Semantic-based system for exercise programming and dietary advice

Givi-Giorgi Mamatsashvili[1], Konrad Ponichtera[1], Mikołaj Małkiński[1], Maria Ganzha[1,2], and Marcin Paprzycki[2,3]

[1] Warsaw University of Technology, Warsaw, Poland
[2] Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland
[3] Warsaw Management University, Warsaw, Poland

Abstract. Growing health awareness, results in interest in healthy eating and "fitness". While information about exercising and dieting is readily accessible, it is difficult, for an inexperienced person, to find what is right for her/him in terms of *both* diet and exercise. The aim of this work is to introduce a system, based on semantic technologies, which addresses user goals with joint exercise programs and meal suggestions.

Keywords: Ontology · Semantic technologies · Inference · Nutrition · Fitness · Cloud · Scalability · High Availability

1 Introduction

Recently, people are, more than ever, conscious about diet and exercise. This results in creation of multiple "fitness tools". Since numerous exercise programs and diet plans can be found online (or as smartphone apps, Section 2), it seem that there is no need for yet another application. However, while certain exercise programs work for some, they might not work for other users. Therefore, existing predefined programs might not be "optimal" for a given individual. Note also that such programs ignore current health and/or experience, and fitness goals of the user. Here, personal trainer can help, but such service can be expensive.

Overall, while human input seems needed, the question arises: how to minimize it (before it becomes counter productive). Here, applications should generate exercise programs, while allowing for customization (where variants should work as well as the original one). Similarly, diets should offer variants, as people do not want to follow a strictly-defined diet. Hence, an application should collect personalizing information during registration, and use it to generate advice.

Take, for example, an intermediate lifter who wishes to get stronger. Assume that (s)he formulates goal(s), history, and characteristics. Here, a personalized program should be developed, aimed at development muscles that (s)he specified, while suggesting low-calorie meals, to help loose weight. Moreover, if (s)he does not like certain exercise(s), application should propose a modified program, which still takes into account her/his preferences.

To develop needed application, domain knowledge must be captured. Here, we have selected ontologies, to represent exercises, meals, ingredients, muscles, etc., and their relations (in context of fitness). Existence of an ontology allows application of semantic technologies to "ask right questions" and infer answers.

In following sections, we discuss (1) related works, (ii) design and implementation of the proposed application, and (iii) present an example how it works.

2 Related work

Let us start with fitness applications. Here, Stronglifts[4] provides exercise programs and ability to track them. It suggests weights adjustments, e.g. when to decrease the weights, during certain exercises. It is possible to log details of workouts, and export logs to other devices. However, user preferences are not considered. Furthermore, support for experienced lifters is limited.

Freeletics[5] specializes in nutrition and fitness. Here, few features are free, but more advanced ones require subscription. In the application, one has access to various workouts and tools, such as nutrition guidance and a digital AI coach.

Both these apps can be a great resources for newcomers. However, they lack adaptivity, flexibility and features for experienced users. Although Freeletics features the digital coach, it does not *combine* exercises with meal suggestions.

Semantic technologies have been used in relevant contexts. Authors of [7], discuss how to help to understand diet. Proposed agent system relies on an ontology containing nutritional information about foods, collected from convenience stores in Taiwan. Here, users "inform the system" what food they consume. This information is analyzed and confronted with what is determined to be a "healthy diet". Overall, this is a good example how ontologies can support better understanding of health.

Similarly, approach presented in [6] discusses use of biomedical ontology, to support clinical decisions. Here, the ontology, containing relevant clinical information, aids detecting irregularities, such as wrong diagnoses, unobserved diseases, etc. Due to the intricacy of the problem, ontologies are used to represent complex knowledge. This illustrates how a semantic technologies can be used to develop an assistant, tasked with aiding the user.

3 Proposed approach

In this context, we have decided to use ontologies to represent knowledge about physical exercises and nutrition. The proposed system consists of two main components, related to exercise and dietary advice.

3.1 Knowledge representation

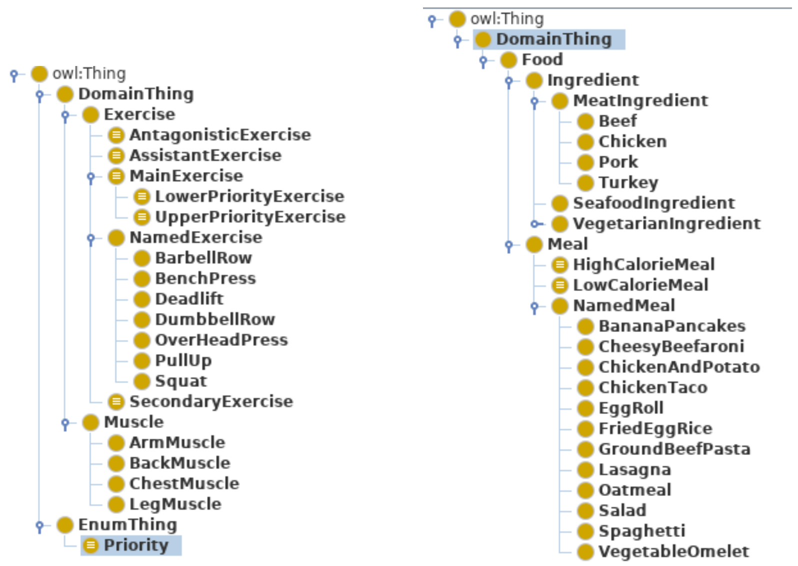
In [4], an ontology is defined as an explicit specification of a conceptualization. Overall, ontologies formally represent knowledge, by capturing entities existing

