

АКАДЕМИЯ МАРКЕТИНГА И СОЦИАЛЬНО-ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ – ИМСИТ (г. Краснодар)

Факультет информатики и вычислительной техники

Кафедра математики и вычислительной техники

«Допустить к защите»

Зав. кафедрой М и ВТ

_____ Н. С. Нестерова

«___» _____ 20__ г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему «Разработка программного обеспечения планирования тренировок для
спортивных компьютеров»

(по материалам НАН ЧОУ ВО «Академия маркетинга и социально-
информационных технологий – ИМСИТ» (г. Краснодар), г. Краснодар)

Направление 09.03.01 «Информатика и вычислительная техника»

Направленность «Автоматизированные системы

обработки информации и управления»

Работу выполнил

студент 5 курса

заочной формы обучения

группы 13-ЗИВТспо-01

Ябчинский Антон

Владимирович

Научный руководитель:

к. т. н., доцент

К. Н. Цебренко

Краснодар

2016

АКАДЕМИЯ МАРКЕТИНГА И СОЦИАЛЬНО-ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ – ИМСИТ (г. Краснодар)

Факультет информатики и вычислительной техники
Кафедра математики и вычислительной техники

«Утверждаю»
Заведующий кафедрой

«___» 20__ г.

ЗАДАНИЕ
на выпускную квалификационную работу

Студенту Ябчинскому Антону Владимировичу

Тема выпускной квалификационной работы «Разработка программного обеспечения планирования тренировок для спортивных компьютеров» (по материалам НЧОУ ВО Академия маркетинга и социально-информационных технологий ИМСИТ)

Утверждена приказом ректора от 10 марта 2016 г. № 226.

Основные вопросы, подлежащие разработке: анализ предметной области и постановка задачи, построение модели предметной области, создание программного обеспечения, разработка программного кода.

Основная литература:

1. Мински, Я. Программирование на языке OCaml [Текст] / Я. Мински, А. Мадхавапедди, Д. Хикки. — М.: ДМК Пресс, 2014. — 536 с.
2. Фаулер, М. Предметно-ориентированные языки программирования [Текст] / М. Фаулер. — М.: Вильямс, 2011. — 576 с.
3. Компиляторы: принципы, технологии и инструментарий [Текст] / А. Ахо, М. Лам, Р. Сети, Д. Ульман. — 2-е изд. — М.: Вильямс, 2015. — 1184 с.

Срок представления законченной работы 11 мая 2016 г.

Дата выдачи задания: 12 ноября 2015 г.

Руководитель _____ /Цебренко К. Н. /
подпись Ф. И. О.

Задание получил: 12 ноября 2015 г.

Студент _____ /Ябчинский А. В. /
подпись Ф. И. О.

РЕФЕРАТ

Выпускная квалификационная работа: 91 с., 28 рис., 5 табл.,
25 источников, 2 прил.

ВЕБ-СЕРВИС, ГРАММАТИКА ЯЗЫКА, СПОРТИВНЫЙ КОМПЬЮТЕР,
ТРАНСЛЯТОР, ТРЕНИРОВКА, ФОРМАТ ФАЙЛА FIT, ФУНКЦИОНАЛЬНОЕ
ПРОГРАММИРОВАНИЕ, ЯЗЫК ПРЕДМЕТНОЙ ОБЛАСТИ.

Объектом исследования являются спортивные компьютеры и описание тренировок в формате файла FIT.

Цель работы состоит в изучении возможности создания тренировок для спортивных компьютеров в формате FIT на персональном компьютере и в реализации соответствующего программного обеспечения.

К полученным результатам относятся формальный язык предметной области для описания тренировок, транслятор в формат файла FIT, веб-служба для удалённого доступа к транслятору.

Новизна результатов заключается в непротиворечивости языка предметной области, в веб-службе для удалённого создания тренировок и в режиме адресации ресурсов веб-службы.

Внедрение результатов предусматривается в спортивных школах, секциях, а также индивидуально спортсменами-любителями и пользователями спортивных компьютеров.

Эффективность работы характеризуется компактностью и непротиворечивостью языка предметной области, скоростью трансляции в формат FIT, возможностью создания тренировок удалённо при помощи веб-службы.

Определения, обозначения и сокращения

DSL — язык предметной области (англ. Domain Specific Language)

FTP — функциональная пороговая мощность (англ. Functional Threshold Power)

ЧСС — частота сердечных сокращений

каденс — частота оборотов педалей велосипеда

уд./мин — ударов в минуту

об./мин — оборотов в минуту

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	7
1 Исследование предметной области.....	10
1.1 Академия ИМСИТ.....	10
1.2 Спортивные компьютеры.....	10
1.3 Формат файлов ANT+ FIT.....	14
2 Обзор существующих продуктов.....	20
3 Требования к программному обеспечению.....	23
3.1 Постановка задачи.....	23
3.2 Анализ требований.....	24
4 Техническое задание на разработку.....	27
4.1 Введение.....	27
4.2 Назначение разработки.....	27
4.3 Требования к программе или программному изделию.....	28
4.4 Требования к программной документации.....	35
4.5 Технико-экономические показатели.....	35
4.6 Порядок контроля и приемки.....	36
5 Модель предметной области.....	37
6 Выбор метода и инструментов разработки.....	43
6.1 Язык описания тренировок.....	43
6.2 Формирование FIT файлов.....	52
6.3 Графический интерфейс пользователя.....	53
6.4 Веб-служба.....	54
7 Реализация разработки.....	56
7.1 Реализация программных модулей.....	56
7.2 Разработка интерфейса пользователя.....	61
7.3 Испытания.....	62
8 Разработка проектной документации.....	63

8.1 Руководство пользователя.....	63
8.2 Руководство системного администратора.....	71
8.3 Руководство по установке.....	77
9 Оценка эффективности программного продукта.....	78
10 Мероприятия по охране труда.....	81
10.1 Общие требования безопасности при работе за компьютером.....	81
10.2 Безопасность перед началом работы.....	83
10.3 Безопасность во время работы за компьютером.....	84
10.4 Безопасность при возникновении аварийной ситуации.....	85
10.5 Безопасность по окончании работы.....	85
ЗАКЛЮЧЕНИЕ.....	87
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	89
ПРИЛОЖЕНИЕ А Исходный код основных модулей.....	92
ПРИЛОЖЕНИЕ Б Исходный код основных модулей веб-службы.....	127

ВВЕДЕНИЕ

Согласно результатам работы государственной программы «Развитие физической культуры и спорта» в 2015 году, почти треть граждан России (а точнее 31,7 %) систематически занимаются физической культурой и спортом. И это число, вероятно, будет только увеличиваться в будущем: образ жизни всё большего числа людей не позволяет им в достаточной мере обеспечить организм физической нагрузкой, люди вынуждены создавать такую нагрузку искусственно.

Вместе с этим, всё более дешёвая носимая электроника проникает во всё большее число сфер жизни человека. Занятия физической культурой и любительским спортом не являются исключениями: любому пользователю Интернет или владельцу смартфона (а это практически каждый современный человек) доступно большое множество приложений, электронных дневников тренировок и т. д. Многие смартфоны позволяют подключить к ним различные датчики (шагомеры, датчики частоты сердечных сокращений, датчики частоты оборотов педалей велосипеда), «умные» напольные весы и т. п. Спортсменам-любителям доступны также различные специализированные спортивные и мультиспортивные компьютеры. Обычно такие устройства имеют небольшой размер и массу, часто выполняются в виде наручных часов, обеспечивают большую длительность работы по сравнению со смартфонами.

Кроме отображения на экране и записи текущих параметров тренировки, такие спортивные компьютеры и приложения зачастую позволяют также планировать тренировки: как составлять долгосрочные тренировочные планы, так и редактировать отдельные тренировки и этапы тренировок. Если в случае смартфона большой экран позволяет делать это относительно быстро и удобно, то многие специализированные спортивные компьютеры имеют маленький, иногда монохромный, экран, несколько кнопок и относительно малый объём внутренней памяти. Предпочтительней было бы создавать и хранить такие тренировки на настольном компьютере или ноутбуке и копировать в память

устройства по необходимости.

Целью данной работы является разработка программного обеспечения, позволяющего создавать и редактировать тренировки для обширной группы спортивных компьютеров — компьютеров, поддерживающих формат файлов ANT+ FIT. По данным компании Dynastream Innovations Inc., являющейся разработчиком данного формата, на сегодняшний день продаётся 618 продуктов, поддерживающих формат передачи данных ANT+ от различных производителей.

Для достижения данной цели потребуется решить ряд задач: проанализировать предметную область — спортивные компьютеры и сценарии тренировок в формате файлов FIT; выработать требования к программному обеспечению и построить модель предметной области; на основании требований и модели, выбрать подходящие для решения задачи языки программирования и инструменты; реализовать разработку и составить необходимую документацию.

Объектом исследования работы является использование технических средств отслеживания тренировок индивидуально спортсменами-любителями и в рамках подготовки спортсменов, представляющих Академию ИМСИТ на различных соревнованиях.

Предметом исследования выступают сценарии тренировок для воспроизведения на специализированных спортивных компьютерах, программное обеспечение для создания таких сценариев.

Структурно работа состоит из введения, десяти глав, заключения и двух приложений.

В главе 1 изучаются спортивные компьютеры и формат файлов сценариев тренировок.

В главе 2 оцениваются существующие программные средства, предназначенные для решений той же или схожей задачи. Выявляются их особенности, преимущества и недостатки.

В главах 3 и 4 описывается выработка требований к будущему программ-

ному обеспечению и составление формального технического задания на разработку.

В главе 5 выявляются основные сущности предметной области и связи между ними, строится непротиворечивая модель предметной области.

Выбор инструментов, методологий и языков программирования для реализации различных частей разработки описан в главе 6.

Главы 7 и 8 содержат описание реализации разработки (описание компонентов программного обеспечения и их основных модулей) и текст некоторых проектных документов: руководство пользователя и администратора, руководство по установке.

В главе 9 оценивается увеличение производительности действий пользователя при использовании разработанного программного обеспечения и приводится сравнение разработанного продукта с наиболее близким аналогом.

В приложениях представлен исходный код основных модулей программного обеспечения.

1 Исследование предметной области

1.1 Академия ИМСИТ

Академия основана 14 октября 1994 года в городе Краснодаре и является одним из крупных негосударственных вузов на юге России. Включает в свою структуру 5 факультетов, 6 кафедр, центр дополнительного образования детей и взрослых, автошколу.

Как и в других учебных заведениях высшего образования, в программу обучения входят занятия физической культурой для студентов. Целью таких занятий обычно не является улучшение физических возможностей занимающихся и подготовка к каким-либо соревнованиям. В этом случае применение каких бы то ни было специальных технических средств вряд ли оправдано.

В то же время, академия регулярно выставляет спортсменов на различных спортивных соревнованиях городского и краевого уровня. Для подготовки спортсменов на необходимом уровне, требуются регулярные тренировки и очень строгое следование программе тренировок. В календаре должны быть обозначены несколько важнейших стартов, и годовой план подготовки должен подводить спортсменов к обозначенным датам в наилучшей форме. При подобной тщательной подготовке применение технических средств может быть оправдано, и может облегчить контроль за выполнением тренировок и учёт изменяющихся показателей. Для этого могут применяться спортивные компьютеры.

1.2 Спортивные компьютеры

Полезной возможностью спортивных компьютеров является возможность программировать пользовательские тренировки. Такая программа описывает какие упражнения нужно выполнять, с какой интенсивностью и как долго. При

выполнении такой тренировки спортивный компьютер даёт указания тренирующемуся с помощью сообщений на экране и звуковых сигналов.

Программы можно создавать как непосредственно на самом устройстве, так и с помощью программ на персональном компьютере, с последующим копированием файла тренировки в память устройства. В рамках данной работы рассматриваются только спортивные компьютеры, способные работать с форматом файлов FIT. Это прежде всего спортивные компьютеры фирмы Garmin, одни из самых популярных среди пользователей.

В качестве конкретного примера спортивного компьютера можно привести велосипедный компьютер Garmin Edge 500, внешний вид которого представлен на рисунке 1.

Компьютер имеет монохромный экран с размерами $31,8 \times 37,0$ мм и разрешением 128×160 точек. Встроенная аккумуляторная батарея ёмкостью 700 мА·ч обеспечивает время работы до 18 часов. Велосипедный компьютер может подключаться к настольному компьютеру или ноутбуку через интерфейс USB, объём внутренней памяти до 50 МиБ. Диапазон рабочих температур устройства от минус 15 до 60 °C, класс влагозащиты IPX7.

Велосипедный компьютер имеет приёмник сигналов GPS для записи маршрута следования и барометрический высотомер для определения высоты над уровнем моря и подсчёта набора высоты за тренировку.

Управление всеми функциями устройства осуществляется с помощью четырёх кнопок, расположенных на корпусе. Сопряжение с дополнительными датчиками осуществляется по беспроводному протоколу ANT+ на частоте 2,4 ГГц. Устройство распознаёт следующие дополнительные датчики:

- датчик частоты сердечных сокращений (ЧСС);
- датчик частоты оборотов педалей;
- датчик частоты оборотов колеса велосипеда;
- измеритель мощности.



Рисунок 1 — Велосипедный компьютер Garmin Edge 500

Характеристики и возможности устройства описываются в [1]. Данное устройство по цене и набору возможностей можно отнести к среднему классу. В более дорогих моделях появляется цветной экран большего размера, возможность пользоваться картами местности, беспроводная синхронизация с персональными компьютерами и смартфонами. Более дешёвые модели обычно поддерживают меньшее число дополнительных датчиков, может отсутствовать GPS приёмник и барометрический высотомер.

Рассмотрим процесс создания тренировки на примере данного велосипедного компьютера. Тренировка представляет собой программу, последовательность следующих друг за другом этапов. Каждый этап характеризуется двумя главными элементами: во-первых, условием завершения данного этапа трени-

ровки; во-вторых, целевым показателем и требуемым диапазоном значений, в котором требуется этот показатель удерживать. Кроме того, для этапа можно указать является ли он этапом отдыха или нет.

Условиями завершения этапа могут являться:

- прошествие заданного времени;
- преодоление заданного расстояния;
- падение (увеличение) ЧСС ниже (выше) заданного значения;
- падение (увеличение) выдаваемой велосипедистом мощности ниже (выше) заданного значения;
- израсходование указанного числа килокалорий.

Кроме того, условие завершения этапа может отсутствовать. В этом случае этап завершается только вручную, при нажатии кнопки на корпусе устройства.

Целевыми показателями этапа тренировки могут являться:

- скорость движения;
- ЧСС;
- частота оборотов педалей (каденс);
- выдаваемая мощность.

Для скорости, ЧСС и мощности возможно либо явно указать нижнюю и верхнюю границы диапазона значений, либо выбрать одну из настроенных заранее зон. Для каденса не существует предустановленных зон, диапазон задаётся явно. При выполнении тренировки, компьютер следит за тем, что значение целевого показателя находится в заданном диапазоне, и выдаёт сообщения на экран и звуковые сигналы, если это не так.

Так же как и условие завершения этапа, целевой показатель может отсутствовать. В этом случае компьютер не отслеживает значения показателей и не выдаёт предупреждений на экран и звуковых предупреждений.

В последовательность этапов тренировки можно вводить циклы. Создание повторений осуществляется путём добавления в тренировку специального этапа, который осуществляет переход к одному из предшествующих этапов либо заданное число раз, либо пока не выполнится заданное условие. Условия в этом случае аналогичны условиям завершения этапа, описанным выше.

Программирование последовательности этапов осуществляется с помощью трёх кнопок устройства: «вниз», «вверх» и «выбор». Так при вводе числовых значений требуется кнопками «вниз» и «вверх» выбрать нужную цифру из списка для каждого из разрядов числа и подтвердить выбор кнопкой «выбор». При вводе текста (название тренировки) требуется кнопками «вниз» и «вверх» выбрать каждый символ строки.

1.3 Формат файлов ANT+ FIT

Все файлы — записанные параметры тренировки, настройки устройства, физиологические параметры пользователя, запланированные тренировки и т. д. — хранятся в универсальном двоичном формате ANT+ FIT. Протокол FIT (Flexible and Interoperable Data Transfer) специально разработан для хранения и передачи данных в различных спортивных и около-медицинских устройствах — спортивных компьютерах, весах, измерителях кровяного давления и т. п. Это компактный двоичный формат, который обеспечивает:

- оперативную совместимость данных между различными устройствами;
- масштабируемость от встраиваемых систем (например, датчик частоты сердечных сокращений) до крупных распределённых приложений;
- прямую совместимость с последующими версиями и расширяемость.

Файл FIT состоит из глобального заголовка файла, контрольной суммы и последовательности записей, каждая из которых, в свою очередь, содержит заголовок и содержимое. Содержимое записи — это либо сообщение, содержащее набор полей с некоторыми значениями, либо описательное сообщение,

определяющее количество и тип полей в сообщениях с данными.

Все возможные типы сообщений определены в глобальном профиле FIT. Из этого множества сообщений выделяют специализированные подмножества — профили, зависящие от устройства. Глобальный профиль FIT поддерживается и развивается компанией-разработчиком формата — Dynastream Innovations. Производители конкретных продуктов могут вводить свои типы сообщений, для чего зарезервирован диапазон идентификаторов сообщений. Такие сообщения будут обрабатываться только устройствами данного производителя.

Заголовок файла занимает 14 байт, заголовок одного сообщения — 1 байт. Содержимое одного сообщения может иметь произвольную длину, в зависимости от количества и типа полей, содержащихся в нём. Полностью формат FIT описан в [2, с. 16].

В соответствии с конкретным применением формата, естественным образом выделяются группы сообщений, которые сопутствуют друг другу. Например, если файл служит для хранения данных о кровяном давлении, то в нём обязательно содержатся сообщения «профиль пользователя», «информация об устройстве» и «кровяное давление».

Группа сообщений, характерных для некоторого применения, определяется типом FIT файла. Все возможные на сегодняшний день типы файлов кратко представлены в таблице 1. В полной мере особенности всех типов файлов описаны в [3, с. 11], но в рамках данной работы интерес представляет только тип файлов № 5. Рассмотрим данный тип файлов FIT подробнее.

В соответствии со спецификацией, файл должен содержать, как минимум, сообщения «file_id» (идентификатор файла), «workout» (общая информация о тренировке) и по крайней мере одно сообщение «workout_step» (описание этапа тренировки). Тренировка описывается как последовательность этапов. Каждый этап используется для описания того, какую работы необходимо выполнять в его рамках и какой длительности, либо для создания повторений. Основными

сообщениями являются сообщения «workout» и «workout_step».

Таблица 1 — Типы файлов FIT

№	Тип файла	Назначение
1	Устройство	Описание возможностей и файловой системы устройства.
2	Настройки	Описание настроек устройства и параметров пользователя (возраст, вес и т. п.)
3	Настройки спорта	Описание настроек, специфичных для конкретного вида спорта
4	Активность	Запись потоков данных и событий с датчиков
5	Тренировка	Описание структурированной активности
6	Маршрут	Описание планируемого маршрута следования
7	План	Описание графика тренировок
9	Масса тела	Запись данных весов
10	Общие данные	Обобщение всех записанных тренировок
11	Цели	Описание целей для показателей тренировок и массы тела
14	Кровяное давление	Запись данных о кровяном давлении
15	Мониторинг А	Подробная запись данных мониторинга
20	Общие данные активности	В отличие от файла активности, содержит только общие данные
28	Ежедневный мониторинг	Запись общих данных мониторинга раз в сутки
32	Мониторинг Б	Подробная запись данных мониторинга
34	Сегмент	Описание результатов виртуальных соревнований
35	Список сегментов	Описание доступных сегментов

Сообщение «workout» содержит общую информацию о тренировке в следующих полях:

- 1) вид спорта, поле sport;
- 2) требуемые возможности устройства (например, если в тренировке используется ЧСС, то устройство должно поддерживать работу с датчиками ЧСС), поле capabilities;
- 3) количество этапов, поле num_valid_steps;
- 4) название, wkt_name.

В спецификация указано более пятидесяти возможных значений для поля sport, включая такие виды спорта как гольф, охота и рыбалка. Будем рассматривать только те виды спорта, для которых спортивные компьютеры чаще всего применяются. Таким образом, поле sport может принимать следующие приемлемые значения:

- велосипедный спорт (cycling);
- бег (running);
- плавание (swimming).

Сообщение «workout_step» описывает этап тренировки:

- 1) номер этапа, поле message_index;
- 2) название этапа, wkt_step_name;
- 3) тип условия окончания, duration_type;
- 4) значение для условия окончания, duration_value;
- 5) тип целевого показателя, target_type;
- 6) значение целевого показателя, target_value;
- 7) нижняя граница целевого показателя, custom_target_value_low;
- 8) верхняя граница целевого показателя, custom_target_value_high;
- 9) интенсивность, поле intensity.

Поле intensity может принимать одно из следующих значений:

- энергично (active);
- отдых, (rest);
- разминка, (warmup);

- заминка (cooldown).

Значения полей duration_value и target_value динамические и зависят от значения поля duration_type. Вместе эти три поля описывают условие для окончания этапа тренировки или описывают условие для повторения. Все возможные варианты условий представлены в таблице 2.

