily focus on improving usability and generalizability. We mentioned several means to overcome the limitations of our approach: (1) using a customized language for specifying Drools rules and (2) implementing compatibility with a metadata standard for questionnaires.

REFERENCES

Bak, J., Thorborg, K., Clausen, M. B., Johannsen, F. E., Kirk, J. W., and Bandholm, T. (2022). Using the app “injurymap©” to provide exercise rehabilitation for people with acute lateral ankle sprains seen at the hospital emergency department – a mixed-method pilot study.

Bangor, A., Kortum, P., and Miller, J. (2009). Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. *Journal of Usability Studies*, 4(3):114–123.

Bennell, K., Talbot, R., Wajswelner, H., Techovanich, W., Kelly, D., and Hall, A. (1998). Intra-rater and inter-rater reliability of a weight-bearing lunge measure of ankle dorsiflexion. *Australian Journal of Physiotherapy*, 44(3):175–180.

Biørn-Hansen, A., Grønli, T.-M., and Ghinea, G. (2018). A survey and taxonomy of core concepts and research challenges in cross-platform mobile development. *ACM Comput. Surv.*, 51(5).

Boyer, J. and Mili, H. (2011). *Agile Business Rule Development*. Springer Berlin Heidelberg.

Brooke, J. (1996). SUS: A 'Quick and Dirty' Usability Scale. In Jordan, P., Thomas, B., McClelland, I., and Weerdmeester, B., editors, *Usability Evaluation in Industry*, pages 189–194. Taylor & Francis, London.

Correia, F. D., Molinos, M., Neves, C., Janela, D., Carvalho, D., Luis, S., Francisco, G. E., Lains, J., and Bento, V. (2021). Digital rehabilitation for acute ankle sprains: Prospective longitudinal cohort study. *JMIR Rehabilitation and Assistive Technologies*, 8(3):e31247.

Craig, C. L., Marshall, A. L., Sjöström, M., Bauman, A. E., Booth, M. L., Ainsworth, B. E., Pratt, M., Ekelund, U., Yngve, A., Sallis, J. F., and Oja, P. (2003). International Physical Activity Questionnaire: 12-Country Reliability and Validity. *Medicine & Science in Sports & Exercise*, 35(8):1381–1395.

Croon, R. D., Geuens, J., Verbert, K., and Abeele, V. V. (2021). A systematic review of the effect of gamification on adherence across disciplines. In *Lecture Notes in Computer Science*, pages 168–184. Springer International Publishing.

Doherty, C., Bleakley, C., Hertel, J., Caulfield, B., Ryan, J., and Delahunt, E. (2016). Recovery From a First-Time Lateral Ankle Sprain and the Predictors of Chronic Ankle Instability. *The American Journal of Sports Medicine*, 44(4):995–1003.

Eils, E. and Rosenbaum, D. (2001). A multi-station proprioceptive exercise program in patients with ankle instability. *Medicine & Science in Sports & Exercise*, 33(12):1991–1998.

Hayes, M. H. S. and Patterson, D. G. (1921). Experimental development of the graphic rating method. *Psychological Bulletin*, 18:98–99.

Hertel, J. and Corbett, R. O. (2019). An Updated Model of Chronic Ankle Instability. *Journal of Athletic Training*, 54(6):572–588.

Hiller, C. E., Refshauge, K. M., Bundy, A. C., Herbert, R. D., and Kilbreath, S. L. (2006). The Cumberland Ankle Instability Tool: A Report of Validity and Reliability Testing. *Archives of Physical Medicine and Rehabilitation*, 87(9):1235–1241.

Houston, M. N., Hoch, J. M., Gabriner, M. L., Kirby, J. L., and Hoch, M. C. (2015). Clinical and laboratory measures associated with health-related quality of life in individuals with chronic ankle instability. *Physical Therapy in Sport*, 16(2):169–175.

Nicholas, M. K. (2007). The pain self-efficacy questionnaire: Taking pain into account. *European Journal of Pain*, 11(2):153–163.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77.

Stahl, T. and Völter, M. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley.

Terada, M., Pietrosimone, B. G., and Gribble, P. A. (2013). Therapeutic Interventions for Increasing Ankle Dorsiflexion After Ankle Sprain: A Systematic Review. *Journal of Athletic Training*, 48(5):696–709.

van Putte-Katier, N., van Ochten, J. M., van Middelkoop, M., Bierma-Zeinstra, S. M., and Oei, E. H. (2015). Magnetic resonance imaging abnormalities after lateral ankle trauma in injured and contralateral ankles. *European Journal of Radiology*, 84(12):2586–2592.

Voelter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L. C. L., Visser, E., and Wachsmuth, G. (2013). *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. dslbook.org.