[4] https://stronglifts.com/

[5] https://www.freeletics.com/

in a given domain, and expressing relations between them. Both, entities and relations have names, allowing to represent contexts for any entity ("its place" within the domain/world). Ontologies are expressed in standardized formats (RDF, RDFS and OWL), which can be interpreted by a machine.

Our solution introduces two ontologies, representing "fitness" and "food". Ontology of fitness captures relations between physical exercises and muscles (body function), while food ontology models nutritional attributes of meals. Ontologies, forming knowledge base, are independent from the remaining parts of the application and can be modified, without changing other components of the system. Furthermore, stored knowledge can be shared between applications, or exposed to the outside world (e.g. within Linked Open Data[6]) to be (re)used.



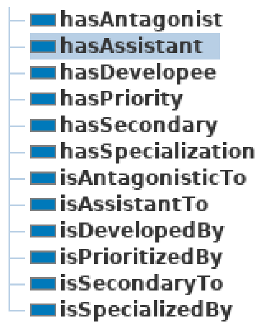
(a) Exercise ontology (b) Food ontology

Fig. 1: Exercise and food ontologies – top level concepts

Exercise ontology To the best of our knowledge, there are no ontologies of physical exercises. Hence, we have developed our own (in case one exists, our ontology can be replaced by it). It is based on a specific "exercise philosophy", well recognized in the community. Here, exercises are classified as *antagonistic*, *assistant*, *main* or *secondary*, assigned into groups, and arranged in a specific order. Moreover, different arrangements are recommended, based on users experience: *beginner*, *intermediate* or *advanced*. Figure 1a (left panel) illustrates

[6] https://lod-cloud.net

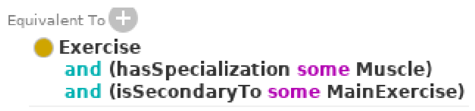
how these concepts are represented in the exercise ontology. The top level concept is *DomainThing*. It has two subclasses: *Exercise* and *Muscle*. The former contains one *primitive* class, the *NamedExercise*, which has subclasses for each exercise type. Exercise classes can have, from one to many, instances, capturing their variations. For example, *Deadlift* has two instances: *RomanianDeadlift* and *StiffLeggedDeadlift*, which are similar, but technically different. Note that this is not a comprehensive ontology of fitness. We have developed it to the point where its usefulness can be shown. Hence, if the proposed approach is to become a real-world application, this ontology has to be further developed.



hasAntagonist
hasAssistant
hasDevelopee
hasPriority
hasSecondary
hasSpecialization
isAntagonisticTo
isAssistantTo
isDevelopedBy
isPrioritizedBy
isSecondaryTo
isSpecializedBy

Fig. 2: Object properties from the exercise ontology

All instances in exercise ontology can have object properties, listed in Figure 2. Based on these properties, an exercise is inferred as belonging to one of the following *defined* classes: *AntagonisticExercise*, *AssistantExercise*, *MainExercise*, or *SecondaryExercise*. Namely, an exercise is a *MainExercise* if it has *lower* or *upper* priority, represented by an attribute *hasPriority*, or a *SecondaryExercise* if it develops a specific muscle and is secondary to a main exercise, represented by attributes *hasDevelopee* and *isSecondaryTo*. Additionally, each *MainExercise* can be classified as either *Lower* or *UpperPriorityExercise*. This division is possible thanks to a sibling class of *DomainThing*, called *EnumThing*, which has a single subclass *Priority*. These rules are described as equivalence statements for the defined classes (see, Figure 3). Here, *defined* classes, and *object properties*, are key to the exercise program generation algorithm.



```
Exercise
 and (hasSpecialization some Muscle)
 and (isSecondaryTo some MainExercise)
```

Fig. 3: Definition of a *SecondaryExercise*

Users can specify muscles they want to develop. For this purpose, we have introduced the *Muscle* class, with four subclasses: *ArmMuscle*, *BackMuscle*, *ChestMuscle*, and *LegMuscle*. Each individual muscle, e.g. *Triceps*, is an instance of a subclass. This allows capturing relationships between exercises and muscles.