Таблица 2 — Условия окончания этапа тренировки

Поле duration_type	Поле duration_value	Поле target_value
по времени	время	
по расстоянию	расстояние	
по ЧСС, меньше чем	ЧСС	
по ЧСС, больше чем	ЧСС	
по килокалориям	килокалории	
без условия		
повтор число раз	номер этапа	число раз
повтор пока не пройдёт время	номер этапа	время
повтор пока не пройдено расстояние	номер этапа	расстояние
повтор пока не потрачено калорий	номер этапа	калории
повтор пока ЧСС меньше чем	номер этапа	ЧСС
повтор пока ЧСС больше чем	номер этапа	ЧСС
повтор пока мощность меньше чем	номер этапа	мощность
повтор пока мощность больше чем	номер этапа	мощность
по мощности, меньше чем	мощность	
по мощности, больше чем	мощность	

Значения полей target_value, custom_target_value_low и custom_target_value_high зависят от значения поля target_type. Эти три поля определяют целевой показатель этапа тренировки и диапазон значений показателя. Диапазон значе-

ний либо явно задаётся в полях `custom_target_value_low` и `custom_target_value_high` (в этом случае поле `target_value` содержит ноль), либо номер предустановленной зоны в поле `target_value`. Все возможности по выбору целевого показателя для этапа тренировки показаны в таблице 3.

Таблица 3 — Целевые показатели этапа тренировки

Поле <code>target_type</code>	Поле <code>target_value</code>	Поле <code>custom_target_value_low</code>	Поле <code>custom_target_value_high</code>
скорость	номер зоны	мин. скорость	макс. скорость
ЧСС	номер зоны	мин. ЧСС	макс. ЧСС
без цели			
каденс	номер зоны	мин. каденс	макс. каденс
мощность	номер зоны	мин. мощность	макс. мощность

Значения ЧСС и мощности могут задаваться либо как абсолютные значения в уд./мин и ваттах, либо как процент от максимальной ЧСС (от 0 до 100 %) и процент от функциональной пороговой мощности (от 0 до 1000 %).

2 Обзор существующих продуктов

На текущий момент можно выделить несколько настольных приложений или веб-сервисов, позволяющих создавать сценарии тренировок для спортивных компьютеров:

- настольное приложение Garmin Training Center;
- веб-сервис Garmin Connect;
- веб-сервис Endomondo.

Настольное приложение Garmin Training Center разработано компанией Garmin в 2008 году. Приложение позволяет вести дневник тренировок, планировать тренировочные маршруты и, среди прочих возможностей, планировать тренировки и экспортить их в формате FIT.

Судя по всему, процесс создания файлов скрыт от пользователя. Приложение самостоятельно записывает выходной файл в память спортивного компьютера, который должен быть подключён к настольному компьютеру как USB накопитель. Вероятно, при этом приложение будет работать только со спортивными компьютерами Garmin, и откажется записывать файл в память других спортивных компьютеров.

Приложение работает только в операционной системе Windows, в других системах его использование затруднено или невозможно.

К 2014 году поддержка Garmin Training Center была окончательно прекращена. На 2016 год, по всей видимости, приложение окончательно удалено с веб-сайта компании Garmin и загрузить его невозможно.

Веб-сервис Garmin Connect разрабатывался как замена настольному приложению Garmin Training Center. Практически все функции настольного приложение — дневник тренировок, планирование маршрутов и т. д. — были перенесены в веб-сервис. Для работы с сервисом требуется регистрация, сервис бесплатен для пользователей.

Сервис позволяет создавать сценарии тренировок для различных видов спорта и поддерживает практически все доступные для формата FIT возможности. Редактирование сценариев тренировок осуществляется при помощи графического редактора тренировок (рисунок 2).

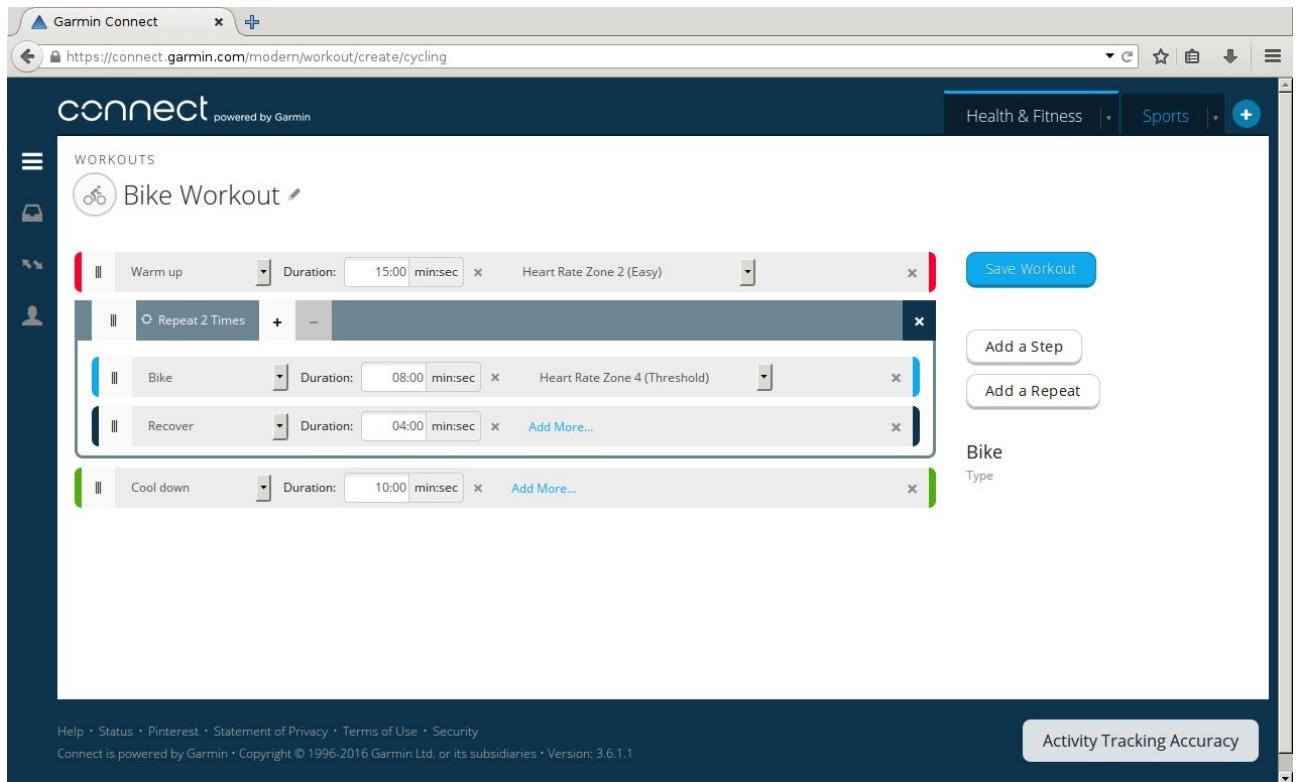


Рисунок 2 — Создание тренировки в Garmin Connect

Тем не менее одна важная возможность не реализована в редакторе тренировок Garmin Connect — сложные условия повторения последовательности шагов. Редактор позволяет добавить только повторение заданное число раз и не позволяет, например, создать повторение с условием по времени, расстоянию, килокалориям и т. д. Такие возможности предусмотрены форматом FIT и поддерживаются редакторами, встроенными в некоторые спортивные компьютеры.

Формирование файлов тренировок полностью скрыто от пользователя. Веб-сервис самостоятельно осуществляет запись файла в память спортивного компьютера и работает исключительно со устройствами фирмы Garmin.

Для работы с веб-сервисом требуется установить специальное программное обеспечение Garmin Express для сопряжения веб-обозревателя и файловой системы спортивного компьютера. Это программное обеспечение работает только в операционных системах Windows и Mac OS. В других системах экспорт тренировок в память спортивного компьютера невозможен.

Таким образом, для создания сценария тренировки с помощью Garmin Connect потребуется операционная система Windows или Mac OS, подключение к Интернет и спортивный компьютер Garmin, подключённый к настольному ПК. Возможности формата FIT используются не полностью — в сценарии тренировки возможны только простые повторения.

Веб-сервис Endomondo — один из множества дневников тренировок, доступных пользователям в Интернет. Работа с сервисом осуществляется либо через веб-обозреватель, либо посредством приложения для популярных мобильных платформ. Пользователям предлагаются бесплатная и платная версии сервиса.

Бесплатная версия имеет ограниченный набор возможностей. Для платной версии заявлена возможность создания и экспорта сценариев тренировок в формате FIT для беговых спортивных компьютеров. На текущий момент стоимость платной подписки на сервис Endomondo составляет 5,99 долларов США в месяц.

3 Требования к программному обеспечению

3.1 Постановка задачи

Требуется обеспечить возможность создавать и редактировать тренировки на персональном компьютере. Тренировки должны быть совместимы со спортивными компьютерами, поддерживающими формат файлов ANT+ FIT.

Пользователь должен иметь возможность неограниченно создавать новые и изменять ранее созданные тренировки. В любой момент времени должна быть возможность из тренировки получить FIT файл, пригодный для записи в память спортивного компьютера.

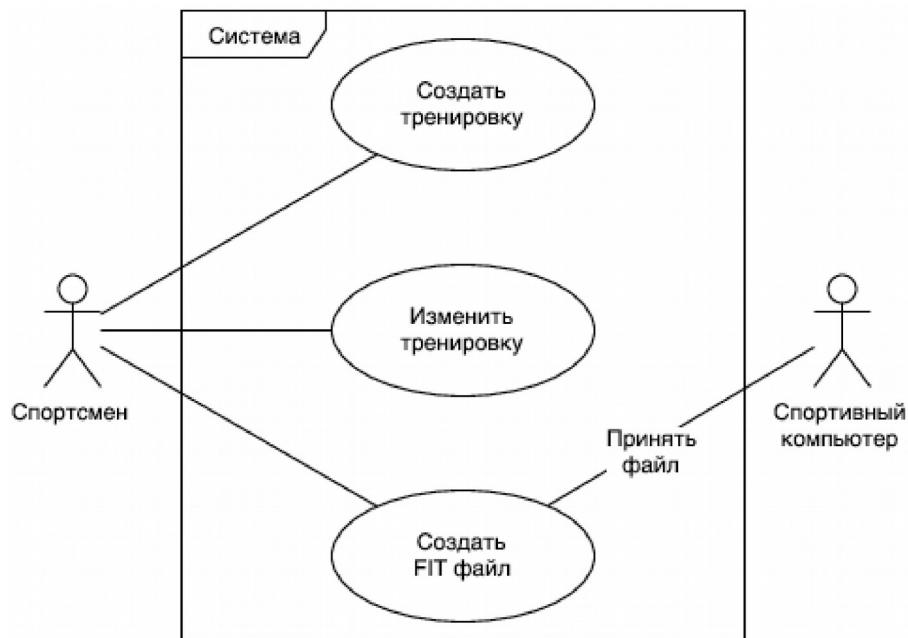


Рисунок 3 — Варианты использования программного обеспечения

Интерфейс пользователя не имеет решающего значения: программное обеспечение может как обладать графическим интерфейсом пользователя, так и быть консольным приложением с текстовым интерфейсом. Главное — это обеспечить возможность создания и редактирования тренировок.

Крайне желательно обеспечить возможность создавать тренировки через некий веб-сервис, не имея установленного локально программного обеспечения.

3.2 Анализ требований

Определяющими являются требования обеспечить создание тренировок, их редактирование и сохранение в виде FIT файлов, пригодных для записи в память спортивного компьютера.

К интерфейсу пользователя специальных требований не предъявляется: это может быть и графический, и текстовый интерфейс. Текстовый интерфейс проще для реализации и намного проще для тестирования. Следовательно, программное обеспечение должно предоставлять текстовый интерфейс пользователя. Более того, тренировки можно хранить в виде простого текста, если реализовать некий высокоуровневый формальный язык описания тренировок.

Простой текст является крайне выгодным форматом представления данных. Для создания, редактирования и просмотра таких файлов подходит любой текстовый редактор на любой платформе.

В случае специализированных для конкретных приложений форматов файлов часто бывает, что программное обеспечение перестаёт поддерживаться и больше не работает на современных операционных системах. Файлы созданные в таком приложении через много лет могут быть недоступны для чтения, и извлечение из них полезной информации может потребовать значительных усилий. Текстовый файл можно прочитать в любое время на любой платформе, даже через много лет после того, как он был создан.

Другой привлекательной особенностью простого текста является существование множества универсальных инструментов обработки и поиска текста. Например, для поиска подстрок в текстовых данных, в том числе в довольно больших объёмах, можно использовать стандартную утилиту grep командной

строки Unix; для сравнения текстовых файлов — утилиту diff и т. д. Для работы с изменяющимися текстовыми данными очень удобно использовать системы контроля версий: хранится информация обо всех версиях документа, в любой момент можно получить версию документа на любую дату, сравнить разные версии между собой и т. п. Преимущества текстового формата часто рассматриваются в различных работах о проектировании и программировании, в особенности в среде Unix. Например, в [4, с. 105] и в [5, с. 62]. А в [6, с. 135] этой теме удалена отдельная глава.

Таким образом, требуется разработать формальный компьютерный язык описания тренировки и транслятор из этого языка в двоичный формат ANT+ FIT. Для создания и изменения тренировок пользователи должны будут использовать любой текстовый редактор, а также смогут использовать любые универсальные инструменты для поиска строк в файлах описания тренировок, архивирования файлов, сравнения файлов тренировок и т. д. Такой инструмент трансляции также можно будет использовать в конвейере с другими инструментами командной строки.

Такой подход также должен положительно сказаться на надёжности и быстродействии программного обеспечения. Во-первых, такое программное обеспечение сродни чистой математической функции: результат его работы зависит только от входных данных. Такие функции очень хорошо поддаются тестированию, в том числе автоматическому. Для автоматического тестирования потребуется создать набор тестовых описаний тренировок (корректных и некорректных) и соответствующих им результатов работы транслятора (ошибка или корректный выходной файл). Во-вторых, такое программное обеспечение в значительной степени менее требовательно к быстродействию системы и объёму доступной памяти, в отличии от сложного программного обеспечения с графическим интерфейсом.

В работах по тестированию программного обеспечения (например, в [7]) указываются, что программное обеспечение хорошо поддаётся тестированию,

если оно разделено на как можно более независимые друг от друга части и если события, возникающие в ходе выполнения программы, могут быть детерминировано воспроизведены при разных запусках. Последнее требование намного проще выполнить при отсутствии графического интерфейса.

Важным требованием является создание тренировки посредством веб-сервиса, не имея локально установленного программного обеспечения. Для этого потребуется реализовать HTTP сервер, который должен будет принимать как запрос текстовое описание тренировки и создавать соответствующий FIT файл. Веб-сервер должен предоставлять страницу для ввода описания тренировки. Сервер должен запускаться на самой популярной серверной операционной системе — Linux актуальных версий. Возможность запуска на других операционных системах является желательной, но не критичной.

В будущих версиях программного обеспечения следует предусмотреть возможность редактирования тренировок в графическом представлении. Это должна быть веб-страница, обеспечивающая графический интерфейс пользователя (создание, редактирование, перемещение и удаления шагов тренировки, задание названия и вида спорта тренировки). В первоначальной версии программного обеспечения такая возможность не является критичной.

4 Техническое задание на разработку

4.1 Введение

4.1.1 Наименование программы

Наименование программы — «Система создания тренировок в формате FIT для спортивных компьютеров». Далее по тексту «система».

4.1.2 Краткая характеристика области применения

Система предназначена к применению в профильных подразделениях на объектах Заказчика.

4.2 Назначение разработки

4.2.1 Функциональное назначение

Функциональным назначением системы является предоставление пользователю возможности создания и редактирование тренировок для спортивных компьютеров в формате FIT.

4.2.2 Эксплуатационное назначение

Конечными пользователями программы должны являться сотрудники профильных подразделений объектов Заказчика.

4.3 Требования к программе или программному изделию

4.3.1 Требования к функциональным характеристикам

4.3.1.1 Требования к составу выполняемых функций

Система должна обеспечивать следующие функции:

- преобразование текстового описания тренировки на формальном языке в файл формата FIT;
- создание тренировки в формате FIT в ответ на HTTP запрос с текстовым описанием тренировки на формальном языке.

Система должна обеспечивать создание тренировок со всеми возможностями, предусмотренными форматом FIT. А именно:

- задание названия тренировки;
- задание вида спорта тренировки;
- задание шагов тренировки;
- задание повторов шагов тренировки.

Для каждого шага тренировки должна обеспечиваться следующие возможности:

- задание названия шага тренировки;
- задание интенсивности шага тренировки;
- задание условия прекращения выполнения шага тренировки или указание отсутствия условия;
- задание целевого показателя шага тренировки или указание отсутствия целевого показателя.

Для шага тренировки должна обеспечиваться возможность задать одно из следующих условий прекращения шага тренировки:

- по времени, указанному в минутах и секундах;
- по расстоянию, указанному в метрах или километрах;
- по превышению (или падению) частоты сердечных сокращений выше (или ниже) порогового значения, указанного либо в ударах в минуту, либо в процентах от максимальной частоты сердечных сокращений;
- по превышению (или падению) мощности выше (или ниже) порогового значения, указанного либо в ваттах, либо в процентах от функциональной пороговой мощности;
- по израсходованию указанного количества килокалорий.

Для шага тренировки должна обеспечиваться возможность задать один из следующих целевых показателей:

- номер зоны скорости от 1 до 10;
- диапазон скорости, указанный либо в метрах в секунду, либо в километрах в час;
- номер зоны частоты сердечных сокращений от 1 до 5;
- диапазон частоты сердечных сокращений, указанный либо в ударах в минуту, либо в процентах от максимальной частоты сердечных сокращений;
- номер зоны каденса (частоты оборотов педалей велосипеда) от 1 до 10;
- диапазон значений каденса, указанный в оборотах в минуту;
- номер зоны мощности от 1 до 7;
- диапазон мощности, указанный либо в ваттах, либо в процентах от функциональной пороговой мощности.

4.3.1.2 Требования к организации входных данных

Входные данные должны быть организованы в виде текста на формальном языке описания тренировок. Текст должен быть составлен только из печат-

ных символов, в кодировке ASCII.

К формальному языку описания тренировок специальных требований не предъявляется. Разработка языка описания тренировок осуществляется Исполнителем.

Файлы указанного формата должны размещаться (храниться) на локальных или съемных носителях, отформатированных согласно требованиям операционной системы.

4.3.1.3 Требования к организации выходных данных

Выходные данные должны быть организованы в виде файлов в формате ANT+ FIT. Файлы должны иметь подтип файла № 5 «Workout File», согласно спецификации формата FIT. В выходном файле должно быть корректно установлено поле, описывающее требуемые возможности спортивного компьютера (англ. capabilities). Выходной файл должен распознаваться спортивными компьютерами фирмы Garmin как корректный файл тренировки.

Файлы указанного формата должны размещаться (храниться) на локальных или съемных носителях, отформатированных согласно требованиям операционной системы.

4.3.1.4 Требования к временным характеристикам

Требования к временным характеристикам системы не предъявляются.

4.3.2 Требования к надёжности

4.3.2.1 Требования к обеспечению надежного (устойчивого) функционирования программы

Надежное (устойчивое) функционирование системы должно быть обеспечено выполнением совокупности организационно-технических мероприятий, перечень которых приведен ниже:

- организацией бесперебойного питания технических средств;
- использованием лицензионного программного обеспечения;
- регулярным выполнением рекомендаций Министерства труда и социального развития РФ, изложенных в Постановлении от 23 июля 1998 г. «Об утверждении межотраслевых типовых норм времени на работы по сервисному обслуживанию ПЭВМ и оргтехники и сопровождению программных средств»;
- регулярным выполнением требований ГОСТ 51188-98 «Задача информации. Испытания программных средств на наличие компьютерных вирусов».

4.3.2.2 Время восстановления после отказа

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать 5 минут при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических

средств и переустановки программных средств.

4.3.2.3 Отказы из-за некорректных действий оператора

Отказы системы возможны вследствие некорректных действий оператора (пользователя) при взаимодействии с операционной системой. Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу конечного пользователя без предоставления ему административных привилегий.

4.3.3 Условия эксплуатации

4.3.3.1 Климатические условия эксплуатации

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

4.3.3.2 Требования к видам обслуживания

Программа не требует проведения каких-либо видов обслуживания.

4.3.3.3 Требования к численности и квалификации персонала

Минимальное количество персонала, требуемого для работы программы, должно составлять не менее 2 штатных единиц: системный администратор и конечный пользователь программы — оператор.

Системный администратор должен иметь высшее профильное образование и сертификаты компании-производителя операционной системы. В пере-

чень задач, выполняемых системным администратором, должны входить:

- задача поддержания работоспособности технических средств;
- задачи установки (инсталляции) и поддержания работоспособности системных программных средств — операционной системы;
- задача установки (инсталляции) программы.

Конечный пользователь программы (оператор) должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы.

Персонал должен быть аттестован на II квалификационную группу по электробезопасности (для работы с конторским оборудованием).

4.3.4 Требования к составу и параметрам технических средств

В состав технических средств должен входить IBM-совместимый персональный компьютер (ПЭВМ), включающий в себя:

- 1) процессор Intel Pentium или аналогичный с тактовой частотой не менее 1,8 ГГц;
- 2) оперативную память объемом не менее 1 ГиБ;
- 3) жесткий диск или аналогичное запоминающее устройство с объемом свободной памяти не менее 1 ГиБ.

4.3.5 Требования к информационной и программной совместимости

4.3.5.1 Требования к информационным структурам и методам решения

Требования к информационным структурам (файлов) на входе и выходе, а также к методам решения не предъявляются.

4.3.5.2 Требования к исходным кодам и языкам программирования

Требования к исходным кодам и языкам программирования не предъявляются.

4.3.5.3 Требования к программным средствам, используемым программой

Системные программные средства, используемые программой, должны быть представлены лицензионной локализованной версией операционной системы.

4.3.5.4 Требования к защите информации и программ

Требования к защите информации и программ не предъявляются.

4.3.6 Требования к маркировке и упаковке

Программа поставляется в виде программного изделия на дистрибутивном (внешнем оптическом) носителе (компакт-диске).

4.3.6.1 Требование к маркировке

Программное изделие должно иметь маркировку с обозначением наименования разработчика, наименования продукта, номера версии, даты изготовления.

Маркировка должна быть нанесена на программное изделие в виде наклейки, выполненной полиграфическим или любым другим способом.