Food ontology Several ontologies deal with food/nutrition (see, also, Section 2). However, they are either too detailed ([2]), or focused on a "problem not relevant in our context". Moreover, choices like [1], do not capture all factors needed for meal recommendations. Hence, we have created an ontology, representing meal ingredients, their calories, and how they are composed into meals. The structure of the food ontology, is presented in Figure 1b. Here, ingredients have been separated into: *MeatIngredient*, *SeafoodIngredient* and *VegetarianIngredient* classes. Each of them has subclasses. For instance, class *MeatIngredient* has the following subclasses: *Beef*, *Chicken*, *Pork* and *Turkey*.

Composition of ingredients into meals is expressed through instances of subclasses of the *NamedMeal* class, e.g., class *Salad* has instances, representing different salads (with slightly different ingredients). Ingredients are represented through *hasIngredient* property. Additionally, meal instances have an approximate number of calories, described using *hasCalories* property.

Again, this is a "minimal ontology" developed for the prototype, and should be viewed as such. However, it allows to capture user preferences; e.g. availability of number of calories, allows recommending meals appropriate to lose, maintain or gain weight. Moreover, structure of the ontology, allows dealing with special requirements, such as food allergies or suggesting vegan meals (see, [5]). Finally, it can allow suggesting meals composed of ingredients present in the users kitchen (in presence of an IoT-enabled refrigerator, see Section 4).

4 Technical aspects

Let us now briefly describe the key technical aspects of the developed system and its components.

4.1 System description

The application should be accessible from a web browser on a desktop, or a mobile device. Hence, the interface has been created with Angular framework. The server-side part was developed in Java, using Spring Boot suite. It consists of two modules: *Core* and Internet of Things (*IoT*), depicted in Figure 4. *Core* takes care of all aspects of the application, e.g. connection with the database, serving Angular interface, creating and authenticating users, managing session and handling user requests. The *IoT* is to deal with home appliances, such as smart refrigerators. It serves as a proxy between *Core* and appliance(s). Currently, it allows only delivering individual dietary advice, where meals are composed from ingredients found in "smart fridges".

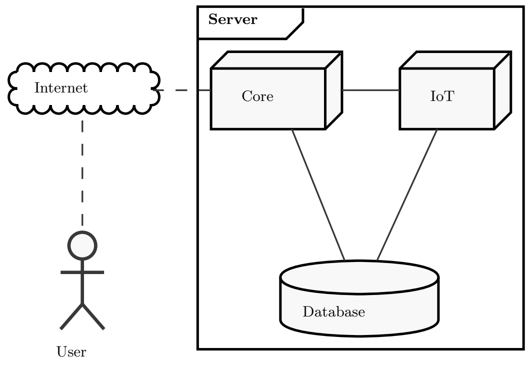


Fig. 4: Diagram showcasing individual parts the solution is composed of. In the IoT module, vendor API connection is replaced with mocked database.

Proposed approach allows separation of business logic (*Core* module), from handling external interfaces. Since modules are "independent", if one of them requires update (e.g. due to change in vendor API), it can be updated without altering the other one. Currently, for testing purposes, the *IoT* module is emulated by a database with information about ingredients "found in smart fridge".

4.2 Application of Semantic Technologies

Presented solution uses ontologies (from two domains) and reasoning (see, Section 3). Ontologies are represented in OWL, while operations on them use Apache Jena[7], which allows formulating SPARQL[8] queries. A sample SPARQL query, returning all exercises "assistant to classic bench press", is depicted in Figure 5.

```
PREFIX sf: <http://www.stayfit.oof/ontologies/exercise#>
SELECT ?exercise
WHERE {
    ?exercise sf:isAssistantTo sf:ClassicBenchPress
}
```

Fig. 5: Sample SPARQL query, listing exercises assistant to classic bench press.

Query can be executed against the ontology graph, or obtained through an inference process (see, Section 4.3). The latter is used for extraction of knowledge, which was not explicitly stated during ontology creation. Moreover, Semantic

[7] https://jena.apache.org/
[8] https://www.w3.org/TR/rdf-sparql-query/

Web Rule Language (SWRL[9]) was used to express logical implications, inferring additional triples. An example SWRL rule is presented in Figure 6.

```
sf:MainExercise(?e) ^ sf:Deadlift(?e)
-> sf:isAntagonisticTo(sf:HipHinge, ?e)
```

Fig. 6: Example SWRL rule. It states that if exercise e is a *main exercise* and a *deadlift*, a *hip hinge exercise* will be *antagonistic* to e.

The left-hand side of the implication contains prerequisites for the right-hand side to hold. In this case, if exercise e is a main exercise and is of deadlift type, then every hip hinge exercise will be antagonistic to it.