4.3.6.2 Требования к упаковке

Упаковка программного изделия должна осуществляться в упаковочную тару предприятия-изготовителя.

4.3.6.3 Условия упаковывания

Упаковка программного изделия должна проводиться в закрытых вентилируемых помещениях при температуре от 15 до 40 °С и относительной влажности не более 80 % при отсутствии агрессивных примесей в окружающей среде.

4.4 Требования к программной документации

4.4.1 Предварительный состав программной документации

Состав программной документации должен включать в себя:

- 1) техническое задание;
- 2) руководство по установке;
- 3) руководство системного администратора;
- 4) руководство оператора.

4.5 Технико-экономические показатели

Ориентировочная экономическая эффективность не рассчитывается.

Предполагаемое число использования программы в год — 365 сеансов работы на одном рабочем месте.

4.5.1 Экономические преимущества разработки

Экономические преимущества разработки не рассчитываются.

4.6 Порядок контроля и приемки

4.6.1 Виды испытаний

Контроль и приёмка разработки осуществляются на основе испытаний контрольно-отладочных примеров. При этом проверяется выполнение всех функций программы.

5 Модель предметной области

Центральными сущностями предметной области являются тренировки, этапы или шаги тренировок, а также условия завершения и цели в рамках отдельного этапа. Главные сущности представлены на рисунке 4 в виде диаграммы классов.

Тренировка в модели представляется классом `Workout`, имеющим атрибуты `name` (название тренировки), `sport` (вид спорта) и `steps` (последовательность шагов). Согласно спецификации формата FIT, название и вид спорта для тренировки могут быть не заданы. Поэтому эти атрибуты имеют кратность `[0..1]`. Количество шагов в тренировке должно быть строго больше нуля, отсюда кратность `[1..*]`. Вид спорта представлен как перечислимый тип `Sport` — класс со стереотипом `enumeration`.

Шаг тренировки может быть как отдельным шагом, так и повторением последовательности шагов. В модели это отражено следующим образом. Собственно тип `Step` является интерфейсом, который используется для агрегации в тренировке. Этот интерфейс реализуют два класса: класс отдельного шага `SingleStep` и класс повторения `RepeatStep`.

Отдельный шаг имеет атрибуты `name` (название шага), `duration` (условие окончания этапа, сколько этап длится), `target` (цель этапа) и `intensity`. Все атрибуты необязательны и имеют кратность `[0..1]`. Интенсивность представляется перечислимым типом `Intensity`. Условие окончания и цель этапа имеют типы `Condition` и `Target` и описаны в отдельных пакетах: соответственно `conditions` и `targets`. Диаграммы этих пакетов представлены на отдельных рисунках, чтобы не занимать место на диаграмме основных сущностей.

Шаг не включает в себя номер, имеющийся в спецификации FIT, потому что информация о порядке следования шагов неявно содержится в самой последовательности. Включение поля `message_index` создало бы возможность для противоречия в модели: шаг с некоторым индексом `N` в последовательности

шагов располагается по другому смещению М. Отсутствие явного поля для индекса шага исключает такую возможность.

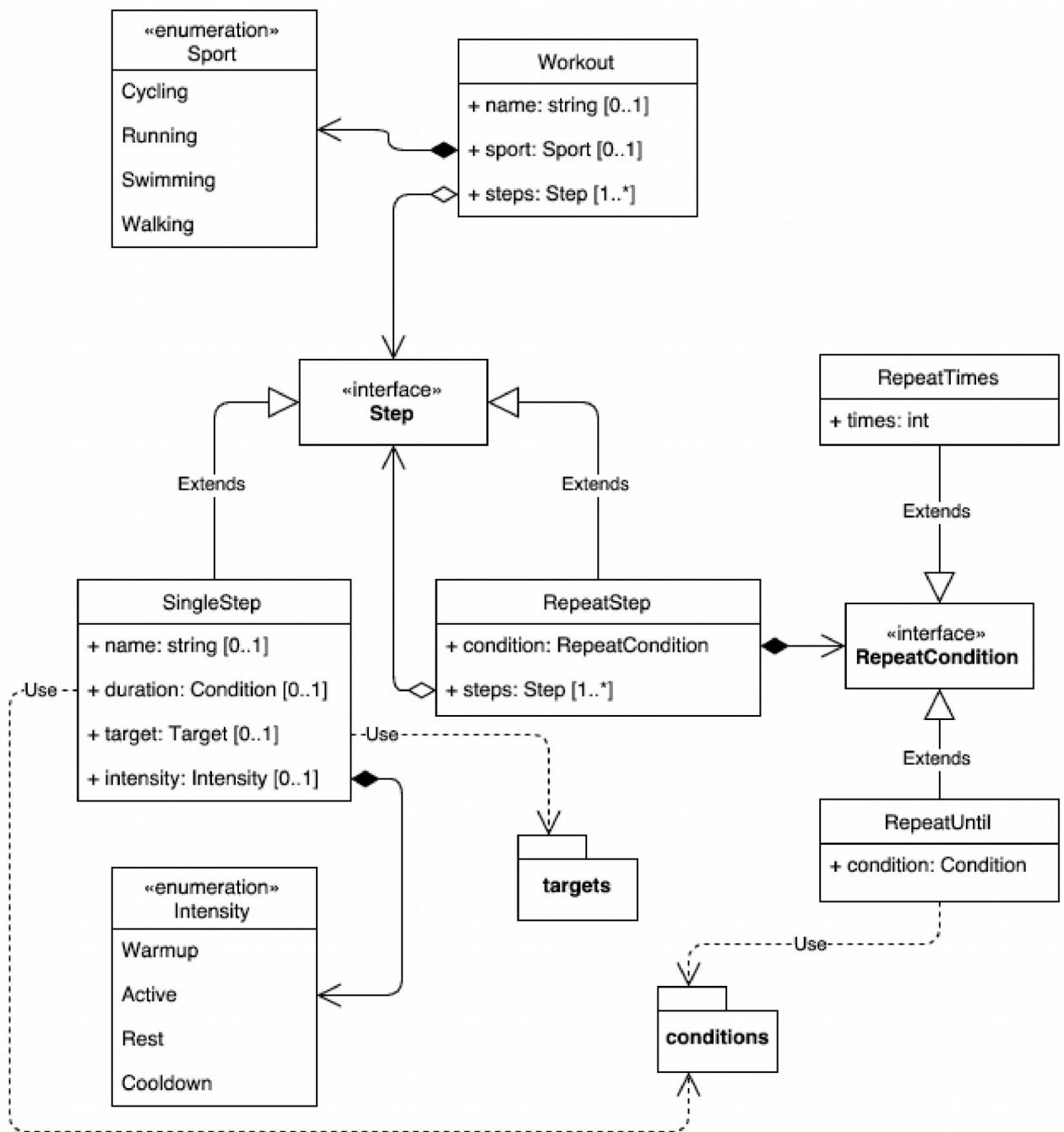


Рисунок 4 — Диаграмма классов основных сущностей

Что касается повторений — класса RepeatStep, — то в модели они представлены иначе, чем в формате FIT, который подразумевает, что повторение со-

держит номер шага. Это допускало бы противоречивые состояния модели, когда повторение по номеру ссылается на шаг, которого нет в тренировке.

Также при изменении порядка шагов требовалось бы обновлять номера в повторениях. Поэтому повторение в модели определяется рекурсивно: класс RepeatStep реализует интерфейс Step, непосредственно включает последовательность шагов (атрибут steps) и управляет временем их жизни.

Так же как и в самой тренировке, количество шагов в повторении должно быть строго больше нуля.

Повторение осуществляется пока выполняется условие повторения: либо повторение заданное число раз, либо пока не выполнится заданное условие. Условие повторения (поле condition) ассоциируется с интерфейсом RepeatCondition. Этот интерфейс реализуют два класса: RepeatTimes (повторение заданное число раз) и RepeatUntil (повторение до тех пор, пока не выполнится заданное условие). Условие в RepeatUntil имеет тот же тип, что и условие в отдельном шаге, класс RepeatUntil зависит от пакета conditions.

Условие окончания этапа тренировки представляется интерфейсом Condition, изображённым на диаграмме классов пакета conditions на рисунке 5.

Этот интерфейс реализуют классы, обеспечивающие конкретные условия: DistanceCondition (этап выполняется пока не пройдено заданное расстояние), TimeCondition (этап длится заданное время), CaloriesCondition} (этап длится пока не израсходовано заданное число килокалорий), HeartRateCondition (этап выполняется пока ЧСС меньше или больше заданной) и PowerCondition (этап выполняется пока мощность больше или меньше заданной).

Для отношения «больше» или «меньше» классы HeartRateCondition и PowerCondition включают атрибуты rel, ассоциированные с перечислимым типом Relation. Кроме того, данные классы должны включать ЧСС и мощность либо как абсолютные значения, либо как проценты от максимальной ЧСС и функциональной мощности. Для этого в модель введены интерфейсы HeartRate и Power. Реализуются эти интерфейсы классами AbsoluteHeartRate,

PercentHeartRate, AbsolutePower и PercentPower.

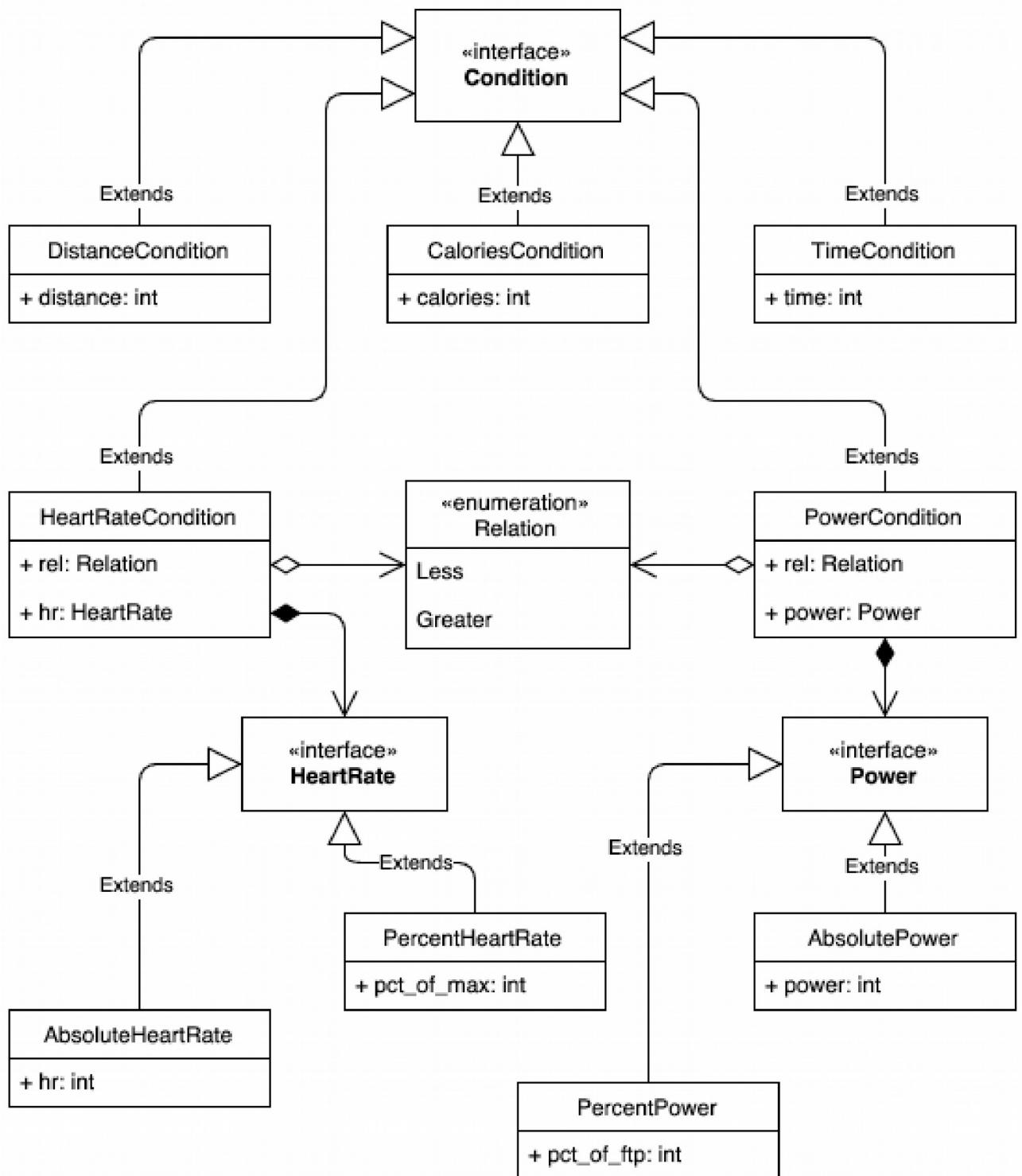


Рисунок 5 — Диаграмма классов пакета «conditions»

Классы пакета targets представлены на рисунке 6. Цель этапа тренировки

в модели представляется интерфейсом Target. Диапазон для целевого показателя может быть задан либо явно (класс RangeTarget, реализующий интерфейс Target), либо как номер зоны, для которой в настройках устройства задан диапазон (класс ZoneTarget, реализует Target). Класс ZoneTarget содержит целочисленный атрибут zone — номер зоны. От данного класса унаследованы классы для конкретных целевых показателей.

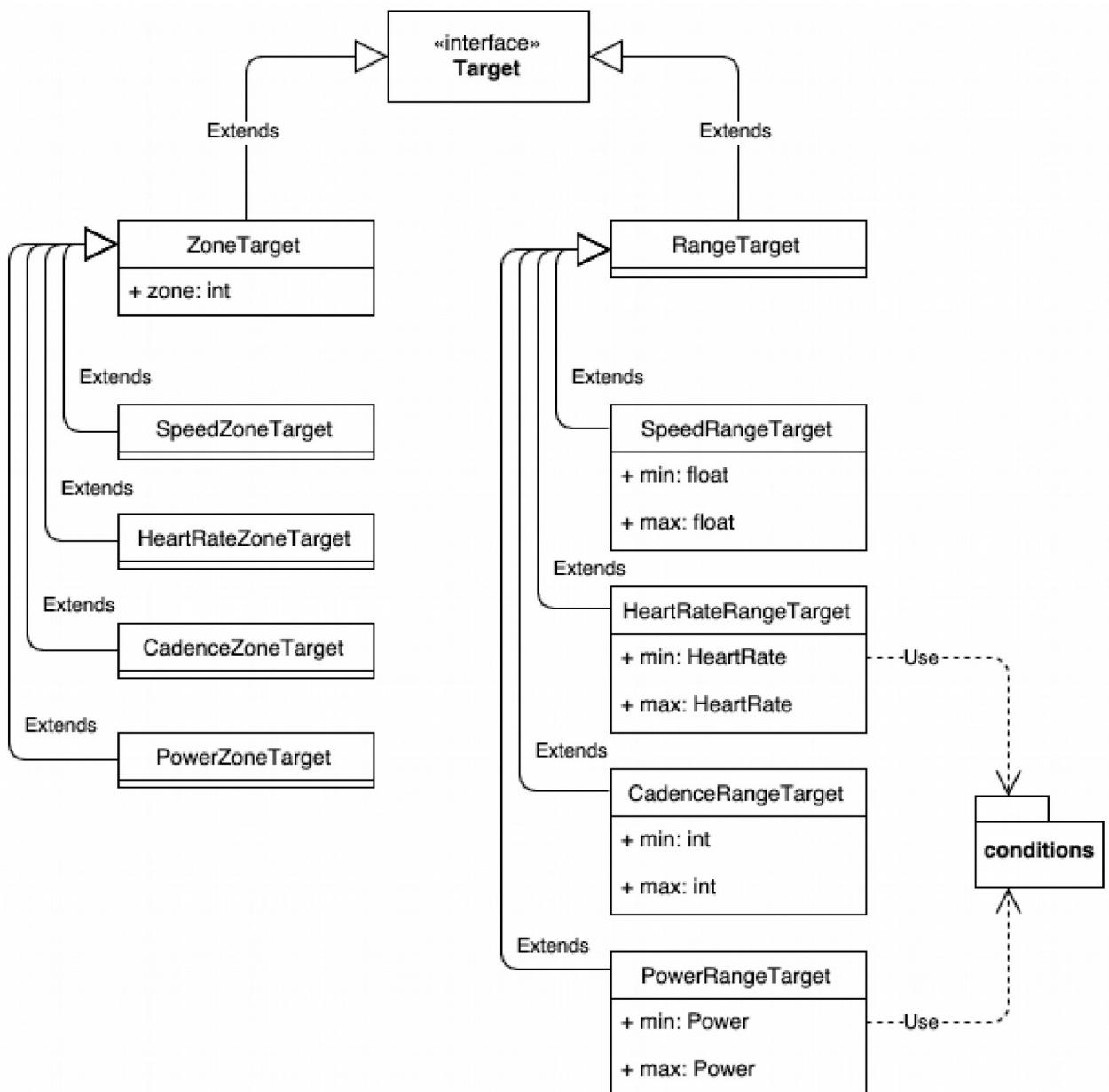


Рисунок 6 — Диаграмма классов пакета «targets»

От класса RangeTarget унаследованы классы диапазонов целевых показателей для скорости, ЧСС, каденса и мощности. Причём ЧСС и мощность могут задаваться либо как абсолютные значения, либо как проценты; классы HeartRateRangeTarget и PowerRangeTarget зависят от интерфейсов HeartRate и Power, описанных в пакете conditions выше.

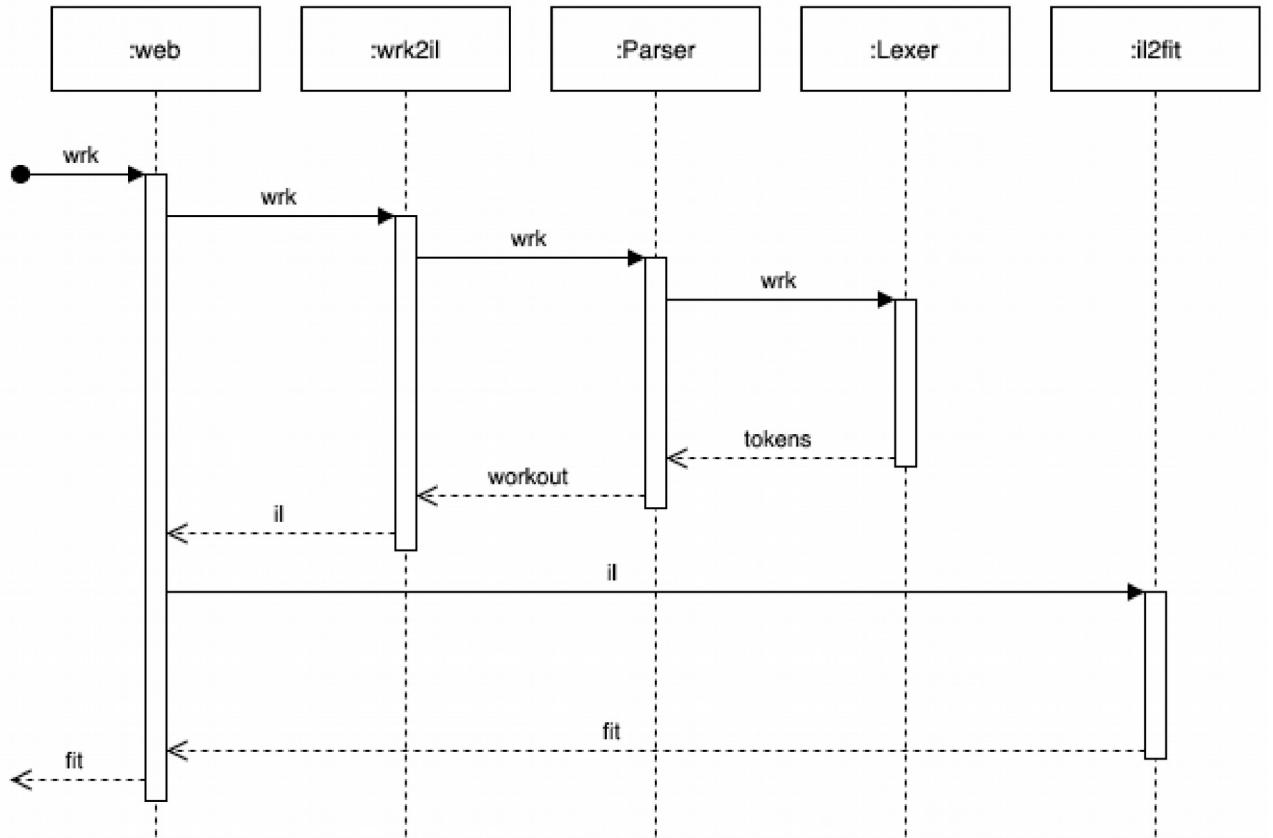


Рисунок 7 — Взаимосвязь компонентов при обработке запроса

6 Выбор метода и инструментов разработки

6.1 Язык описания тренировок

В соответствии с требованиями, необходимо разработать формальный язык предметной области (англ. Domain Specific Language) описания тренировок для спортивных компьютеров и обеспечить его трансляцию в формат ANT+ FIT. В книге [8] рассматривается применение так называемого языкоориентированного программирования и создание языков предметной области.

Язык разрабатывался на основе модели предметной области, выделенной в разделе «Модель предметной области». Из языка исключались лишние ключевые слова, устранились возможности для создания противоречивых состояний, окончательный вариант языка представлен далее в виде синтаксических диаграмм.

В языке в различных контекстах используются натуральные числа, действительные числа и текстовые строки — базовые элементы языка. Соответствующие синтаксические диаграммы представлены на рисунках 8, 9 и 10.

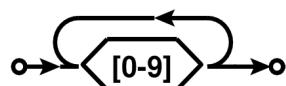


Рисунок 8 — Синтаксическая диаграмма «нат. число»

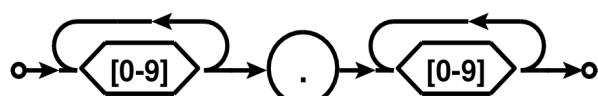


Рисунок 9 — Синтаксическая диаграмма «действ. число»

Натуральное число представляется как последовательность цифр. Действительное число содержит целую и дробную части, разделённые символом «точка». И та, и другая часть должна содержать по крайней мере одну цифру.

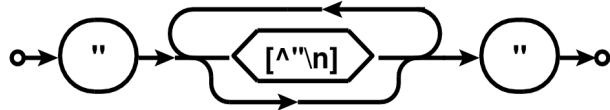


Рисунок 10 — Синтаксическая диаграмма «строка»

Текстовая строка — это последовательность произвольных символов (возможно, пустая), заключённая в кавычки. При этом среди символов не должно встречаться символа кавычек и символа перехода на новую строку.

Центральным понятием предметной области и стартовым нетерминальным символом грамматики языка является тренировка (рисунок 11). Тренировка имеет имя — текстовая строка с последующим символом «двоеточие», — далее следует ключевое слово, обозначающее вид спорта тренировки, а затем список шагов тренировки.

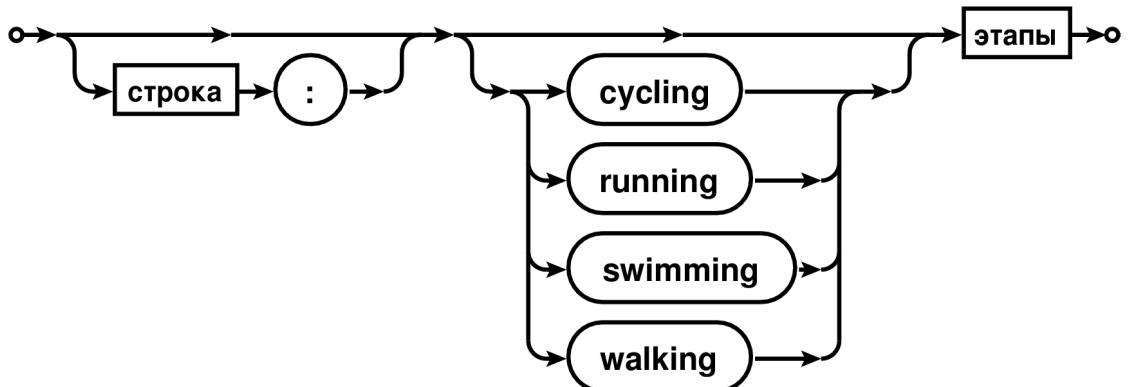


Рисунок 11 — Синтаксическая диаграмма «тренировка»

Список этапов представляется как заключённая между символами открывающей и закрывающей квадратных скобок последовательность шагов, разделённых символом «точка с запятой» (рисунок 12). Последовательность обязательно содержит по крайней мере один элемент.

Каждый шаг — это либо обычный отдельный шаг тренировки, либо описание повторения непустой подпоследовательности шагов.

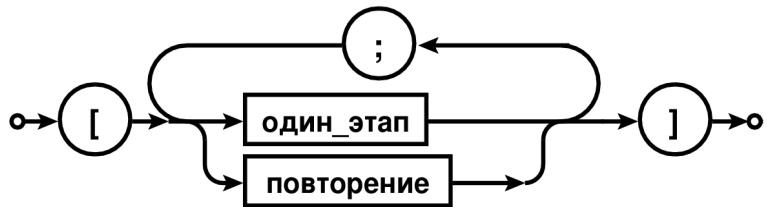
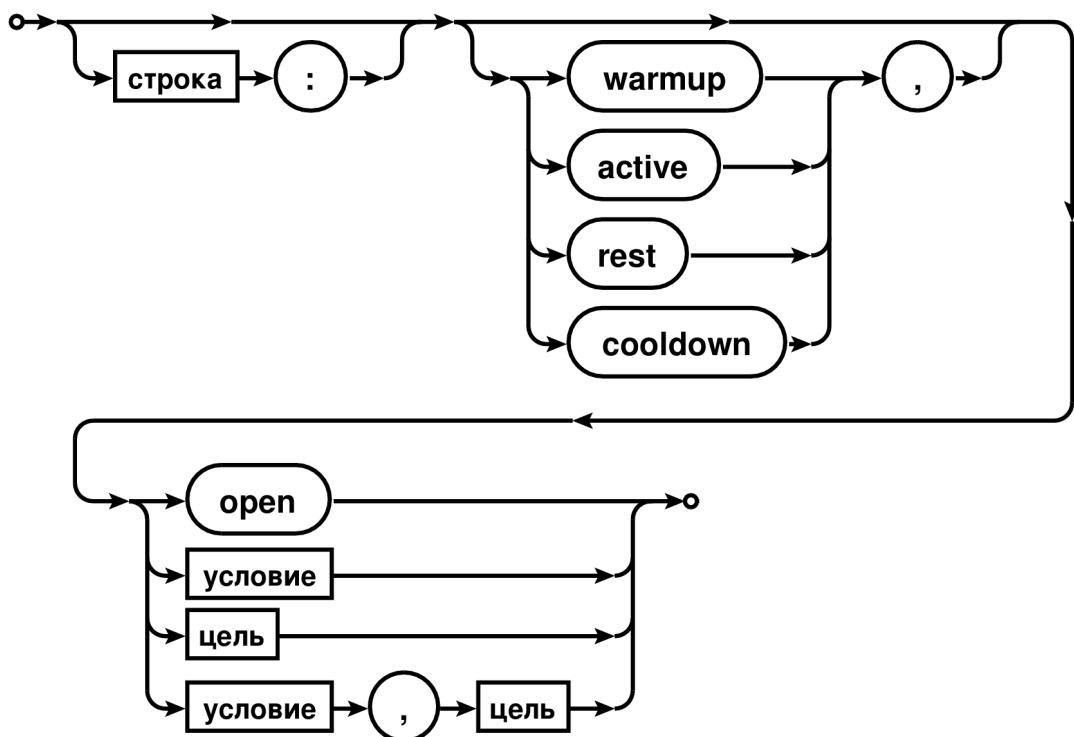


Рисунок 12 — Синтаксическая диаграмма «этапы»

Описание отдельного шага состоит из трёх частей (диаграмма на рисунке 13). Сначала записывается название шага тренировки — строка с последующим символом «двоеточие».



чевое слово «*open*» — шаг завершается только вручную и не имеет целевого показателя. Может присутствовать либо только условие, либо только цель, либо и то, и другое. В последнем случае условие и цель разделяются запятой.

Повторения последовательности шагов записывается как условие повторения между открывающей и закрывающей круглыми скобками, после которых следует список шагов для повторения (рисунок 14). Таким образом, определения списка шагов и повторения взаиморекурсивны.

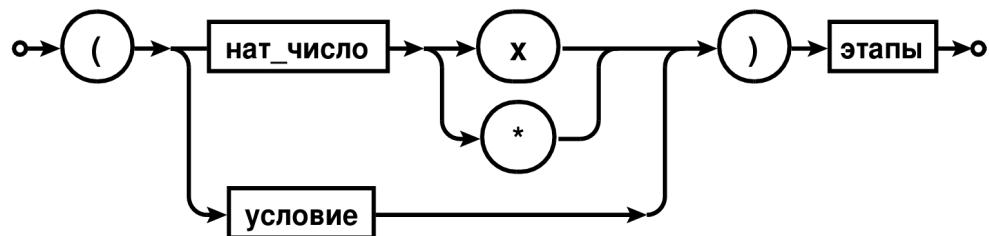


Рисунок 14 — Синтаксическая диаграмма «повторение»

Условие повторения либо указывает какое количество раз должны быть повторены шаги из списка — натуральное число за которым следует либо латинская буква «*x*», либо символ «звёздочка» (умножение), — либо совпадает с определением условия окончания отдельного шага из определения выше. В этом случае шаги из списка должны повторяться пока не выполнится данное условие.

Синтаксическая диаграмма условия окончания шага тренировки (также используется в условиях повторения) показана на рисунке 15. Определения схожи для разных условий: сначала записывается ключевое слово языка, идентифицирующее условие («*time*», «*distance*» и т. д.), затем указывается значение. В случае частоты сердечных сокращений и мощности перед значением также записывается символ «меньше» или «больше».

Различные целевые показатели шага тренировки (синтаксическая диаграмма на рисунке 16) имеют схожий вид: сначала ключевое слово языка, идентифицирующее целевой показатель («*speed*», «*hr*» и т. д.), а затем либо клю-

чевое слово «zone» и номер зоны, либо два значения, разделённых дефисом — нижняя и верхняя границы диапазона целевого показателя.

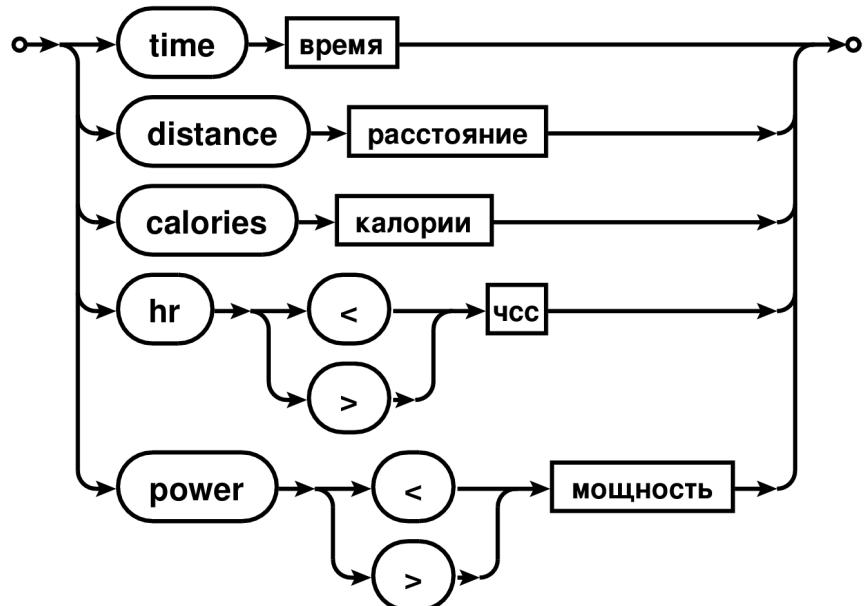


Рисунок 15 — Синтаксическая диаграмма «условие»

Значения различных показателей (как для целей, так и для условий) все описываются схожим образом: натуральное число и ключевое слово, идентифицирующее единицы измерения. Синтаксические диаграммы для значений времени, расстояния, каденса, килокалорий, ЧСС, мощности и скорости представлены на рисунках 17, 18, 19, 20, 21, 22 и 23.

Единицы измерения указывать необязательно, для каждого типа значений есть единица измерения по умолчанию. Это секунды для времени, метры для расстояний, обороты в минуту для каденса, килокалории, километры в час для скорости, удары в минуту для ЧСС и ватты для мощности. Допускается задавать значения ЧСС и мощности в процентах. В этом случае после числа следует знак процента. Скорость допускается указывать как натуральное или как действительное число. Для представления времени допускается использовать альтернативный формат — часы, минуты и секунды, разделённые двоеточием. Если количество часов равно нулю, его можно не указывать.

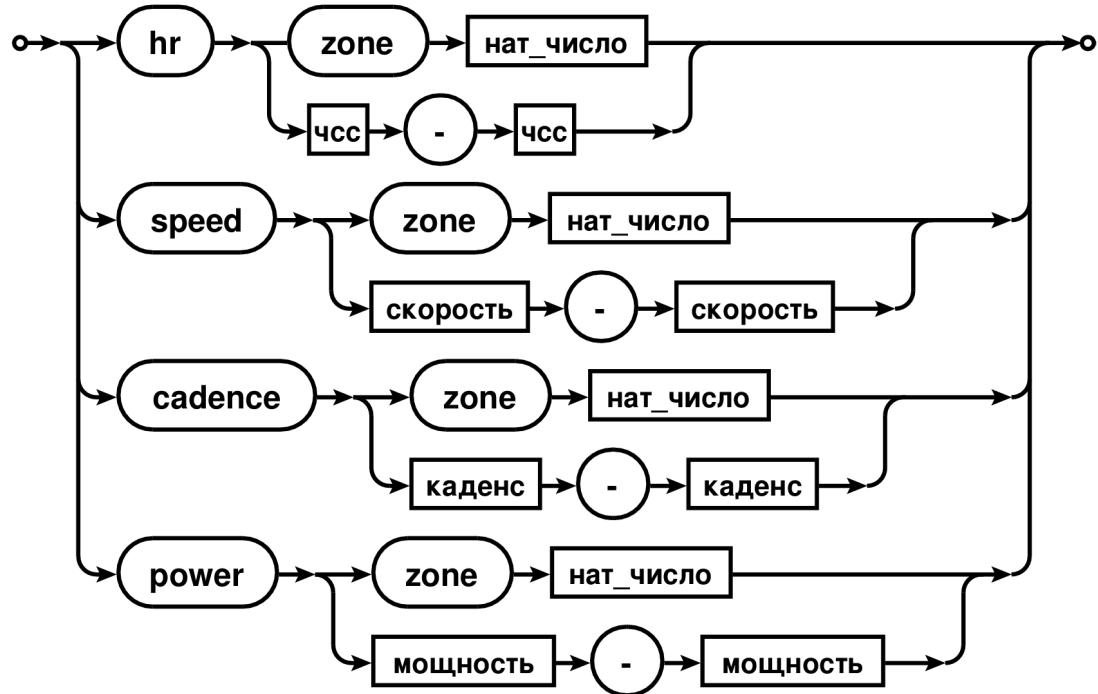


Рисунок 16 — Синтаксическая диаграмма «цель»

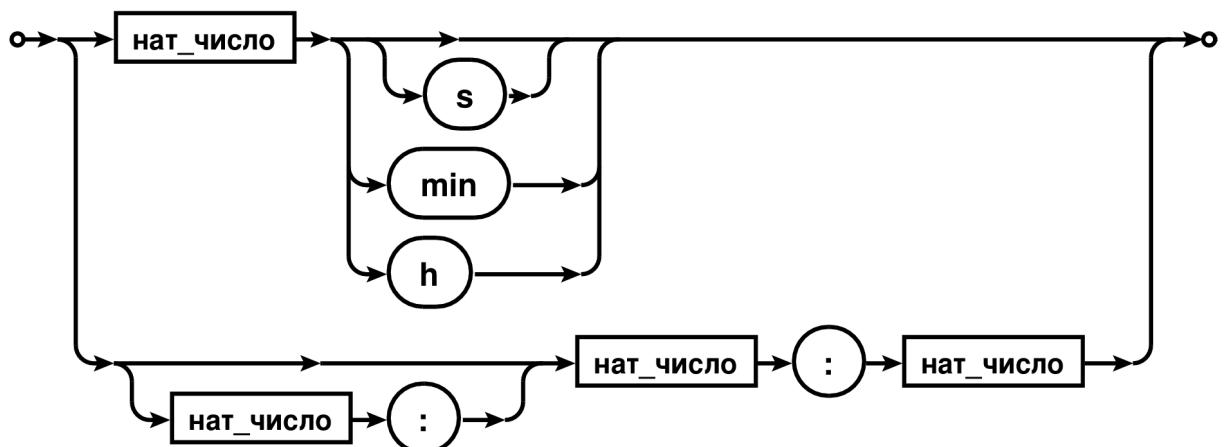


Рисунок 17 — Синтаксическая диаграмма «время»

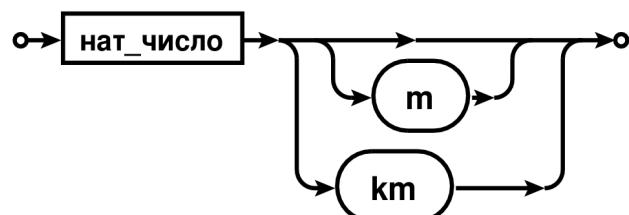


Рисунок 18 — Синтаксическая диаграмма «расстояние»

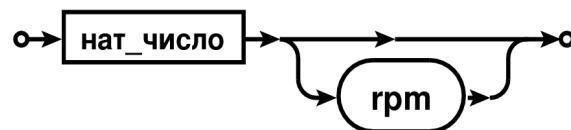


Рисунок 19 — Синтаксическая диаграмма «каденс»

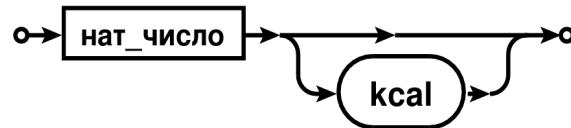


Рисунок 20 — Синтаксическая диаграмма «калории»

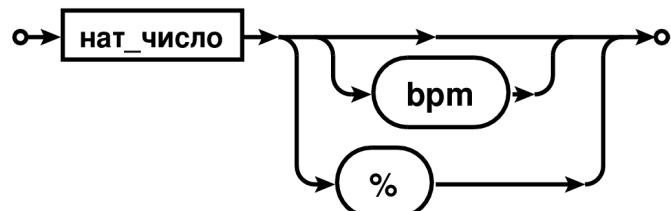


Рисунок 21 — Синтаксическая диаграмма «чсс»

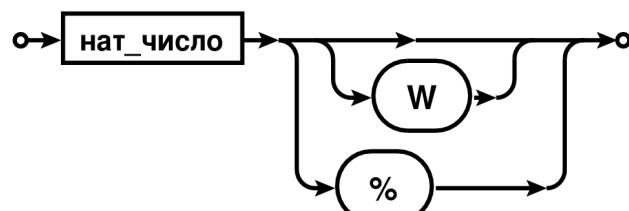


Рисунок 22 — Синтаксическая диаграмма «мощность»

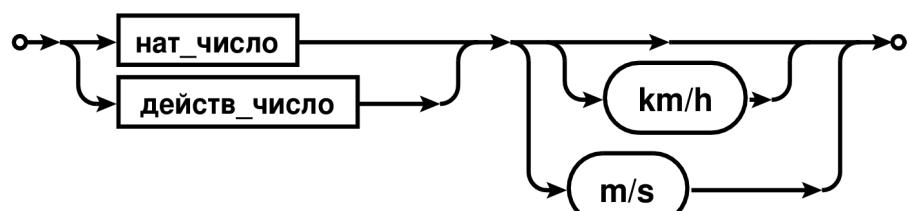


Рисунок 23 — Синтаксическая диаграмма «скорость»

Чтобы ссылаться на данный язык далее по тексту, примем для него название WRK (сокращение от «Workout File» в терминах спецификации формата

FIT) в качестве рабочего. Далее по тексту данный язык называется WRK.

В [3] приведены несколько примеров низкоуровневой структуры тренировки в формате FIT. Для демонстрации языка описания тренировок, пример из [3] представлен ниже в виде текста на языке WRK.

```
[ "A": warmup, time 1 min, hr zone 2;  
  (3x) [ "B1": active, distance 500 m, power zone 5;  
         "B2": active, distance 500 m, power zone 3 ];  
  "C": cooldown, hr < 125, power zone 1 ]
```

Рисунок 24 — Пример тренировки на языке WRK

В листинге на рисунке 24 описывается тренировка, для которой не заданы название и вид спорта. Тренировка имеет четыре шага. Шаг с названием «A» является разминкой, выполняется одну минуту, спортивный компьютер будет предписывать поддерживать частоту сердечных сокращений в зоне 2. Шаги «B1» и «B2» выполняются пока в рамках каждого из них не будет преодолено 500 метров. Предписывается поддерживать мощность в зонах 5 и 3. Последовательность шагов «B1» и «B2» будет повторяться три раза. Шаг «C» является «заминкой», выполняется пока ЧСС не упадёт ниже 125 ударов в минуту, с поддержанием мощности в зоне 1. После этого тренировка считается выполненной.

Для реализации транслятора с языка предметной области WRK будет использоваться язык программирования OCaml. Этот язык хотя и поддерживает различные методологии программирования (включая объектно-ориентированную), но используется обычно прежде всего как язык функционального программирования. В книге «Concepts, Techniques, and Models of Computer programming» авторы подробнейшим образом рассматривают и сравнивают различные методологии программирования и показывается, что функциональное программирование — наиболее простая из них. Преимущества функционального программирования на конкретных примерах рассматривается также в

[9] и [10, с. 44]. Если задача позволяет это сделать, использовать следует именно функциональный подход.

Язык OCaml является статически типизированным с автоматическим выводом типов. Строгая система типов языка, кроме того, предоставляет очень много возможностей для описания предметной области и позволяет описать модель предметной области таким образом, что некорректные состояния становятся непредставимыми в рамках этой модели. Это гарантируется системой типов. Эта особенность является очень важной с точки зрения корректности программ, демонстрируется в статье [11] и неоднократно демонстрируется тем же автором в других работах (например, в [12]).

Задача трансляции языка WRK очень хорошо подходит для функционального программирования и для языка OCaml в частности. Язык компилируется в машинный код, полученные программы не сильно уступают в производительности программам на таких низкоуровневых языках как C и C++. При этом в OCaml реализована статическая проверка типов и их вывод, сопоставление с образцом и многие другие высокуюровневые возможности.

Язык OCaml поставляется с инструментом для создания лексических анализаторов — «ocamllex». Также с языком поставляется инструмент для создания синтаксических анализаторов «ocamlyacc», но он считается устаревшим и вместо него используется полностью совместимый сторонний инструмент «menhir». Генератор синтаксических анализаторов позволяет из описания грамматики языка автоматически получить программный код для разбора языка во внутреннее представление, модель предметной области.

Так как написание транслятора с относительно сложного языка WRK не является тривиальной задачей, целесообразно использовать подобные инструменты. Преимущество автоматически сгенерированных анализаторов перед реализованными вручную показаны, например, в [13].