It is possible to build not only primitive rules, but also defined ones, like `MainExercise`, subclasses of which are populated during reasoning, as opposed to primitive classes where their subclasses are found explicitly in the ontology. This significantly increases reasoning capabilities and allows to "keep ontology clean" by splitting reasoning into taxonomy-based and rule-based. However, misuse of SWRL rules might lead to reasoning problems. Hence, every rule should be very well documented, so that its selective testing is possible.

4.3 Inference

The proposed approach depends on semantic reasoning and use of SWRL. Here, we have found that the native reasoner built into Apache Jena does not support SWRL. Thus we have used an external reasoner, the Openllet[10] (fork of Pellet[11]), capable of performing both OWL DL reasoning, and resolving SWRL rules.

4.4 Scalability and high availability

Today, monolithic solutions are being replaced by microservice-based. This results in creation of deployment methods independent from physical machines [3]. The key factor is fast launching of new instances, without excessive and time-consuming configuration. One of possible approaches is containerization. Containers are similar to virtual machines, but are created from templates and, usually, don't have pre-allocated resources. In the developed solution all components were containerized using Docker[12] container engine.

Although containers do not offer the same level of isolation as the virtual machines, they became an important building block of cloud environments. However, they are still just a runtime method. They do not provide load balancing,

[9] https://www.w3.org/Submission/SWRL/
[10] https://github.com/Galigator/openllet
[11] https://github.com/stardog-union/pellet
[12] https://www.docker.com

high availability, or failure recovery. This created demand for another layer of abstraction, for dynamic creation and management of the pool of containers (which became known as container orchestration). For this purpose we have decided to use Kubernetes[13] orchestrator. It takes care of managing the pool of physical container hosts and deploying the application in a highly available manner (resilient to failure). The example of Kubernetes deployment, of the presented solution is showcased in Figure 7.

Having the cluster configured, it is possible to deploy the whole solution at once. Orchestrator will try to evenly schedule the pods among cluster nodes in order to achieve the best availability in case of node failure. All the pods communicate with each other thanks to the virtual pod network which spans among all nodes of the cluster. User, trying to load the application, sends a request to the global load balancer. It can be configured on the cluster or rented from the cloud service provider. The global load balancer ensures uniform load distribution among pods of the same type, so that no pod is flooded with excess amount of requests.

Due to the modular structure, it is easy to add nodes to the cluster, to increase overall computing power, and to secure availability in case of node failure. Kubernetes tries to automatically ensure that all deployed applications are available and reschedules pods from the dead nodes on the remaining ones.

Choosing Docker and Kubernetes, as fundamental for the deployment, resulted in creating extremely scalable and reliable environment, ready for the cloud-oriented market. What's more, due to adoption of standardized open source solutions, it was possible to achieve that without falling into vendor lock-in, typical for Platform as a Service (PaaS) environments.

Obviously, such complex deployment was not needed for the demostrator. However, we have decided to ensure that a large-scale deployment is possible. We have tested the infrastructure and found that, as expected, it was as scalable and as resilient as the Docker+Kubernetes pair guarantees (see, Section 5).

5 Experimental verification

The proposed application has been thoroughly tested for various usage scenarios. Let us discuss a simple example of a typical usage, showcasing the main functionality of the application, along with its current shortcomings.

5.1 Typical usage scenario

Let us assume that user is a 23 year old male, 179 cm tall, who weighs 87 kg. He creates an account and provides the above data during registration. He also also states that he wishes to lose weight, and have only a basic training – he has time to train only few times per week, for a limited number of hours. Furthermore, he identifies himself as an intermediate lifter, and chooses to train hamstrings during deadlifts, triceps during bench presses, upper back during overhead presses,

[13] https://kubernetes.io

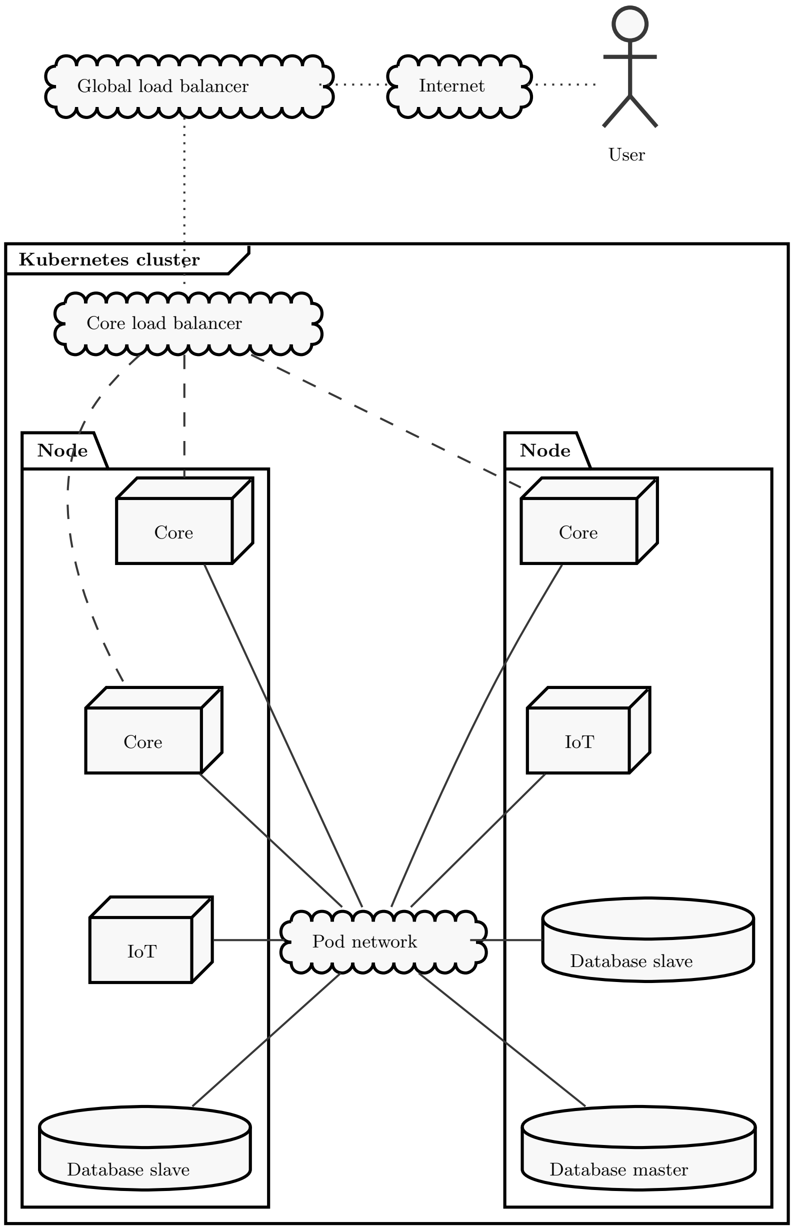
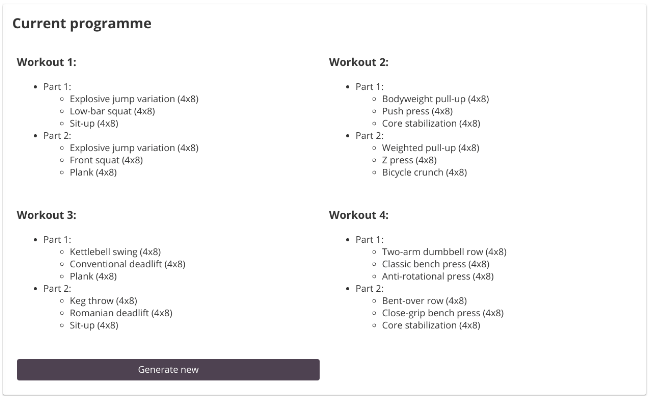


Fig. 7: Diagram example double node Kubernetes deployment of three instances of core module and two instances of IoT module as well as the master-slave database. The presented IoT module usage is the testing one, where vendor API connection is replaced with mocked database.

and quadriceps during squats. Once the registration is complete, application generates an exercise program, which may look as in Figure 8.



Current programme

Workout 1:

- Part 1:
  - Explosive jump variation (4x8)
  - Low-bar squat (4x8)
  - Sit-up (4x8)
- Part 2:
  - Explosive jump variation (4x8)
  - Front squat (4x8)
  - Plank (4x8)

Workout 2:

- Part 1:
  - Bodyweight pull-up (4x8)
  - Push press (4x8)
  - Core stabilization (4x8)
- Part 2:
  - Weighted pull-up (4x8)
  - Z press (4x8)
  - Bicycle crunch (4x8)

Workout 3:

- Part 1:
  - Kettlebell swing (4x8)
  - Conventional deadlift (4x8)
  - Plank (4x8)
- Part 2:
  - Keg throw (4x8)
  - Romanian deadlift (4x8)
  - Sit-up (4x8)

Workout 4:

- Part 1:
  - Two-arm dumbbell row (4x8)
  - Classic bench press (4x8)
  - Anti-rotational press (4x8)
- Part 2:
  - Bent-over row (4x8)
  - Close-grip bench press (4x8)
  - Core stabilization (4x8)

Fig. 8: Example exercise program, generated for intermediate user

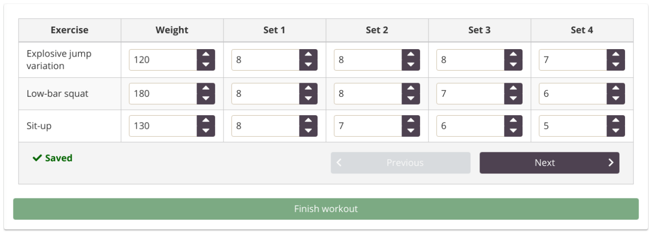
Proposed program includes four workouts, each of them based on specific barbell movements: Bench Press, Deadlift, Overhead Press and Squat. Each of these workouts includes a main exercise, primarily used the strength and body development as it requires use of a variety of muscles. These exercises are preceded by antagonistic exercises, i.e. exercises focusing on the muscles, which oppose those used during the main exercise. As a third exercise, an assistant exercise, which helps strengthening of the core for a smoother training experience is suggested. The latter half of the workout is very similar to the former. Once again, we have antagonistic and assistant exercises, but the main exercise is replaced by a secondary exercise. The secondary exercise is a variation of the main exercise, and specializes in developing the user-specified muscle group.