В последние несколько лет интерес к языку OCaml возрастает, создаётся большое количество библиотек и инструментов. Среди них пакетный менеджер

«орам», который позволяет достаточно легко получить окружение для разработки на OCaml со всеми необходимыми библиотеками и инструментами.

6.2 Формирование FIT файлов

Компания-разработчик формата FIT бесплатно распространяет комплект разработки ПО (Software Development Kit) для работы с файлами в формате FIT. Целесообразно использовать данный SDK, чтобы не работать вручную с низкоуровневыми двоичными структурами данных, не рассчитывать контрольные суммы и т. д. Очевидно, это ускорит разработку и позволит избежать большого количества ошибок при реализации. Работа с SDK описывается в [14].

SDK доступен для языков C#, C++, С и Java, и, разумеется, недоступен для языка OCaml. Поэтому для записи файлов в формате FIT потребуется реализовать отдельный компонент на одном из этих языков и обеспечить его связь с транслятором на языке OCaml.

В качестве языка разработки компонента был выбран язык C++. При этом используется обновлённый стандарт языка, который поддерживает лямбда выражения, вывод типов. Обзор новых возможностей и в особенности лямбда выражений приведён в [15, с. 221].

Для обеспечения связи компонентов на языках OCaml и C++ используется следующее решение.

Компонент wrk2il, написанный на языке OCaml, транслирует язык описания тренировок WRK в очень простой промежуточный язык, разбор которого можно было бы осуществить стандартными средствами языка C++, без написания сложных трансляторов. Компонент il2fit, написанный на C++, читает описание тренировки на простом промежуточном языке и формирует FIT файл. Ввод и вывод осуществляются через стандартные потоки ввода и вывода, компоненты wrk2il и il2fit объединяются в конвейер wrk2fit. Таким образом осуществляется трансляция из языка WRK в формат файлов FIT.

Промежуточный язык основан на строках и представляет собой последовательность пар «ключ-значение». При этом значение передаётся на следующей строке после строки с ключом. Сообщения на таком языке достаточно легко создавать и достаточно легко читать и конвертировать в формат FIT.

6.3 Графический интерфейс пользователя

Компоненты wrk2il и il2fit используют интерфейс командной строки для взаимодействия с пользователем, но в операционной системе Windows интерпретатор команд используется не так часто, и большинство пользователей предпочитают графический интерфейс пользователя. Графический интерфейс, согласно требованиям к программному обеспечению, не является критическим, но желательно реализовать такую возможность.

Вероятно, самым простым графическим интерфейсом для данной задачи, является простое окно, в которое пользователь должен перетаскивать файлы с текстовым описанием тренировки. В этом случае можно использовать существующие компоненты wrk2il и il2fit.

Для реализации простых и графических интерфейсов средней сложности очень хорошо подходит язык программирования Tcl/Tk. Язык Tcl является интерпретируемым, динамическим языком. Tk является встроенным предметно-ориентированным языком для описания графических интерфейсов.

Tcl/Tk работает в операционных системах Windows, Mac OS и в большинстве существующих Unix систем.

Компонент Tk используется как встроенное средство для создания графических интерфейсов во многих других языках, помимо Tcl. В [TODO] достаточно просто, на конкретных примерах, показывается создание графических приложений с использованием Tk на языках Perl, Python, Ruby, Tcl.

Для реализации возможности перетаскивать и отпускать файлы (англ. Drag and Drop) потребуется использовать компонент «TkDnD», который не

входит в состав дистрибутива Tcl/Tk, а разрабатывается отдельно.

6.4 Веб-служба

Веб-сервис должен обеспечивать создание FIT файла при получении по HTTP запроса, с описанием тренировки на языке WRK.

Язык описания тренировок изначально проектировался таким образом, чтобы не содержать значимых пробелов. То есть, из такого текста можно удалить большинство пробелов и переводов каретки на новую строку, и полученный текст останется синтаксически правильным.

Это позволяет закодировать описание тренировки прямо в адресе ресурса. веб-сервис реализован таким образом, чтобы отвечать на HTTP запросы с методом GET и описанием тренировки в URL. Например, создание тренировки из листинга на рисунке 50 можно инициировать отправкой GET запроса на следующий URL (если сервис запущен для адреса localhost и для стандартного номера порта~80): «`http://localhost/workouts/[{"A": warmup, time 60, hr zone 2; (3x) "B1": active, distance 500, power zone 5; "B2": active, distance 500, power zone 3}; "C": cooldown, hr< 125, power zone 1]`».

Описание тренировки одновременно является идентификатором ресурса FIT файла на этом сервере. Это естественным образом позволяет кэшировать результат работы сервера на стороне клиента и в промежуточных прокси-серверах. Пользователь может сохранять закладки на подобные ресурсы и в любой момент запросить создание FIT файла.

При проектировании веб-сервиса учитываются принципы REST (Representational State Transfer). Сегодня REST стала преобладающей моделью проектирования веб-сервисов. Согласно статье [17], конкретная реализация веб-сервиса должна в том числе следовать следующим принципам:

- явное использование HTTP-методов;
- несохранение состояния;

- предоставление URI, аналогичных структуре каталогов.

Конкретные рекомендации по разработке Web-сервисов в соответствии с REST (описание HTTP-методов и кодов, рекомендации по именованию ресурсов и т. д.) даются в [18].

При обработке запроса сервер транслирует описание с помощью вызова описанного выше компонента wrk2fit и отвечает клиенту готовым FIT файлом. Если описание тренировки было некорректным, сервер отвечает ошибкой с HTTP кодом 400.

Обработка подобных запросов является ключевой особенностью веб-сервиса. Остальные возможности (редактирование текстового описания тренировки в окне Интернет обозревателя, графический редактор тренировок и т. п.) должны быть реализованы в статичных файлах, выполняться на стороне клиента, и формировать ссылки на ресурсы, подобные приведённому выше.

Для реализации веб-сервиса был выбран язык программирования Erlang. Данный язык очень широко используется для реализации различных веб-сервисов и считается очень хорошо масштабируемым. Для реализации непосредственно HTTP сервера используется библиотека «cowboy». Данная библиотека позволяет реализовывать очень сложные REST интерфейсы, управлять обработкой запросов на каждом этапе. Обзор возможностей библиотеки представлен её автором в [19].

В текущей версии программного обеспечения такой выбор, вероятно, избыточен, но веб-сервис — это та часть программного обеспечения, в которой в будущих версиях планируется добавление наибольшего числа возможностей. Некоторые из них потребуют хранения информации о пользователях, обработки более сложных запросов и т. д. То есть в будущем возможности языка Erlang и библиотеки «cowboy» должны будут использоваться более полно.

Взаимосвязь компонентов системы показана на рисунке 7 в виде диаграммы последовательностей при обработке одного запроса.

7 Реализация разработки

7.1 Реализация программных модулей

7.1.1 Компонент wrk2il

Этот компонент программного обеспечения написан на языке программирования OCaml и предназначен для трансляции языка описания тренировок в промежуточный язык.

Компонент состоит из шести модулей, часть из которых являются вспомогательными, не относится к предметной области и здесь не рассматриваются (например, модуль Non_empty_list для представления непустых списков).

Центральным модулем компонента является модуль Workout (тренировка). Модули на языке OCaml состоят из двух частей: интерфейса модуля (файл с расширением .mli), который содержит описания типов, подмодулей, сигнатуры функций; реализации модуля (файл .ml) с деталями реализации. Исходный код интерфейса модуля представлен в приложении А.

Как упоминалось ранее, система типов языка OCaml позволяет очень точно описать предметную область, отразить связи между сущностями и иногда их кратность. Такой подход часто демонстрируется в работах по языку OCaml (например, в [11], [12]) и по родственному ему языку F# в [20].

Например, в данном случае в строке 129 определяется тип данных для представления тренировок, который очень точно соотносится с описанием предметной области: имя и вид спорта могут отсутствовать (используется параметризованный тип option), список шагов тренировки обязательно имеет хотя бы один элемент.

В строке 116, в отдельном подмодуле, определяется тип данных для представления шагов тренировки. Это определение хорошо демонстрирует возмож-

ности системы типов. Шаг тренировки — это алгебраический тип данных, тип-сумма (либо отдельный шаг, либо повторение с условием списка шагов). Определения шага и повторения списка шагов взаимно рекурсивны. Элементы отдельного шага optionalны, повторение содержит обязательное условие и всегда непустой список шагов для повторения.

Все эти ограничения описаны системой типов и гарантируются компилятором. Таким образом, если грамотно спроектировать типы, можно добиться того, что некорректные состояния модели будут непредставимы в программе, некорректная программа не будет компилироваться.

В этом модуле часто используется такая возможность языка, как закрытые псевдонимы типов. Например, в строке 33 зона ЧСС определена как закрытый псевдоним типа целое число. В результате со значениями этого типа можно работать как со значениями типа целое число, но создать новые значения этого типа можно только внутри данного модуля. Для этого служит функция-конструктор `zone_of_int`, объявленная на строке 37, которая проверяет, что значение находится в требуемом диапазоне. Иначе возбуждается исключение.

Таким образом, благодаря системе типов языка OCaml, если где-то в программе существует значение типа тренировка, то оно гарантированно имеет корректную структуру и содержит корректные данные. Некорректные состояния значений этого типа создать невозможно.

В модуле `Parser` содержится описание грамматики языка WRK. Исходный код модуля представлен в приложении А.

В строках 1–44 описываются символы грамматики. Стартовым символом является символ `workout`. После описания символов, начиная со строки 48, следует описание правил грамматики.

Данный файл транслируется генератором компиляторов Menhir в модули на языке OCaml для разбора языка описания тренировок во внутреннее представление, описанное в модуле `Workout`. Часто используются параметризованные правила грамматики (описаны в [21, с. 12]), что позволяет повторно использо-

зователь одно и то же по структуре правило в разных контекстах.

В модуле `Lexer` содержится описание лексического анализатора языка описания тренировок. Задача лексического анализатора — разбиение входного потока символов на символы грамматики из модуля `Parser`. Исходный код модуля представлен в приложении А.

Содержимое модуля — это набор правил сопоставления набора строк из входного потока символам грамматики. В каждом правиле слева указано регулярное выражение для потока входных символов, а справа соответствующий ему символ грамматики.

Данный файл транслируется генератором лексических анализаторов `ocamllex`, который поставляется вместе с компилятором языка OCaml. Результатом трансляции являются модули на языке OCaml.

Модуль `Repr` (от англ. *representation*) содержит функции для форматирования внутреннего представления тренировки в виде текста на промежуточном языке. В приложении А представлен исходный код модуля.

Функции этого модуля однотипны, форматируют тот или иной компонент тренировки в текст на промежуточном языке.

В этом модуле широко используется такая возможность языка OCaml, как сопоставление с образцом. Каждая функция форматирования фактически является набором правил для оператора `match` языка OCaml. В каждом правиле слева указывается та или иная составная часть тренировки, а справа — текст на промежуточном языке, который ей соответствует.

Модуль `Wrk2il` содержит точку входа в программу и объединяет остальные модули в единую программу. Исходный код данного модуля представлен в приложении А.

Модуль обесцвечивает разбор параметров командной строки, трансляцию описания тренировки во внутреннее представление (тем самым проверяя синтаксическую и логическую корректность входных данных) и форматирование внутреннего представления в текст на промежуточном языке. Если в ходе разбо-

ра входных данных возникла ошибка, выводится сообщение об ошибке с указанием номера строки входного файла.

7.1.2 Компонент il2fit

Данный компонент предназначен для преобразования описания тренировки на промежуточном языке в двоичный формат FIT. Компонент реализован на языке программирования C++ и использует SDK от фирмы-разработчика формата FIT для создания двоичных сообщений.

Исходный код компонента содержится в файле il2fit.cpp и представлен в листинге в приложении А.

Модуль состоит из нескольких однотипных функций, которые построчно вычитывают пары ключ-значение из входного потока данных и устанавливают соответствующие поля в структурах данных FIT. По завершении той или иной секции входных данных в выходной поток выдаётся одно из сообщение в формате FIT. Каждая такая функцию обрабатывает свой тип сообщений.

Например, со строки 230 начинается определение функции, создающей в выходном FIT файле секцию идентификатора файла. Сначала такая функция создаёт объект сообщения FIT, устанавливает в нём некоторые значения по умолчанию. Затем читает строки на промежуточном языке пока не встретит маркер конца секции. После этого возвращает созданный объект с сообщением FIT. В модуле содержатся подобные функции для создания сообщений формата FIT типов FileCreatorMesg, fileIdMesg, WorkoutMesg, WorkoutStepMesg.

Главная функция программы содержит основной цикл программы. Определение данной функции начинается на строке 521.

Функция читает из входного потока идентификаторы создаваемых секций FIT файла и вызывает соответствующую данной секции функцию. После создания сообщения секции оно записывается в выходной файл и управление передаётся в основной цикл программы. Основной цикл прекращается когда закон-

чится входной поток строк, либо когда встретится специальный строковый маркер «EOF».

7.1.3 Компонент wrk2fit

Для выполнения компонентов wrk2il и il2fit как единого целого, реализован сценарий интерпретатора команд, который объединяет данные компоненты в единую программу. Сценарий реализован в двух вариантах — для интерпретатора команд `sh`, используемого в Linux и других POSIX системах, и для интерпретатора `cmd` из операционной системы Windows. Исходный код сценария для интерпретатора `sh` представлен в приложении А.

7.1.4 Компонент web

В данном компоненте реализована веб-служба, позволяющая удалённо преобразовать описание тренировки в двоичный файл FIT. Данная служба может использоваться как напрямую, для создания FIT файлов пользователями, так и сторонними проектами, реализующими, например, графический интерфейс редактирования тренировок.

Компонент реализован на платформе Erlang/OTP. Часть модулей обеспечивает требуемую структуру Erlang/OTP приложения, является типовой для таких приложений и здесь не рассматривается. Описание структуры Erlang/OTP приложения приводится в [22, с. 285] и в [23].

Модуль `wrked_port` содержит функции для вызова компонентов `wrk2il` и `il2fit`. Для вызова внешних исполняемых компонентов используется механизм портов языка Erlang. Особенностью данного механизма является то, что не предусмотрена возможность отправить на стандартный поток ввода открытого порта сигнал «конец файла». Именно этого сигнала ожидают компоненты `wrk2il` и `il2fit` для завершения работы.

Для решения этой проблемы в механизм обработки входных данных компонентами wrk2il и il2fit была добавлена обработка фиктивного сигнала «конец файла» — специального строкового маркера, который веб-сервер записывает в открытый порт для этих программ.

В модуле wrked_app осуществляется запуск HTTP сервера и настройка обработчиков запросов. Веб-сервис обрабатывает запросы с адресами ресурсов вида «/workouts/:sport/:name/:wrk», где «:sport» и «:name» задают вид спорта и имя тренировки, а «:wrk» является описанием тренировки. Для остальных запросов сервер пытается найти и отправить статический файл из директории приложения. Для корневого пути сервер отвечает статическим файлом «index.html».

Модуль wrked_handler2 осуществляет обработку HTTP запросов и отправку созданных FIT файлов.

Обработка запроса происходит в несколько этапов, в соответствии с правилами, описанными в [24]. Модуль вычитывает часть URL с описанием тренировки, вызывает для этого описания компоненты wrk2il и il2fit, формирует предлагаемое при сохранении имя файла и отвечает клиенту сформированным FIT файлом в теле ответа.

7.2 Разработка интерфейса пользователя

Так как в операционной системе Windows интерпретатор команд используется не так часто, и большинство пользователей предпочтают графический интерфейс пользователя, реализован исполняемый компонент wrk2fit-dnd, который является аналогом сценариев wrk2fit, но предоставляет графический интерфейс — окно, в которое пользователь должен перетаскивать файлы описания тренировок для их преобразования. Исходный код данного компонента представлен в приложении А.

Исходный код содержит операции создания единственного графического

элемента приложения — надписи, которая занимает всё окно приложения, описание процедур-обработчиков различных событий (двойной щелчок мыши, пользователь перетащил в окно программы файл и т. п.), связывание процедур-обработчиков с графическим элементом.

7.3 Испытания

Оценка корректности работы программного обеспечения выполнялась на основании модульного тестирования отдельных компонентов и их модулей. Модульные тесты выполняются автоматически при сборке компонентов.

В компонент wrk2il входит модуль Test, исходный код которого представлен в приложении А.

Модуль содержит отдельные наборы тестов для различных функций компонента. Например, отдельный набор тестов для лексического анализатора, набор тестов синтаксического анализатора и т. д. Всего для компонента написано 36 тестов, которые успешно выполняются.

Модульное тестирование компонента il2fit осуществляется схожим образом. Тесты написаны непосредственно в файле компонента il2fit.cpp и включаются при сборке директивой препроцессора языка C++. В наборе тестов содержатся тесты для каждой функции компонента. Всего для компонента написано 38 тестов.

При разработке этого компонента применялся подход к разработке, называемый «разработка через тестирование». При таком подходе разработка осуществляется в цикле: написание теста конкретной функции; написание кода приложения, позволяющего успешно выполнить тест; рефакторинг написанного кода. Обзор метода приводится в [25].

8 Разработка проектной документации

8.1 Руководство пользователя

8.1.1 Введение

Данное программное обеспечение предназначено для создания файлов описания тренировок в формате FIT для спортивных компьютеров.

wrk2fit предполагается использовать индивидуально владельцами спортивных компьютеров, спортсменами-любителями и их тренерами.

Предполагается, что пользователь обладает навыками работы в операционной системе, способен создавать, копировать и удалять файлы, редактировать текстовые файлы.

8.1.2 Назначение и условия применения

Программное обеспечение wrk2fit преобразует текстовое описание тренировки в формат файлов FIT для воспроизведения на спортивных компьютерах. Преобразование осуществляется либо через интерпретатор команд вызовом команды wrk2fit, либо через графический интерфейс при перетаскивании файла с описанием тренировки в окно программы wrk2fit-dnd.

Для применения программного обеспечения требуется персональный компьютер с центральным процессором архитектуры amd64 или i386 с тактовой частотой не менее 1 ГГц, не менее 512 МиБ оперативной памяти и не менее 20 МиБ свободного пространства в файловой системе.

Программное обеспечение работает в операционных системах Linux и Windows. Требуется использовать дистрибутив программного обеспечения совместимый с операционной системой и архитектурой процессора.

8.1.3 Подготовка к работе

Дистрибутив программного обеспечения wrk2fit состоит из архивного файла, внутри которого располагаются исполняемые файлы и необходимые файлы данных. В имени файла архива содержится дата выпуска данной версии программного обеспечения.

Для подготовки к работе следует распаковать содержимое архива в любое удобное место файловой системы. Для распаковки потребуется до 20 МиБ свободного места на файловой системе. После распаковки архива в файловой системе должны появиться следующие директории: bin, erts-7.3, lib, releases.

Для проверки работоспособности следует перейти в директорию bin и убедиться, что исполняемые файлы, входящие в состав программного обеспечения, запускаются.

Следует запустить файлы wrk2il, il2fit и wrk2fit. В результате успешного запуска каждой программы должно появиться окно интерпретатора команд, без каких-либо сообщений. Окно можно закрыть.

Следует запустить файл wrk2fit-dnd. В результате успешного запуска должно появиться окно с заголовком wrk2fit. Окно можно закрыть.

8.1.4 Описание операций

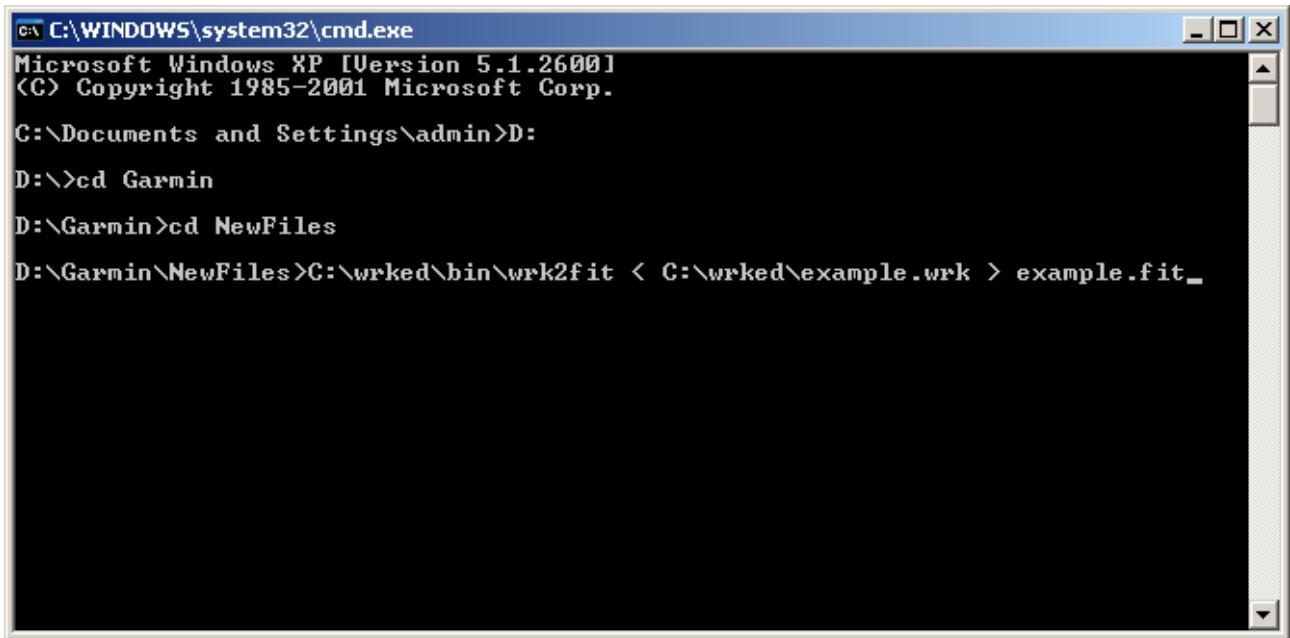
Программное обеспечение позволяет выполнять две операции над текстовыми описаниями тренировок: преобразование в формат FIT средствами командного интерпретатора и преобразование средствами графического интерфейса.