Main exercises have priority specified, i.e. if we declare some exercise to prioritize lower or upper part of the body, it can be treated as a main exercise. Here, the inference is done by checking the property of the existence. Antagonistic exercises are inferred through SWRL rules. For each exercise, where antagonism can be declared there is a dedicated SWRL rule, which specifies classes of exercises. For example, hip hinge is antagonistic to each exercise e, which is also a deadlift. Note that e is *not* explicitly asserted as a main exercise. This fact is resolved by the reasoner first, so that the SWRL rule engine will be provided with an already inferred taxonomy. Similarly in the case of assistant exercises; each of them has a dedicated SWRL rule, which defines the assistance to some main exercise. For instance, if some exercise prioritizes lower part of the body (and, by definition, is a main exercise), sit ups are known to be assistant to it. Inference of secondary exercises is also done by defining an SWRL rule for each such exercise (having decline bench press, it is secondary to some exercise if that exercise is main one and a bench press).

Muscles are individuals, divided into four classes: arm, back, chest and leg muscles. They can be specialized by some exercises. That fact is resolved in two ways; either by explicitly asserting that some individual exercise has specialization of an individual muscle (e.g. front squat has specialization of quadriceps) or by using SWRL rules. Due to limitation of OWL DL, it is not possible to assert a triple of the form individual-property-class, e.g. incline bench press (individual) has specialization (property) of chest muscle (class). As a workaround, it is possible to apply SWRL rules, to state that if some muscle is a chest one, it is specialized by the incline bench press. Because "is specialized by" and "has specialization" properties are inverses of each other, having such rule defined, the SWRL engine resolves that each incline bench press has specialization of every asserted and inferred chest muscle, which is the desired result.

In summary, we can state that the inference on the exercise ontology worked just as expected. An intermediate lifter has been presented with program consisting of four workouts, each with relevant exercises.

Having the program generated, user is able to log the workout in the application, as he has completed it, by stating the number of successful repetitions of each prescribed set (see, Figure 9).



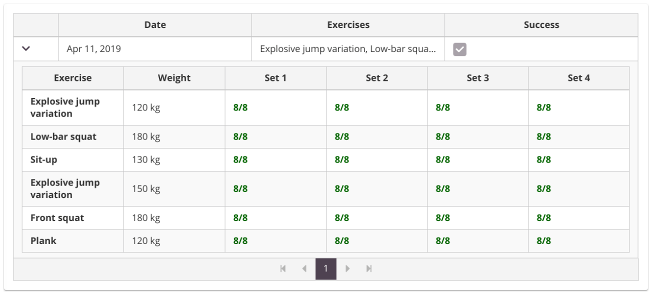
Exercise	Weight	Set 1	Set 2	Set 3	Set 4
Explosive jump variation	120	8	8	8	7
Low-bar squat	180	8	8	7	6
Sit-up	130	8	7	6	5

Fig. 9: The view of ongoing workout

Assuming that 8 successful repetitions took place during each set, once the user finishes his workout, a new entry will appear in the workout history dashboard (see, Figure 10).

Specifically, the newly-registered session will be marked as successful, due to the fact that 8 repetitions were completed. In the case when the session was not successful, it will still be registered in the history, along with the number of completed repetitions.

Separately, in the meal advice view (see, Figure 11) the user is presented with three meal choices – chicken taco, peanut butter oatmeal, and an egg roll. The meals, shown on the screen, were all described as having less than 650 calories per serving. This property has been used to classify them as low-calorie meals. The suggestion is made based on the fact, that the user is attempting to lose weight (see, above). In this case, user should be presented with only low calorie meals. Similarly, the user who wants to gain weight would be presented with meals inferred to be high-calorie ones (having more than 750 calories per serving)