Наименование операции: преобразование описания тренировки средствами командного интерпретатора.

Условия выполнения операции: запущен и успешно функционирует интерпретатор команд.

Подготовительные действия: отсутствуют.

Основные действия: ввести в командный интерпретатор путь к исполняемому файлу wrk2fit, затем символ «<» и путь к входному файлу с описанием тренировки, затем символ «>» и желаемый путь к выходному файлу в формате FIT, выполнить команду. Пример запуска исполняемого компонента wrk2fit представлен на рисунке 25.



The screenshot shows a Microsoft Windows XP Command Prompt window titled 'C:\WINDOWS\system32\cmd.exe'. The window title bar also displays 'Microsoft Windows XP [Version 5.1.2600] (C) Copyright 1985-2001 Microsoft Corp.'. The command line shows the following sequence of commands:

```
C:\Documents and Settings\admin>D:  
D:>cd Garmin  
D:\Garmin>cd NewFiles  
D:\Garmin\NewFiles>C:\wrked\bin\wrk2fit < C:\wrked\example.wrk > example.fit
```

Рисунок 25 — Запуск компонента wrk2fit

Наименование операции: преобразование описания тренировки средствами графического интерфейса.

Условия выполнения операции: запущено и успешно функционирует приложение wrk2fit-dnd.

Подготовительные действия: отсутствуют.

Основные действия: перетащить файл с текстовым описанием тренировки в окно приложения wrk2fit-dnd, выбрать файл для сохранения результата, сохранить выходной файл. Пример работы с исполняемым компонентом wrk2fit-dnd представлен на рисунке 26.

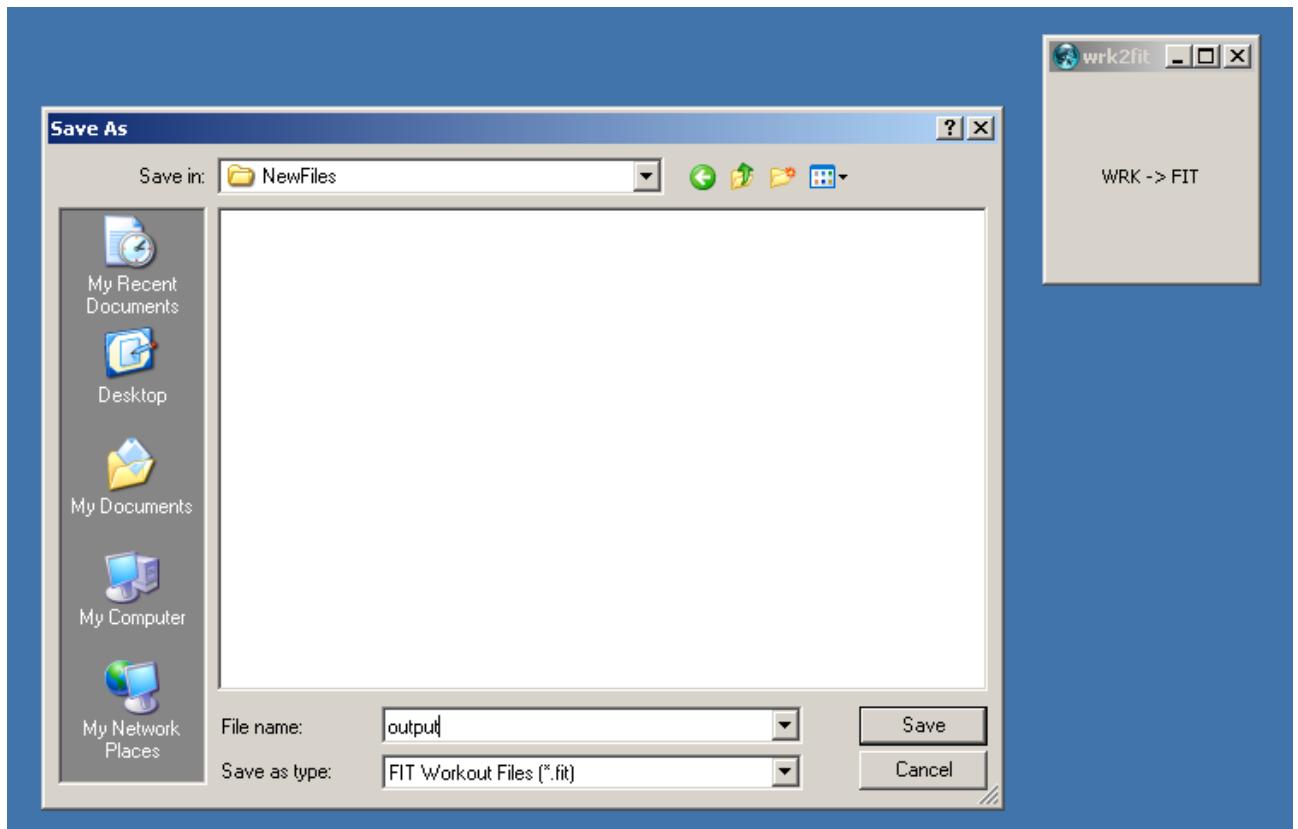


Рисунок 26 — Запуск компонента wrk2fit-dnd

Язык описания тренировки использует символы кодировки Latin-1 и содержит имя тренировки и список этапов тренировки.

Описание тренировки начинается с имени тренировки, которое задаётся как строка символов между символами кавычек (символ с кодом 34), после которой следует символ двоеточия (символ с кодом 58). Например, «"Tempo":». Имя тренировки допускается (но не рекомендуется опускать). В этом случае тренировка в памяти спортивного компьютера будет обозначаться пустой строкой символов в качестве имени.

Сразу за именем тренировки следует описание списка этапов тренировки. Список открывается символом «[» (символ с кодом 91), затем следуют описания этапов тренировки, отделённые друг от друга символом «точка с запятой» (символ с кодом 59), окончание списка обозначается символом «]» (символ с кодом 93). Например, «[<этап-1>; <этап-2>; <этап-3>]».

Каждый этап тренировки — это либо отдельный шаг тренировки, либо повторение последовательности подэтапов. Рассмотрим описание отдельного шага тренировки.

Отдельный шаг тренировки характеризуется интенсивностью, условием завершения шага и целевым показателем. В рамках отдельного шага описания этих частей отделяются друг от друга символом запятой (символ с кодом 44). Любая из этих характеристик может быть опущена. Если одновременно условие и целевой показатель, то они заменяются ключевым словом «open» (открытый шаг тренировки).

Интенсивность шага тренировки задаётся одним из следующих ключевых слов языка: «warmup» (разминка), «active» (интенсивно), «rest» (отдых), «cooldown» (заминка).

Затем следует описание условия завершения шага. Возможны следующие условия прекращения шага тренировки: по прошествии времени, по пройденному расстоянию, по частоте сердечных сокращений, по мощности, по израсходованию указанного числа килокалорий. Сначала записывают ключевое слово («time», «distance», «hr<», «hr>», «power<», «power>» или «calories»), а затем значение. Единицы измерения значения зависят от условия: секунды, минуты или часы для времени; метры или километры для расстояния; удары в минуту или проценты от максимума для ЧСС; ватты или проценты от предельной мощности для мощности; килокалории. Например, для завершения шага тренировки через полчаса можно записать «time 30 min», «time 0.5 h», «time 1600» или «time 30:00». Для завершения шага после преодоления двух километров можно записать «distance 2000» или «distance 2 km». Для завершения шага по превышению ЧСС заданного порога можно записать «hr> 150», или, если указывать порог в процентах, то «hr> 80 %».

Если для шага тренировки указан целевой показатель, спортивный компьютер будет контролировать удержание показателя в заданных пределах и сигнализировать о выходе показателя за эти пределы. Возможны следующие це-

левые показатели: скорость движения, ЧСС, частота оборотов педалей, мощность. Сначала записывают ключевое слово («speed», «hr», «cadence» или «power»), а затем допустимый диапазон значений целевого показателя. Диапазон записывается либо как два значения (нижняя и верхняя граница диапазона), разделённые символом дефис (код символа 45), либо как номер предустановленной зоны (номера зон и соответствующие диапазоны показателей задаются в настройках спортивного компьютера). Для указания зоны целевого показателя записывают ключевое слово «zone» и номер зоны.

Единицы измерения значений зависят от целевого показателя: метры в секунду или километры в час для скорости; удары в минуту или проценты от максимума для ЧСС; обороты в минуту для частоты оборотов педалей; ватты или проценты от предельной мощности для мощности. Например, для указания удерживать скорость в диапазоне от 30 до 35 километров в час, можно записать «speed 30-35 km/h» или «speed 8.3-9.7». Для указания удерживать ЧСС во второй зоне можно записать «hr zone 2». Для указания удерживать частоту оборотов педалей в диапазоне от 85 до 95 оборотов в минуту, можно записать «cadence 85-95».

Таким образом, для описания отдельного шага тренировки можно записать «warmup, time 15 min, hr zone 2». Такая запись будет означать разминочный шаг тренировки продолжительностью 15 минут, в течение которых предполагается удерживать частоту сердечных сокращений во второй зоне.

Для записи повтора последовательности этапов тренировки записывают условие повтора в круглых скобках (символы с кодами 40 и 41) и список этапов тренировки (формат записи списка этапов рассмотрен выше).

Возможно повторить список этапов заданное число раз или пока не выполнится условие (условия в этом случае идентичны условиям прекращения отдельного шага тренировки, но в этом случае они распространяются на весь список подэтапов).

Для повторения заданного числа раз записывают число и символ умноже-

ния (код символа 42) или латинскую букву х (код символа 120).

Например, для того, чтобы повторять последовательность из двух отдельных шагов четыре раза, можно записать «(4x) [active, time 8 min; rest, time 4 min]». Для того, чтобы повторять последовательность из шагов работы и отдыха пока не будет израсходована одна тысяча килокалорий, можно записать «(calories 1000) [active, time 8 min; rest, time 4 min]».

В текстовое описание тренировок допускается вставлять комментарии, заключённые между символами фигурных скобок (коды символов 123 и 125). Например, «{это комментарий}». Между элементами языка (ключевыми словами, значениями и т. д.) допускается вставлять любое количество пробелов, символом перевода строки или символов табуляции.

Пример описания тренировки представлен в листинге на рисунке 27.

```
"Workout2": [
    { Стандартная разминка }
    warmup, time 15 min, hr zone 2;
    (time 5 min) [
        active, time 1 min, cadence 105-115;
        warmup, time 1 min, cadence 85-95
    ];
    active, time 5 min, hr zone 4;
    rest, time 5 min;
    { Основная часть }
    (4x) [
        active, time 15 min, hr 154-158;
        rest, time 5 min
    ];
    cooldown, time 10 min
]
```

Рисунок 27 — Пример описания тренировки

Данный пример использует большинство возможностей языка (комментарии, различные виды повторений и т. д.).

8.1.5 Аварийные ситуации

При ошибках в работе аппаратных средств (кроме носителей данных и программ) восстановление функции системы возлагается на ОС.

При ошибках, связанных с программным обеспечением (ОС и драйверы устройств), восстановление работоспособности возлагается на ОС.

При неверных действиях пользователя (неправильный формат файла с описанием тренировки) программа wrk2fit выдаёт сообщение об ошибке с указанием номера ошибочной строки во входном файле, и возвращает управление интерпретатору команд. Программа wrk2fit-dnd изменяет цвет окна и выводит сообщение в центре окна, ожидает одну секунду и возвращается в рабочее состояние.

8.1.6 Рекомендации по освоению

Рекомендуется текстовые описания тренировок сохранять в определённой директории файловой системы и преобразовывать их в формат FIT по необходимости, перед записью в память устройства. Осмысленные имена файлов в последующем облегчат поиск требуемого файла.

Рассмотрим контрольный пример работы с системой.

Создайте обычновенный текстовый файл (файл с расширением txt) в любом доступном текстовом редакторе. В качестве содержимого файла введите следующий текст и сохраните файл: «"Test":[warmup, open; active, open; cooldown, open]».

Запустите программу wrk2fit-dnd и дождитесь появление окна программы. Подключите спортивный компьютер через USB кабель к ПК, до-

ждитесь определения спортивного компьютера операционной системой.

Перетащите созданный ранее текстовый файл в окно программы wrk2fit-dnd. Система предложит выбрать имя для сохранения выходного файла. Сохраните файл в директорию «Garmin\NewFiles» в памяти спортивного компьютера.

Закройте окно программы и безопасно извлеките накопитель, связанный с памятью спортивного компьютера. Включите спортивный компьютер и перейдите в раздел тренировок. В данном разделе должна появиться тренировка с именем «Test», состоящая из трёх шагов. Удалите тестовую тренировку из памяти спортивного компьютера.

8.2 Руководство системного администратора

8.2.1 Общие сведения

Данное программное обеспечение предназначено для создания файлов описания тренировок в формате FIT для спортивных компьютеров.

Программное обеспечение может использоваться как полностью локально, так и содержит серверную часть, для обеспечения создания файлов тренировок через сеть.

8.2.2 Архитектура и принципы функционирования

В состав программного обеспечения входит несколько программных компонентов, которые можно отнести исключительно к локальной, не сетевой части, и HTTP сервер для обработки удалённых запросов на преобразование текстового описания тренировки.

Программа wrk2il преобразует текстовое описание тренировки в текст на промежуточном языке и проверяет корректность этого текста. Это исполняемый файл в формате операционной системы.

Программа il2fit преобразует текст на промежуточном языке в двоичный формат файлов FIT. Это исполняемый файл в формате операционной системы.

Программа wrk2fit объединяет программы wrk2il и il2fit в единую программу с помощью конвейера. Это файл сценариев интерпретатора команд sh для ОС Linux или cmd для ОС Windows.

Программа wrk2fit-dnd — это аналог сценария wrk2fit, но с графическим интерфейсом пользователя. Представляет собой графическое окно для перетаскивания в него файлов с текстовым описанием тренировки. Для нормального функционирования требует наличия и исправной работы программ wrk2il и il2fit.

Управление серверной частью программного обеспечения осуществляется через сценарий интерпретатора команд wrked. Через него осуществляется запуск и останов сервера, опрос его состояния. Кроме того, при работе в ОС Windows сценарий используется для регистрации HTTP сервера в списке служб операционной системы.

8.2.3 Системные требования

Для применения программного обеспечения требуется персональный компьютер с центральным процессором архитектуры amd64 или i386 с тактовой частотой не менее 1 ГГц, не менее 512 МиБ оперативной памяти и не менее 20 МиБ свободного пространства в файловой системе.

Программное обеспечение работает в операционных системах Linux и Windows. Требуется использовать дистрибутив программного обеспечения совместимый с операционной системой и архитектурой процессора.

8.2.4 Установка программы

Дистрибутив программного обеспечения wrk2fit состоит из архивного

файла, внутри которого располагаются исполняемые файлы и необходимые файлы данных. В имени файла архива содержится дата выпуска данной версии программного обеспечения.

Для подготовки к работе следует распаковать содержимое архива в любое удобное место файловой системы. Для распаковки потребуется до 20 МиБ свободного места на файловой системе. Следует ограничить возможность модификации распакованных файлов пользователями. После распаковки архива в файловой системе должны появиться следующие директории: bin, erts-7.3, lib, releases.

Директория bin содержит исполняемые компоненты программного обеспечения: программы wrk2il, il2fit, wrk2fit, wrk2fit-dnd и серверный сценарий управления wrked. Остальные файлы в данной директории являются служебными и не должны модифицироваться или перемещаться.

Директория erts-7.3 содержит служебные файлы виртуальной машины сервера. Её содержимое не должно модифицироваться или перемещаться.

Директория lib содержит программные компоненты реализации HTTP сервера. Её содержимое не должно модифицироваться или перемещаться.

Директория releases содержит поддиректорию 0.3.12, в которой размещается конфигурационный файл сервера. Файлы и директории в директории releases не должны перемещаться и, за исключением файла конфигурации, не должны модифицироваться.

8.2.5 Административная консоль

Управление работой серверной части программного обеспечения осуществляется средствами сценария wrked интерпретатора команд, расположенного в директории bin. Сценарий распознаёт ряд команд и должен запускаться как «wrked <команда>».

Команды «start» и «stop» служат для запуска и останова HTTP сервера.

Команда «restart» объединяет две команды и инициирует сначала остановку, а затем повторный запуск сервиса.

При работе под операционной системой Windows сценарий распознаёт команды «install» и «uninstall». Команда «install» регистрирует сервер с списке служб операционной системы. В этом случае сервис будет запускаться при запуске операционной системы наряду с другими службами Windows. Команда «uninstall» останавливает сервис и отменяет такую регистрацию.

Команда «ping» служит для проверки состояния сервиса и завершается успешно если сервис запущен и функционирует.

8.2.6 Файл конфигурации

Локальные компоненты программного обеспечения не имеют файлов конфигурации.

Файл конфигурации сервера располагается в директории releases, в поддиректории 0.3.12 (текущая версия серверной части программного обеспечения) и имеет имя sys.config. Файл имеет специальный формат. Редактирование данного файла подразумевает, что формат будет сохраняться. Файл конфигурации по умолчанию представлен в листинге на рисунке 28.

```
%% -*- mode: erlang -*-
[ {wrked, [ {addr, {127,0,0,1}},
           {port, 1080},
           {il2fit_path, "bin/il2fit"},
           {wrk2il_path, "bin/wrk2il"} ] } ].
```

Рисунок 28 — Файл конфигурации сервера

Параметр «addr» служит для задания адреса сетевого интерфейса, на кото-

ром сервер должен принимать соединения. Если, например, требуется принимать соединения через интерфейс с IP-адресом 192.168.1.12, то значение параметра нужно поменять на «{addr, {192,168,1,12}}». Адрес 0.0.0.0 служит для прослушивания сразу всех имеющихся в системе сетевых интерфейсов.

Параметр «port» служит для задания номера TCP порта, на котором сервер должен принимать соединения.

Параметры «wrk2il_path» и «il2fit_path» служат для указания путей в файловой системе к программным компонентам wrk2il и il2fit, которые сервер использует для работы.

8.2.7 Обязательная начальная настройка

Перед началом использования серверной части программного обеспечения, требуется указать абсолютные пути в файловой системе к компонентам wrk2il и il2fit в файле конфигурации сервера (см. раздел «Файл конфигурации»).

Следует поменять значение параметра «addr» в файле конфигурации на адрес сетевого интерфейса сервера или на адрес 0.0.0.0. Иначе сервер будет принимать соединения только по локальному интерфейсу 127.0.0.1 и будет недоступен из сети.

Если требуется постоянная работа сервера, при работе под ОС Windows рекомендуется зарегистрировать сервер в списке служб операционной системы (см. раздел «Административная консоль»). Иначе потребуется производить запуск и останов сервера вручную средствами сценария администрирования.

8.2.8 Проверка правильности функционирования

Для проверки правильности функционирования локальных программных компонентов следует выполнить контрольный пример из «Руководства пользователя».

К проверке правильности функционирования сервера следует приступать после обязательной начальной настройки и запуска сервера.

В любом интернет-обозревателе следует ввести в адресной строке URL «`http://127.0.0.1:1080/workouts/[warmup,open;active,open;cooldown,open]`», подставив вместо 127.0.0.1 и 1080 реально используемые адрес и номер порта сервера. При переходе по указанному URL, сервер должен передать двоичный файл в формате FIT. При указании неправильного описания тренировки (например URL «`http://127.0.0.1:1080/workouts/[]`»), HTTP ответ сервера должен иметь статус 400 Bad Request. При указании неправильного URL (например, «`http://127.0.0.1:1080/notfound`»), HTTP ответ сервера должен иметь статус 404 Not Found.

8.2.9 Аварийные ситуации

Если на клиентские запросы сервер отвечает HTTP сообщением со статусом 500 (внутренняя ошибка сервера), то, вероятнее всего, в файле конфигурации указаны неправильные пути к программным компонентам wrk2il и il2fit, либо эти программные компоненты были перемещены или повреждены. Следует указать правильные пути в файле конфигурации сервера и перезапустить сервер.

Если программные компоненты wrk2il и il2fit оказались повреждены, следует удалить программное обеспечение и выполнить установку заново. При этом рекомендуется скопировать имеющийся файл конфигурации сервера и использовать его в новой инсталляции.

Если после запуска сервер не переходит в рабочее состояние (см. раздел «Административная консоль»), то, вероятно, номер порта, указанный в файле конфигурации, уже используется какой-либо другой службой. Требуется заменить номер порта в файле конфигурации на другой, незанятый, и запустить сервер ещё раз.

8.3 Руководство по установке

Дистрибутив программного обеспечения состоит из архивного файла в формате Zip, внутри которого располагаются исполняемые файлы и необходимые файлы данных. В имени файла архива содержится дата выпуска данной версии программного обеспечения. Например, «wrked-20160508.zip».

Для подготовки к работе следует распаковать содержимое архива в любое удобное место файловой системы. Для распаковки потребуется до 20 МиБ свободного места на файловой системе. Следует ограничить возможность модификации распакованных файлов пользователями.

После распаковки архива в файловой системе должны появиться следующие директории: bin, erts-7.3, lib, releases.

Директория bin содержит исполняемые компоненты программного обеспечения.

9 Оценка эффективности программного продукта

Под эффективностью понимается отношение эффекта, достигаемого от внедрения программного обеспечения, к затратам. Под затратами подразумеваются совокупные затраты на приобретение, установку, настройку и поддержку программного обеспечения.

Главный эффект от использования программного продукта состоит в увеличении производительности деятельности пользователя по созданию сценариев тренировок.

Деятельность по созданию сценария тренировки можно разделить на несколько действий. Далее можно оценить увеличение или уменьшение производительности пользователя в сравнении с другими программными продуктами, рассмотренными в разделе «Обзор существующих продуктов».

Если на совершении действия пользователь экономит $\Delta T = B_i - A_i$ единиц времени, то повышение производительности пользователя можно оценить по следующей формуле:

$$P_i = 100 \frac{B_i - A_i}{A_i} \quad (1)$$

где P_i — увеличение производительности действия, в процентах;

B_i — время на действие без использования программного обеспечения;

A_i — время на действие при использовании программного обеспечения.

Тогда общее увеличение производительности пользователя составит:

$$P = \sum P_i \quad (2)$$

Оценка увеличения производительности конкретных действий пользователя при использовании программного продукта приведена в таблице 4. Увеличение производительности оценивается в сравнении с использованием встроенного в спортивный компьютер редактора сценариев тренировок.

Таблица 4 — Производительность действий пользователя

Действие	B_i , мин	A_i , мин	P_i , %
Запуск редактора сценариев	0,5	0,17	194
Ввод сценария	2	0,75	167
Подготовка сценария к использованию	0,13	0,67	-81

Таким образом, если сравнивать производительность пользователя при использовании программного продукта и при использовании встроенного в спортивный компьютер редактора тренировок, производительность выше на 280 %.

При использовании аналогов, рассмотренных в разделе «Обзор существующих продуктов» производительность пользователя не будет кардинально отличаться.

Для оценки эффективности программного продукта в сравнении с аналогами может использоваться индекс эксплуатационно-технического уровня продукта. Этот показатель является обобщённой характеристикой эксплуатационных свойств продукта, его возможностей, степени новизны.

Индекс эксплуатационно-технического уровня определяется как:

$$J = \sum_{i=1}^n w_i x_i \quad (3)$$

где n — число рассматриваемых показателей;

w_i — коэффициент весомости показателя в долях единиц;

x_i — относительный показатель качества, устанавливаемый экспертным путём по выбранной шкале оценки (например, от 1 до 5).

Выполним сравнение эксплуатационно-технических уровней разработанного программного продукта и наиболее близкого аналога — сервиса Garmin Connect по следующему ряду показателей:

- удобство пользовательского интерфейса;

- скорость доступа к данным;
- полнота реализуемых возможностей;
- скорость обучения пользователя;
- нетребовательность к ресурсам.

Сравнение приводится в таблице 5.

Таблица 5 — Показатели качества в сравнении с аналогом

Показатель	w_i	Проект		Аналог	
		$x_i^{(1)}$	$w_i x_i^{(1)}$		$w_i x_i^{(2)}$
Удобство пользовательского интерфейса	0,1	2	0,2	5	0,5
Скорость доступа к данным	0,2	5	1	2	0,4
Полнота реализуемых возможностей	0,3	5	1,5	4	1,2
Скорость обучения пользователя	0,25	2	0,5	4	1
Нетребовательность к ресурсам	0,15	5	0,75	2	0,3
		3,95		3,4	

Отношение двух индексов называют коэффициентом технического уровня проектируемого программного продукта по отношению к аналогу:

$$A = \frac{J_1}{J_2} \quad (4)$$

где J_1 — индекс эксплуатационно-технического уровня продукта;

— индекс эксплуатационно-технического уровня аналога.

Считается, что разработка системы обоснованна, если .

Индекс эксплуатационно-технического уровня для программного продукта , индекс для аналога . Коэффициент технического уровня , что подразумевает оправданность разработки программного продукта.

10 Мероприятия по охране труда

10.1 Общие требования безопасности при работе за компьютером

Все персональные ЭВМ, используемые в работе, должны соответствовать СанПиН 2.2.2/2.4.1340-03.

К работе на компьютере допускаются лица не моложе 18 лет, обученные и прошедшие инструктаж по охране труда.

Лица, работающие с персональными электронно-вычислительными машинами (ПЭВМ) более 50 % рабочего времени (профессионально связанные с эксплуатацией ПЭВМ), должны проходить обязательные предварительные (при поступлении на работу) и периодические медицинские осмотры в установленном порядке.

Все лица, работающие на компьютерах, должны знать меры защиты и приемы оказания первой медицинской помощи при поражении электрическим током.

При работе с ПК имеет место наличие вредных факторов. Работа на компьютерах связана с нагрузками на зрение, опорно-двигательный аппарат, а также нагрузками эмоционального и психического характера. Нагрузки на зрение обычно проявляются в раздражении и усталости глаз. Из всех видов расстройств опорно-двигательного аппарата наиболее часто встречаются боли в спине, шее и плечевом поясе.

К категории вредных факторов психологического характера можно отнести несколько видов расстройств, которые объединены вместе, поскольку они вызывают изменения в эмоциональной сфере. Усталость является наиболее часто встречающимся расстройством.

Для снижения воздействия неблагоприятных факторов и получения оптимального результата при работе на компьютере необходимо уделить внимание:

- 1) правильному выбору оборудования;
- 2) должной подготовке и обучению операторов;
- 3) регулярному техническому обслуживанию;
- 4) правильному освещению рабочего места.

Помещения, в которых эксплуатируются ПЭВМ, должны иметь естественное и искусственное освещение. Оконные проемы должны быть оборудованы регулируемыми жалюзи или занавесями. Необходимо также следить, чтобы свет из окна или от осветительных приборов не отражался от экрана монитора.

Помещения, в которых эксплуатируются ПЭВМ, должны быть оборудованы защитным заземлением в соответствии с техническими требованиями по эксплуатации.

Подключение ПЭВМ в электрическую сеть производится только через специально установленные штепсельные розетки и вилки с заземляющим контактом. Подключение компьютера проводом без вилки запрещается.

В помещениях, оборудованных ПЭВМ, должна проводиться ежедневная влажная уборка и систематическое проветривание после каждого часа работы на ПЭВМ.

Для обеспечения нормируемых значений освещенности необходимо проводить чистку стекол оконных рам и светильников не реже двух раз в год и своевременно заменять перегоревшие лампы.

Рабочие столы следует размещать таким образом, чтобы экраны компьютеров были ориентированы боковой стороной к световым проемам, чтобы естественный свет падал преимущественно слева. Освещение не должно создавать бликов на поверхности экрана.

Экран монитора должен находиться от глаз пользователя на расстоянии 60–70 см, но не ближе 50 см с учётом размеров алфавитно-цифровых знаков и символов.

Конструкция рабочего стола должна обеспечивать оптимальное размеще-

ние на рабочей поверхности используемого оборудования с учетом его количества и конструктивных особенностей, характера выполняемой работы.

Требования к рабочему столу должны быть следующими:

- 1) рабочий стол должен регулироваться по высоте в пределах 68–80 см. При отсутствии возможности регулировки высоты, она должна составлять 72,5 см;
- 2) ширина рабочей поверхности стола под персональным компьютером — 80–140 см, чтобы с обеих сторон было свободное пространство 30–50 см;
- 3) рабочий стол должен иметь пространство для ног высотой не менее 60 см, шириной не менее 50 см, глубиной на уровне колен , не менее 45 см.

Рабочее место пользователя ПЭВМ следует оборудовать подставкой для ног, имеющей ширину не менее 30 см, глубину не менее 40 см.

Клавиатуру следует располагать на поверхности стола на расстоянии 10–30 см от края стола.

Для снижения утомления при работе на ПЭВМ необходимо соблюдать режим труда и отдыха. Рациональный режим труда и отдыха предусматривает соблюдение регламентированных перерывов.

10.2 Безопасность перед началом работы

Перед началом работы оператор должен совершить следующие действия:

- 1) отрегулировать освещённость на рабочем месте, убедиться в отсутствии отражений (бликов) на экране и клавиатуре, а также встречного светового потока. Для устранения бликов, вызванных ярким солнечным светом, необходимо закрыть окно жалюзи;
- 2) осмотреть рабочее место, проверить правильность подключения оборудования к электросети, убедится в отсутствии повреждений электрических кабелей;
- 3) протереть салфеткой поверхность экрана;

- 4) убедиться в наличии специального коврика для устройства «мышь»;
- 5) проверить правильность установки стола, кресла, подставок для ног, положения оборудования, угла наклона экрана; при необходимости отрегулировать положение кресла, а также расположить оборудование так, чтобы исключить неудобные позы, длительное напряжение тела.

10.3 Безопасность во время работы за компьютером

Требования к безопасности при работе на компьютере включают следующие позиции:

- 1) при необходимости, отрегулировать яркость и контрастность монитора.
Не следует делать изображение слишком ярким;
- 2) чтобы монитор не загрязнялся пылью, надо регулярно стирать ее с экрана тканью с антistатиком;
- 3) не следует открывать крышку монитора и трогать находящиеся там детали;
- 4) минимально допустимое расстояние между лицом пользователя и монитором персонального компьютера 50 см.

Работник обязан:

- 1) выполнять только ту работу, которая ему была поручена, и по которой он прошёл соответствующую подготовку и обучение;
- 2) содержать в порядке и чистоте рабочее место в течение всего рабочего дня;
- 3) держать открытыми все вентиляционные отверстия устройств;
- 4) корректно закрыть все активные задачи при необходимости прекращения работы на относительно короткое время;
- 5) выполнять санитарно-гигиенические требования;
- 6) использовать регламентированные перерывы в работе для отдыха и выполнения рекомендованных упражнений для глаз, шеи, рук, ног;

7) соблюдать расстояние от глаз до экрана в пределах 60–70 см.

10.4 Безопасность при возникновении аварийной ситуации

При возникновении аварийной ситуации оператор должен совершить следующие действия:

- 1) если на частях оборудования компьютера обнаружен электрический ток (искрение, ощущение тока и т. д.), немедленно отключить компьютер от электрической сети и доложить о неисправности непосредственному руководителю. Без его указания к работе не приступать;
- 2) при прекращении подачи электроэнергии компьютер необходимо отключить;
- 3) при возникновении возгорания, немедленно отключить компьютер. Сообщить о пожаре руководителям работ, принять меры по ликвидации очага возгорания первичными средствами пожаротушения и вызвать пожарную охрану;
- 4) при несчастном случае в первую очередь освободить пострадавшего от травмирующего фактора. При освобождении пострадавшего от действия электрического тока необходимо следить за тем, чтобы самому не оказаться в контакте с токоведущей частью. Необходимо о происшедшем несчастном случае сообщить руководителю работ и оказать первую медицинскую помощь пострадавшему.

10.5 Безопасность по окончании работы

По окончании работы производятся следующие действия:

- 1) закрыть все активные задачи (программы);
- 2) отключить питание системного блока;
- 3) отключить питание всех периферийных устройств;

- 4) отключить общее питание ПК;
- 5) очистить внешние поверхности ПК мягкой хлопчатобумажной тканью;
- 6) навести порядок на рабочем столе и убрать все ненужные документы.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была исследована возможность упрощения работы со сценариями тренировок для спортивных компьютеров, а также рассмотрен вопрос о применении таких технических средств при подготовке спортсменов, представляющих на различных соревнованиях Академию маркетинга и социально-информационных технологий ИМСИТ.

Проанализирована предметная область исследования — спортивные компьютеры и сценарии тренировок в формате FIT для воспроизведения на таких компьютерах. Результатом исследования предметной области является не-противоречивая модель предметной области с выявленными сущностями и связями между ними.

Данная модель использована при выявлении требований к будущему программному обеспечению, повлияла на выбор инструментов и языков для реализации программного обеспечения, явилась основанием для структур данных, использованных в реализации.

Использование функциональных языков программирования с мощной системой типов позволяет очень точно описать предметную область, отразить в типах функций и значений ограничения и связи, существующие в модели предметной области. Обеспечение корректности программного обеспечения осуществляется системой типов и компилятором.

В результате анализа требований, сформирован набор задач, которые должно решать программное обеспечение, предложен новый способ решения этих задач — компьютерный язык описания тренировок для спортивных компьютеров, его трансляция в машиночитаемый формат, применяемый в спортивных компьютерах.

Текстовый компьютерный язык описания тренировок позволяет использовать все преимущества текстового формата данных: наличие множества универсальных инструментов для работы с текстом, возможность использования си-

стем контроля версий для отслеживания описаний тренировок, возможность получить информацию из файла даже без сопутствующего формату файлов программного обеспечения.

По мере работы над проектом, обозначился ряд новых направлений для развития разработанного программного обеспечения. К новым возможностям, отсутствующим в текущей реализации, относится, например, графическое представление описания тренировки и графических редактор такого представления в составе имеющейся веб-службы. Такие возможности должны быть проанализированы и реализованы в будущих версиях программного обеспечения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Edge 500 Owner's Manual [Текст] / Garmin Ltd. — [Б. м.: б. и.], 2013. — 64 с.
2. Flexible & Interoperable Data Transfer (FIT) Protocol [Текст] / Dynastream Innovations Inc. — [Б. м.: б. и.], 2014. — 35 с.
3. FIT File Types Description [Текст] / Dynastream Innovations Inc. — [Б. м.: б. и.], 2015. — 86 с.
4. Керниган, Б. Практика программирования [Текст] / Б. Керниган, Р. Пайк. — М.: Вильямс, 2015. — 288 с.
5. Хант, Э. Программист-прагматик. Путь от подмастерья к мастеру [Текст] / Э. Хант, Д. Томас. — М.: Лори, 2016. — 270 с.
6. Реймонд, Э. Искусство программирования для Unix [Текст] / Э. Реймонд. — М.: Вильямс, 2016. — 544 с.
7. Laboon, B. A Friendly Introduction to Software Testing [Электронный ресурс]. — [Б. м.: б. и.], 2016. — Режим доступа: <https://github.com/laboon/ebook> (дата обращения: 08.05.2016).
8. Фаулер, М. Предметно-ориентированные языки программирования [Текст] / М. Фаулер. — М.: Вильямс, 2011. — 576 с.
9. Lipováča, M. Learn You a Haskell for Great Good! [Электронный ресурс]. — [Б. м.: б. и.], 2013. — Режим доступа: <http://learnyouahaskell.com/chapters> (дата обращения: 08.05.2016).
10. Душкин, Р. Функциональное программирование на языке Haskell [Текст] / Р. Душкин. — М.: ДМК Пресс, 2016. — 608 с.
11. Minsky, Y. Effective ML Revisited [Электронный ресурс]. — [Б. м.: б. и.], 2011. — Режим доступа: <https://blogs.janestreet.com/effective-ml-revisited/> (дата обращения: 08.05.2016).
12. Мински, Я. Программирование на языке OCaml [Текст] / Я. Мински, А. Мадхавапедди, Д. Хикки. — М.: ДМК Пресс, 2014. — 536 с.
13. Компиляторы: принципы, технологии и инструментарий [Текст] / А. Ахо, М.

Лам, Р. Сети, Д. Ульман. — 2-е изд. — М.: Вильямс, 2015. — 1184 с.

14. FIT SDK Introductory Guide [Текст] / Dynastream Innovations Inc. — [Б. м.: б. и.], 2015. — 13 с.
15. Мейерс, С. Эффективный и современный C++. 42 рекомендации по использованию C++11 и C++14 [Текст] / С. Мейерс. — М.: Вильямс, 2015. — 304 с.
16. Roseman, M. TkDocs: Information you need to build high-quality Tk user interfaces [Электронный ресурс]. — [Б. м.: б. и.], 2015. — Режим доступа: <http://www.tkdocs.com/tutorial/index.html> (дата обращения: 08.05.2016).
17. Родригес, А. Web-сервисы RESTful: основы [Электронный ресурс]. — [Б. м.: б. и.], 2015. — Режим доступа: <http://www.ibm.com/developerworks/ru/library/ws-restfu/> (дата обращения: 08.05.2016).
18. REST API Tutorial [Электронный ресурс]. — [Б. м.: б. и.], 2016. — Режим доступа: <http://www.restapitutorial.com/> (дата обращения: 08.05.2016).
19. Hoguin, L. Efficient Web Applications with Erlang and Cowboy [Электронный ресурс]. — [Б. м.: б. и.], 2012. — Режим доступа: <http://ninenines.eu/talks/oscon2012/oscon2012.html> (дата обращения: 08.05.2016).
20. Wlaschin, S. Designing with types [Электронный ресурс]. — [Б. м.: б. и.], 2013. — Режим доступа: <http://fsharpforfunandprofit.com/series/designing-with-types.html> (дата обращения: 08.05.2016).
21. Pottier, F. Menhir Reference Manual [Электронный ресурс]. — [Б. м.: б. и.], 2016. — Режим доступа: <http://cristal.inria.fr/~fpottier/menhir/manual.pdf> (дата обращения: 08.05.2016).
22. Armstrong, J. Programming Erlang: Software for a Concurrent World [Текст] / J. Armstrong. — 2-е изд. — Raleigh: Pragmatic Bookshelf, 2013. — 548 с.
23. Trottier-Hebert, F. Learn You Some Erlang for Great Good! [Электронный ресурс]. — [Б. м.: б. и.], 2015. — Режим доступа: <http://learnyousomeerlang.com/content> (дата обращения: 08.05.2016).
24. Hoguin, L. Cowboy User Guide [Электронный ресурс]. — [Б. м.: б. и.], 2016. — Режим доступа: <http://ninenines.eu/docs/en/cowboy/2.0/guide/> (дата обращения: 08.05.2016).

обращения: 08.05.2016).

25. Martin, R. The Cycles of TDD [Электронный ресурс]. — [Б. м.: б. и.], 2014. — Режим доступа: <http://blog.cleancoder.com/uncle-bob/2014/12/17/TheCyclesOfTDD.html> (дата обращения: 08.05.2016).

ПРИЛОЖЕНИЕ А

Исходный код основных модулей

wrk2il/workout.mli

```
module Sport : sig
  type t = Cycling
    | Running
    | Swimming
    | Walking

  val of_string : string -> t

  val to_string : t -> string
end

module Speed : sig
  type t = private float          (* m/s *)
  type zone = private int

  val of_float : float -> t
  val zone_of_int : int -> zone

  val from_kmph : float -> t
end

module Cadence : sig
  type t = private int            (* rpm *)
  type zone = private int

  val of_int : int -> t
  val zone_of_int : int -> zone
end

module Heart_rate : sig
  type absolute = private int     (* bpm *)
  type percent = private int      (* 0-100 % of max *)
  type zone = private int         (* 1-5 *)

  val absolute_of_int : int -> absolute
  val percent_of_int : int -> percent
  val zone_of_int : int -> zone

  type t = Absolute of absolute
    | Percent of percent
end

module Power : sig
  type absolute = private int     (* W *)
  type percent = private int      (* 0-1000 % of FTP *)
  type zone = private int         (* 1-7 *)

  val absolute_of_int : int -> absolute
  val percent_of_int : int -> percent
  val zone_of_int : int -> zone

  type t = Absolute of absolute
    | Percent of percent
end
```

```

end

module Condition : sig
  type order = Less | Greater

  type calories = private int (* kcal *)
  type distance = private int (* m *)
  type time = private int (* s *)

  val calories_of_int : int -> calories
  val distance_of_int : int -> distance
  val time_of_int : int -> time

  type t = Time of time
    | Distance of distance
    | Heart_rate of (order * Heart_rate.t)
    | Calories of calories
    | Power of (order * Power.t)
end

module Repeat : sig
  type times = private int

  val times_of_int : int -> times

  type t = Times of times
    | Until of Condition.t
end

module Target : sig
  module Value : functor (S : sig type t type zone end) ->
    sig
      type range = private (S.t * S.t)

      type t = Zone of S.zone
        | Range of range

      val range_of_pair : S.t * S.t -> range
    end
end

module Cadence_value : module type of Value (Cadence)
module Heart_rate_value : module type of Value (Heart_rate)
module Power_value : module type of Value (Power)
module Speed_value : module type of Value (Speed)

type t = Speed of Speed_value.t
  | Heart_rate of Heart_rate_value.t
  | Cadence of Cadence_value.t
  | Power of Power_value.t
end

module Intensity : sig
  type t = Warm_up
    | Active
    | Rest
    | Cool_down

  val of_string : string -> t
  val to_string : t -> string
end

```

```

module Step : sig
  type single = {
    name      : string option;
    duration  : Condition.t option;
    target    : Target.t option;
    intensity : Intensity.t option;
  } and repeat = {
    condition : Repeat.t;
    steps     : t Non_empty_list.t;
  } and t = Single of single
           | Repeat of repeat
end

type t = {
  name  : string option;
  sport : Sport.t option;
  steps : Step.t Non_empty_list.t;
}

(* {2 Capabilities flags} *)

module Capability : sig
  type t = Speed
    | Heart_rate
    | Distance
    | Cadence
    | Power
    | Grade
    | Resistance

  val to_int32 : t -> int32
end

val caps : t -> Capability.t list

```

wrk2il/parser.mly

```

%token <float> FLOAT
%token <int> INTEGER
%token <string> STRING

%token L_BRACKET R_BRACKET
%token L_PAREN R_PAREN
%token LESS GREATER

%token COLON
%token COMMA
%token HYPHEN
%token PERCENT
%token SEMICOLON
%token TIMES

%token CADENCE
%token CALORIES
%token DISTANCE
%token HR
%token POWER
%token SPEED
%token TIME

```

```

%token BPM
%token H
%token KCAL
%token KM
%token KMPH
%token M
%token MIN
%token MPS
%token RPM
%token S
%token W

%token OPEN
%token ZONE

%token EOF

%token <Workout.Sport.t> SPORT
%token <Workout.Intensity.t> INTENSITY

%start <Workout.t> workout

%%

workout:
| name = option(terminated(STRING, COLON))
  sport = option(SPORT)
  steps = steps EOF
  { { name; sport; steps } }

time_spec_1:
| s = INTEGER S?      { Workout.Condition.time_of_int s }
| min = INTEGER MIN { Workout.Condition.time_of_int (60 * min) }
| min = FLOAT MIN    { Workout.Condition.time_of_int (int_of_float (60.0 *. min)) }
}
| h = INTEGER H       { Workout.Condition.time_of_int (3600 * h) }
| h = FLOAT H        { Workout.Condition.time_of_int (int_of_float (3600.0 *. h)) }
}

%inline
separated_triplet(X, a, Y, b, Z): x = X a y = Y b z = Z { x, y, z }

time_spec_2:
| hms = separated_triplet(INTEGER, COLON, INTEGER, COLON, INTEGER)
  { let h, m, s = hms in
    Workout.Condition.time_of_int (h * 3600 + m * 60 + s) }
| ms = separated_pair(INTEGER, COLON, INTEGER)
  { let m, s = ms in