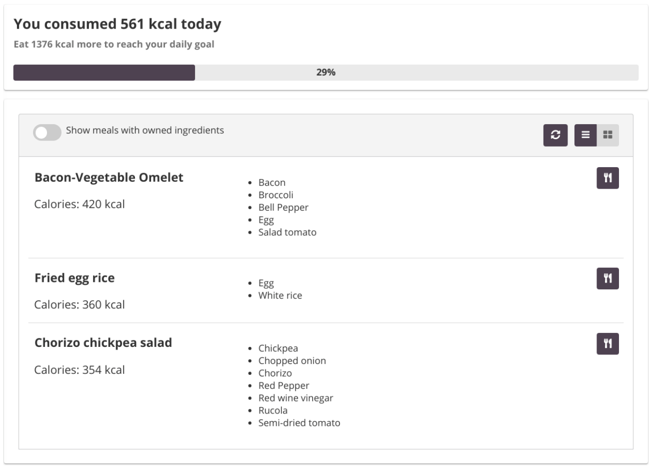


	Date	Exercises	Success
⌄	Apr 11, 2019	Explosive jump variation, Low-bar squa...	✓

Exercise	Weight	Set 1	Set 2	Set 3	Set 4
Explosive jump variation	120 kg	8/8	8/8	8/8	8/8
Low-bar squat	180 kg	8/8	8/8	8/8	8/8
Sit-up	130 kg	8/8	8/8	8/8	8/8
Explosive jump variation	150 kg	8/8	8/8	8/8	8/8
Front squat	180 kg	8/8	8/8	8/8	8/8
Plank	120 kg	8/8	8/8	8/8	8/8

Fig. 10: Successfully finished workout

while in case of "weight maintaining" all types of meals would be presented, including high calorie, low calorie and those in between. It is up to the users to control the amount of calories they consumed on the given day and reach their daily calorie goal. This is why they are presented with a progress bar they can fill by marking the meals they consumed.



You consumed 561 kcal today

Eat 1376 kcal more to reach your daily goal

29%

Show meals with owned ingredients

Bacon-Vegetable Omelet

Calories: 420 kcal

- Bacon
- Broccoli
- Bell Pepper
- Egg
- Salad tomato

Fried egg rice

Calories: 360 kcal

- Egg
- White rice

Chorizo chickpea salad

Calories: 354 kcal

- Chickpea
- Chopped onion
- Chorizo
- Red Pepper
- Red wine vinegar
- Rucola
- Semi-dried tomato

Fig. 11: Meal advice screen, showing amount of consumed calories on a given day, as well as other meal propositions

Each meal has a list of ingredients, along with an approximate number of calories. When the user marks the egg roll, as the consumed meal, the distance from reaching daily calorie goal changes from 1937 to 1374 (as egg roll has approximately 561 calories). This means the user have already consumed the 29% of calories he should consume on the given day, in order to lose weight.

Meals are OWL individuals, of specific primitive classes, having constraints in the form of superclasses. For example, a peanut butter oatmeal is an individual of type *Oatmeal*, which is a subclass of a meal that has oat flakes as an ingredient. Since ingredients are also individuals of some classes formed into a taxonomy (for example, bacon, which is known to be meat) there is a broad possibility for extending the ontology and program functionality to include potential user allergies and intolerances, as well as individual preferences (see, also [5]).

Users have also access to meals that can be made by the ingredients stored in their smart fridges (but these would be suggested in a separate view of the application; see, also, Section 4.1).

5.2 Kubernetes' high availability scenario

Separately, we have tested use of Docker+Kubernets. Due to Kubernetes' health probing mechanisms, the whole system should be immune to failures of its components, no matter if the failure occurred on the application or machine level.

The fundamental high availability principle requires that each component of the developed solution is available in at least two instances. It can be easily seen in the Figure 7 that, no matter which logical or physical component of the solution is taken down, the whole system is still going to function normally, without users knowing that some incident has occurred on the server side.

Software failure scenario Software failure can occur in the case when application in one of the Kubernetes' pods crashes (e.g. due to lack of available memory) or becomes unable to serve the requests properly (e.g. loses connection with the database, or the IoT vendor servers). It can be easily simulated by enforcing pod shutdown. Although Kubernetes will immediately proceed with creation of a new pod instance, it will take some time before it "makes sure" that the pod is working properly and won't redirect the requests to it, in the meantime. One can try to use the functionality of the killed pod, before the new one becomes ready. We have tested that in the *Core* module pod, which died, it was still possible to perform operations like logging in, generating new workout, etc. Similarly, hen the IoT module pod was taken down, it was still possible to request dietary advice, composed of ingredients from the emulated refrigerator.

Hardware failure scenario Hardware failure is more serious because, unlike in case of software one, Kubernetes is not able to revive unavailable node; it has to be done manually by the administrator. The failure can be simulated either by detaching the network cable from the node, or simply by unplugging its power cord. All the pods of the node will become detached and unable to process forwarded requests. The control plane will probe the node and then mark it as unavailable, which will result in all internal load balancers to omit that node during traffic handling. In the meantime, the cluster itself will remain operational, having at least one pod of each type deployed on the remaining nodes. Also, Kubernetes might schedule pods from failed node on the remaining ones, in order to enhance load balancing. When all of the mentioned things are happening, user is not able to notice that a failure and a recovery occur on the server side. In the worst case, if the user's request was meant to be just handled by the pod from the failed node, there might be a slight delay in request serving, while the cluster's internal mechanisms redirect the requests to the working node. The delay is linearly decreasing as the amount of worker nodes of the cluster increases, because the probability that user will be served by that particular node decreases proportionally to the cluster's size.

It is worth noting that administrator's intervention is required only in the case of bare-metal cluster, deployed manually on physical machines. In case when the cluster is provisioned by the cloud provider, recovery from the node failure is going to be automated, just as in case of software pod failure.

Maintenance scenario Cluster nodes might also need a maintenance, which requires disabling their pod scheduling capabilities and gracefully taking down all the already working pods. This process is called node cordoning and draining. Nodes that are shut down are treated just like failed ones, with the exception that Kubernetes is not probing them to check if they are available again. During node draining, each evicted pod has its copy created (on one of the remaining nodes) so that user is not able to notice that some server instances are being shut down. Once the maintenance is complete, nodes can be uncordoned and reattached to the cluster, which will proceed with pod scheduling.

In summary, all performed tests illustrated that the Docker+Kubernets infrastructure is capable of efficiently "protecting" the running application.

5.3 Shortcomings and limitations

One of the the main assumptions, behind the developed application is to minimize the human involvement in providing fitness and dietary advice. This is to be achieved by utilizing semantic technologies and ontologically capturing domains of interest into a semantic knowledge base. However, the decision-making is not perfect. While the generated exercise programs might be efficient in terms of achieving one's weight goal, user satisfaction cannot be guaranteed, as there are far too many factors that the application cannot foresee (being in its current form). It is also worth noting that there is no way to guarantee accuracy of daily caloric intake provided by the system, as every person is unique. This can cause the numbers to be significantly off for some people. This will also require users to re-adjust the number of calories, by experimenting with the suggestions, which basically means human-involvement the application tries to get rid of int he first place.

However, most importantly, the application is not universal. The exercise ontology, used in it, adopts a certain exercise philosophy and, while being capable of generating numerous programs, it lacks variety that users might be looking for. Moreover, users may believe that a different exercise philosophy would be better for them. This problem can be solve, by replacing fitness ontology by a different one (with the remaining parts of the application unchanged).

As noted, current version of food ontology does not take account food allergies, or strict diets, as its based solely around the idea of calories and classifying food based the calorie count. Acknowledging these limitations, it is crucial to take into account that this application is only a prototype and can be improved in numerous ways. Furthermore, it was designed in such a way that incorporating such changes should be relatively easy, e.g. due to modular design and extensible nature of ontologies.

6 Concluding remarks

In this paper we have presented a semantic-based system for exercise-programming and dietary advice. A demonstrator has been developed, in order to test the presented solution and to better understand the needs of the users. The system successfully generated exercise programs, based on provided preferences, while also suggesting some meals that the user could be interested in.

Shortcomings and limitations discussed above, will be prioritized in the upcoming versions. Moreover, the system can be further improved by adding more knowledge to the ontologies, more specifically, we aim to improve the food ontology so that it can satisfy needs of various users. This can be planned by considering the diets the users are following, such as plant-based or ketogenic diets, or the foods they are unable to consume due to certain intolerance and allergies. The system can also be integrated with personal assistants like Amazon Alexa or Google Assistant for convenience. Our goal is to expand and further improve the solution as the time goes, which is made possible thanks to the implementation and the technologies used.

References

1. Pizza ontology. https://protege.stanford.edu/ontologies/pizza/pizza.owl
2. Celik, D.: Foodwiki: Ontology-driven mobile safe food consumption system. TheScientificWorldJournal **2015**, 475410 (07 2015). https://doi.org/10.1155/2015/475410
3. Chen, G.: The Rise of the Enterprise Container Platform. IDC White Paper (July 2018)
4. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. **5**(2), 199–220 (Jun 1993). https://doi.org/10.1006/knac.1993.1008, http://dx.doi.org/10.1006/knac.1993.1008
5. Ponichtera, K., Małkiński, M., Sawicki, J., Ganzha, M., Paprzycki, M.: Fusing individual choices into a group decision with help of software agents and semantic technologies. in press (2019)
6. Riaño, D., Real, F., López-Vallverdú, J.A., Campana, F., Ercolani, S., Mecocci, P., Annicchiarico, R., Caltagirone, C.: An ontology-based personalization of health-care knowledge to support clinical decisions for chronically ill patients. Journal of Biomedical Informatics **45**(3), 429 – 446 (2012)
7. Wang, M.H., Lee, C.S., Hsieh, K.L., Hsu, C.Y., Acampora, G., Chang, C.C.: Ontology-based multi-agents for intelligent healthcare applications. J. Ambient Intelligence and Humanized Computing **1**, 111–131 (06 2010). https://doi.org/10.1007/s12652-010-0011-5