```



```

| pct = INTEGER PERCENT { Workout.Heart_rate.(Percent (percent_of_int pct)) }

power_spec:
| w = INTEGER W?           { Workout.Power.(Absolute (absolute_of_int w)) }
| pct = INTEGER PERCENT { Workout.Power.(Percent (percent_of_int pct)) }

speed_spec:
| kmph = FLOAT KMPH?     { Workout.Speed.from_kmph kmph }
| kmph = INTEGER KMPH?   { Workout.Speed.from_kmph (float_of_int kmph) }
| mps = FLOAT MPS        { Workout.Speed.of_float mps }
| mps = INTEGER MPS      { Workout.Speed.of_float (float_of_int mps) }

cadence_spec: rpm = INTEGER RPM? { Workout.Cadence.of_int rpm }

time_condition: TIME t = time_spec { t }

distance_condition: DISTANCE d = distance_spec { d }

calories_condition: CALORIES c = calories_spec { c }

hr_condition:
| HR LESS h = hr_spec    { Workout.Condition.Less, h }
| HR GREATER h = hr_spec { Workout.Condition.Greater, h }

power_condition:
| POWER LESS p = power_spec { Workout.Condition.Less, p }
| POWER GREATER p = power_spec { Workout.Condition.Greater, p }

condition:
| t = time_condition     { Workout.Condition.Time t }
| c = calories_condition { Workout.Condition.Calories c }
| d = distance_condition { Workout.Condition.Distance d }
| h = hr_condition       { Workout.Condition.Heart_rate h }
| p = power_condition    { Workout.Condition.Power p }

%inline
target_range(X): r = separated_pair(X, HYPHEN, X) { r }

hr_target:
| HR ZONE z = INTEGER
  { Workout.Target.Heart_rate_value.Zone (Workout.Heart_rate.zone_of_int z) }
| HR r = target_range(hr_spec)
  { Workout.Target.Heart_rate_value.(Range (range_of_pair r)) }

speed_target:
| SPEED ZONE z = INTEGER
  { Workout.Target.Speed_value.Zone (Workout.Speed.zone_of_int z) }
| SPEED r = target_range(speed_spec)
  { Workout.Target.Speed_value.(Range (range_of_pair r)) }

cadence_target:
| CADENCE ZONE z = INTEGER
  { Workout.Target.Cadence_value.Zone (Workout.Cadence.zone_of_int z) }
| CADENCE r = target_range(cadence_spec)
  { Workout.Target.Cadence_value.(Range (range_of_pair r)) }

power_target:
| POWER ZONE z = INTEGER
  { Workout.Target.Power_value.Zone (Workout.Power.zone_of_int z) }

```



```

target:
| h = hr_target      { Workout.Target.Heart_rate h }
| s = speed_target   { Workout.Target.Speed s }
| c = cadence_target { Workout.Target.Cadence c }
| p = power_target   { Workout.Target.Power p }

times_condition: n = INTEGER TIMES { Workout.Repeat.times_of_int n }

repeat_condition:
| t = times_condition { Workout.Repeat.Times t }
| c = condition        { Workout.Repeat.Until c }

separated_nonempty_list2(separator, X):
| x = X separator?
  { [x] }
| x = X; separator; xs = separated_nonempty_list2(separator, X)
  { x :: xs }

steps:
| l = delimited(L_BRACKET,
    separated_nonempty_list2(SEMICOLON, step), R_BRACKET)
  { Non_empty_list.of_list l }

step:
| s = single_step { Workout.Step.Single s }
| r = repeat_step { Workout.Step.Repeat r }

repeat_step:
| condition = delimited(L_PAREN, repeat_condition, R_PAREN) steps = steps
  { {Workout.Step.condition; steps} }

single_step:
| name = option(terminated(STRING, COLON))
  intensity = option(terminated(INTENSITY, COMMA))
  p = step_duration_and_target
  { let duration, target = p in
    {Workout.Step.name; duration; target; intensity} }

step_duration_and_target:
| OPEN                                { None, None }
| c = condition                        { Some c, None }
| t = target                           { None, Some t }
| c = condition COMMA t = target { Some c, Some t }

```

wrk2il/lexer.mll

```

{
open Parser
open Workout

exception Error
}

rule read = parse
| ['\t' '\n'] { read lexbuf }

| ['0'-'9']+ as i           { INTEGER (int_of_string i) }
| ('0'-'9')+ '.' ['0'-'9']+ as f { FLOAT (float_of_string f) }
| '"' ([^'"' '\n']+ as s) '"' { STRING s }

```

```

| ' {' [^'}']* ' }'                                { read lexbuf }

| "warmup"    { INTENSITY Intensity.Warm_up }
| "active"    { INTENSITY Intensity.Active }
| "rest"      { INTENSITY Intensity.Rest }
| "cooldown" { INTENSITY Intensity.Cool_down }

| "cycling"   { SPORT Sport.Cycling }
| "running"   { SPORT Sport.Running }
| "swimming"  { SPORT Sport.Swimming }
| "walking"   { SPORT Sport.Walking }

| "cadence"   { CADENCE }
| "calories"  { CALORIES }
| "distance"  { DISTANCE }
| "hr"         { HR }
| "power"     { POWER }
| "speed"     { SPEED }
| "time"      { TIME }

| "open"      { OPEN }
| "zone"      { ZONE }

| "km/h"      { KMPH }
| "m/s"       { MPS }

| "bpm"       { BPM }
| "h"          { H }
| "kcal"      { KCAL }
| "km"        { KM }
| "m"          { M }
| "min"       { MIN }
| "rpm"       { RPM }
| "s"          { S }
| "W"          { W }

| '['         { L_BRACKET }
| ']'         { R_BRACKET }
| '('         { L_PAREN }
| ')'         { R_PAREN }
| ';'         { SEMICOLON }
| '-'         { HYPHEN }
| '<'        { LESS }
| '>'        { GREATER }
| ',',        { COMMA }
| '%'        { PERCENT }
| ':'         { COLON }
| '*' | 'x' { TIMES }

| eof | "EOF" { EOF }

| _ { raise Error }

{
open Batteries

let enum lexbuf =
  Enum.from_loop true
  (fun continue ->
    if continue then
      let token = read lexbuf in

```

```

        token, token <> Parser.EOF
    else raise Enum.No_more_elements)
}

```

wrk2il/repr.ml

```

open Batteries

let from_lexbuf = Parser.workout Lexer.read

let from_channel = from_lexbuf % Lexing.from_channel

let from_string = from_lexbuf % Lexing.from_string

let heart_rate_to_channel chan hr =
  Workout.Heart_rate.(
    match hr with
      Absolute bpm    -> Printf.printf chan "%d" (bpm :> int)
    | Percent percent -> Printf.printf chan "%d%" (percent :> int)
  )

let power_to_channel chan power =
  Workout.Power.(
    match power with
      Absolute w       -> Printf.printf chan "%d" (w :> int)
    | Percent percent -> Printf.printf chan "%d%" (percent :> int)
  )

let condition_to_channel chan condition =
  Workout.Condition.(
    let order_to_channel = function
      | Less -> IO.write chan '<'
      | Greater -> IO.write chan '>' in
    match condition with
      Time s -> Printf.printf chan "time%d" (s :> int)
    | Distance m -> Printf.printf chan "distance%d" (m :> int)
    | Calories kcal -> Printf.printf chan "calories%d" (kcal :> int)
    | Heart_rate (rel, hr) ->
        (IO.nwrite chan "hr";
         order_to_channel rel;
         heart_rate_to_channel chan hr)
    | Power (rel, power) ->
        (IO.nwrite chan "power";
         order_to_channel rel;
         power_to_channel chan power)
  )
(* TODO: cleanup *)
let target_to_channel chan target =
  Workout.Target.(
    match target with
    | Speed speed ->
        (IO.nwrite chan "speed";
         match speed with
           Speed_value.Zone z ->
             Printf.printf chan "zone%d" (z :> int)
         | Speed_value.Range r ->
             let (a, b) = (r :> (Workout.Speed.t * Workout.Speed.t)) in
               Printf.printf chan "%f-%f" (a :> float) (b :> float))
    | Heart_rate hr ->

```

```

(IO.nwrite chan "hr";
match hr with
| Heart_rate_value.Zone z -> Printf.printf chan "zone%d" (z :> int)
| Heart_rate_value.Range r ->
  let (a, b) = (r :> (Workout.Heart_rate.t * Workout.Heart_rate.t)) in
  heart_rate_to_channel chan a;
  IO.write chan '-';
  heart_rate_to_channel chan b)
| Cadence cad ->
  (IO.nwrite chan "cadence";
  match cad with
  | Cadence_value.Zone z -> Printf.printf chan "zone%d" (z :> int)
  | Cadence_value.Range r ->
    let (a, b) = (r :> (Workout.Cadence.t * Workout.Cadence.t)) in
    Printf.printf chan "%d-%d" (a :> int) (b :> int))
| Power power ->
  (IO.nwrite chan "power";
  match power with
  | Power_value.Zone z -> Printf.printf chan "zone%d" (z :> int)
  | Power_value.Range r ->
    let (a, b) = (r :> (Workout.Power.t * Workout.Power.t)) in
    power_to_channel chan a;
    IO.write chan '-';
    power_to_channel chan b)
)

let repeat_to_channel chan = function
| Workout.Repeat.Times n -> Printf.printf chan "%dx" (n :> int)
| Workout.Repeat.Until c -> condition_to_channel chan c

let single_step_to_channel chan
  {Workout.Step.name; duration; target; intensity} =
Option.may (Printf.printf chan "\"%s\":\"") name;
Option.may (fun i ->
  IO.nwrite chan (Workout.Intensity.to_string i);
  IO.write chan ',' intensity;
  match duration, target with
  | None, None -> IO.nwrite chan "open"
  | Some c, None -> condition_to_channel chan c
  | None, Some t -> target_to_channel chan t
  | Some c, Some t ->
    (condition_to_channel chan c;
    IO.write chan ',';
    target_to_channel chan t))

let rec repeat_step_to_channel chan {Workout.Step.condition; steps} =
  IO.write chan '(';
  repeat_to_channel chan condition;
  IO.write chan ')';
  step_list_to_channel chan steps

and step_list_to_channel chan (s1, s) =
  IO.write chan '[';
  step_to_channel chan s1;
  List.iter (fun sn ->
    IO.write chan ';';
    step_to_channel chan sn) s;
  IO.write chan ']'

and step_to_channel chan = function
| Workout.Step.Single s -> single_step_to_channel chan s

```

```

| Workout.Step.Repeat r -> repeat_step_to_channel chan r

let to_channel chan {Workout.name; sport; steps} =
  Option.may (Printffprintf chan "%s:" name;
  Option.may (IO.nwrite chan % Workout.Sport.to_string) sport;
  step_list_to_channel chan steps

let to_string w =
  let chan = IO.output_string () in
  to_channel chan w;
  IO.close_out chan

module Il = struct
  open Workout

  let int32_caps =
    ((List.fold_left Int32.add Int32.zero) %
     (List.map Capability.to_int32)) % caps

  let int_of_heart_rate = function
    Heart_rate.Absolute bpm -> (bpm :> int) + 100
  | Heart_rate.Percent percent -> (percent :> int)

  let int_of_power = function
    Power.Absolute w -> (w :> int) + 1000
  | Power.Percent percent -> (percent :> int)

  let p_line ch line = IO.(nwrite ch line; write ch '\n')

  let p_field ch k v = p_line ch k; p_line ch v

  let p_int_field ch k = p_field ch k % string_of_int

  let p_float_field ch k = p_field ch k % string_of_float

  let p_int32_field ch k = p_field ch k % Int32.to_string

  let p_duration ch cond =
    let duration_type, field_name, value =
      Condition.(
        match cond with
        Time s ->
        "time",
        "duration_time",
        (s :> int)
      | Distance m ->
        "distance",
        "duration_distance",
        (m :> int)
      | Calories kcal ->
        "calories",
        "duration_calories",
        (kcal :> int)
      | Heart_rate (Less, hr) ->
        "hr_less_than",
        "duration_hr",
        (int_of_heart_rate hr)
      | Heart_rate (Greater, hr) ->
        "hr_greater_than",
        "duration_hr",
        (int_of_heart_rate hr)

```

```

| Power (Less, power) ->
  "power_less_than",
  "duration_power",
  (int_of_power power)
| Power (Greater, power) ->
  "power_greater_than",
  "duration_power",
  (int_of_power power)
) in
p_field ch "duration_type" duration_type;
p_int_field ch field_name value

let p_duration_opt ch = function
  None      -> p_field ch "duration_type" "open" (* TODO: duration_value? *)
  | Some cond -> p_duration ch cond

let p_target ch target =
  Target.(
    match target with
    Speed (Speed_value.Zone z) ->
      ( p_field ch "target_type" "speed";
        (* TODO: where is target_speed_zone? *)
        p_int_field ch "target_value" (z :> int);
        p_float_field ch "custom_target_speed_low" 0.0;
        p_float_field ch "custom_target_speed_high" 0.0 )
    | Speed (Speed_value.Range r) ->
      ( p_field ch "target_type" "speed";
        let a, b = (r :> (Speed.t * Speed.t)) in
        p_int_field ch "target_value" 0;
        p_float_field ch "custom_target_speed_low" (a :> float);
        p_float_field ch "custom_target_speed_high" (b :> float) )
    | Heart_rate (Heart_rate_value.Zone z) ->
      ( p_field ch "target_type" "heart_rate";
        p_int_field ch "target_hr_zone" (z :> int);
        p_int_field ch "custom_target_heart_rate_low" 0;
        p_int_field ch "custom_target_heart_rate_high" 0 )
    | Heart_rate (Heart_rate_value.Range r) ->
      ( p_field ch "target_type" "heart_rate";
        let a, b = (r :> (Heart_rate.t * Heart_rate.t)) in
        p_int_field ch "target_hr_zone" 0;
        p_int_field ch "custom_target_heart_rate_low" (int_of_heart_rate a);
        p_int_field ch "custom_target_heart_rate_high" (int_of_heart_rate b) )
    | Cadence (Cadence_value.Zone z) ->
      ( p_field ch "target_type" "cadence";
        (* TODO: where is target_cadence_zone? *)
        p_int_field ch "target_value" (z :> int);
        p_int_field ch "custom_target_cadence_low" 0;
        p_int_field ch "custom_target_cadence_high" 0 )
    | Cadence (Cadence_value.Range r) ->
      ( p_field ch "target_type" "cadence";
        let a, b = (r :> (Cadence.t * Cadence.t)) in
        p_int_field ch "target_value" 0;
        p_int_field ch "custom_target_cadence_low" (a :> int);
        p_int_field ch "custom_target_cadence_high" (b :> int) )
    | Power (Power_value.Zone z) ->
      ( p_field ch "target_type" "power";
        p_int_field ch "target_power_zone" (z :> int);
        p_int_field ch "custom_target_power_low" 0;
        p_int_field ch "custom_target_power_high" 0 )
    | Power (Power_value.Range r) ->
      ( p_field ch "target_type" "power";

```

```

let a, b = (r :> (Power.t * Power.t)) in
  p_int_field ch "target_power_zone" 0;
  p_int_field ch "custom_target_power_low" (int_of_power a);
  p_int_field ch "custom_target_power_high" (int_of_power b) )
)

let p_target_opt ch = function
  None      -> p_field ch "target_type" "open"
  | Some target -> p_target ch target

let p_single_step ch i {Step.name; duration; target; intensity} =
  p_field ch "begin" "workout_step";
  p_int_field ch "message_index" i;
  Option.may (p_field ch "wkt_step_name") name;
  p_duration_opt ch duration;
  p_target_opt ch target;
  Option.may (p_field ch "intensity" % Intensity.to_string) intensity;
  p_field ch "end" "workout_step";
  i + 1

let rec p_step ch i = function
  Step.Single s -> p_single_step ch i s
  | Step.Repeat r -> p_repeat_step ch i r
and p_repeat_step ch i {Step.condition; steps} =
  let k = List.fold_left (p_step ch) i
    (Non_empty_list.to_list steps) in
  p_field ch "begin" "workout_step";
  p_int_field ch "message_index" k;
  (match condition with
    Repeat.Times n -> (
      p_field ch "duration_type" "repeat_until_steps_cmplt";
      p_int_field ch "repeat_steps" (n :> int)
    )
    | Repeat.Until (Condition.Time s) -> (
      p_field ch "duration_type" "repeat_until_time";
      p_int_field ch "duration_time" (s :> int)
    )
    | Repeat.Until (Condition.Distance m) -> (
      p_field ch "duration_type" "repeat_until_distance";
      p_int_field ch "duration_distance" (m :> int)
    )
    | Repeat.Until (Condition.Calories kcal) -> (
      p_field ch "duration_type" "repeat_until_calories";
      p_int_field ch "duration_calories" (kcal :> int)
    )
    | Repeat.Until (Condition.Heart_rate (order, hr)) -> (
      p_field ch "duration_type" Condition.(
        match order with
          Less      -> "repeat_until_hr_less_than"
          | Greater -> "repeat_until_hr_greater_than"
        );
      p_int_field ch "duration_hr" (int_of_heart_rate hr)
    )
    | Repeat.Until (Condition.Power (order, power)) -> (
      p_field ch "duration_type" Condition.(
        match order with
          Less      -> "repeat_until_power_less_than"
          | Greater -> "repeat_until_power_greater_than"
        );
      p_int_field ch "duration_power" (int_of_power power)
    )
  )

```

```

);
p_int_field ch "duration_step" i; (* Repeat from step i *)
p_field ch "end" "workout_step";
k + 1

let rec n_steps = function
[] -> 0
| (Step.Single _) :: tail -> 1 + (n_steps tail)
| (Step.Repeat {Step.steps; _}) :: tail ->
  1 + (n_steps (Non_empty_list.to_list steps)) + (n_steps tail)

let to_channel ch ({name; sport; steps} as workout) =
  p_field ch "begin" "file_id";
  p_field ch "end" "file_id";
  p_field ch "begin" "workout";
  Option.may (p_field ch "wkt_name") name;
  Option.may (p_field ch "sport" % Sport.to_string) sport;
  p_int32_field ch "capabilities" (int32_caps workout);
  p_int_field ch "num_valid_steps"
    (n_steps (Non_empty_list.to_list steps));
  p_field ch "end" "workout";
  ignore (
    List.fold_left (p_step ch) 0 (Non_empty_list.to_list steps))
end

```

wrk2il/wrk2il.ml

```

open Batteries

let parse_args () =
  let name = ref None in
  let sport = ref None in
  let mode = ref Repr.Il.to_channel in
  Arg.parse [ "-name", Arg.String (Ref.set name % Option.some),
             " Override workout name";
             "-sport", Arg.Symbol ([ "cycling"; "running";
                                     "swimming"; "walking" ],
                                     (Ref.set sport % Option.some %
                                      Workout.Sport.of_string)),
             " Override workout sport";
             "-mode", Arg.Symbol ([ "min"; "tr" ],
                                   Ref.set mode % (function
                                         | "min" -> Repr.to_channel
                                         | "tr" -> Repr.Il.to_channel
                                         | _ -> invalid_arg "mode")),
             " Translate or minimize" ]
            (fun _anon -> ())
            "Process workout description language";
  !name, !sport, !mode

let override ?name ?sport =
  (fun workout ->
    if Option.is_some name then
      {workout with Workout.name}
    else workout) %
  (fun workout ->
    if Option.is_some sport then
      {workout with Workout.sport}
    else workout)

```

```

let () =
  let name, sport, to_channel = parse_args () in
  let lexbuf = Lexing.from_channel stdin in
  try
    Repr.from_lexbuf lexbuf |> override ?name ?sport |> to_channel stdout
  with Lexer.Error | Parser.Error ->
    Lexing.(
      let {pos_lnum; pos_bol; pos_cnum; _} = lexeme_start_p lexbuf in
      Printf.eprintf "Error near \"%s\" at line %d, column %d\n"
        (lexeme lexbuf) pos_lnum (pos_cnum - pos_bol);
      exit 1
    )

```

wrk2il/test.ml

```

open Batteries
open OUnit2

let tokens str = Lexer.enum (Lexing.from_string str) |> List.of_enum

let assert_tokens ~ctxt str toks =
  assert_equal ~ctxt (tokens str) toks

let lexer_tests =
  let open Parser in
  let open Workout in
  "Lexer" >::: [
    "Empty input" >:: (fun ctxt -> assert_tokens ~ctxt "" [EOF])
  ; "Spaces" >:: (fun ctxt -> assert_tokens ~ctxt " " [EOF])
  ; "Number" >:: (fun ctxt -> assert_tokens ~ctxt "0123" [INTEGER 123; EOF])
  ; "Name" >:: (fun ctxt -> assert_tokens
                  ~ctxt " \"xyz foo %%\" " [STRING "xyz foo %%"; EOF])
  ; "Brackets" >:::
    (fun ctxt -> assert_tokens ~ctxt
      "[[ ]] " [L_BRACKET; L_BRACKET; R_BRACKET; R_BRACKET; EOF])
  ; "HR condition" >:::
    (fun ctxt -> assert_tokens ~ctxt
      "hr zone 1" [HR; ZONE; INTEGER 1; EOF])
  ; "Workout header" >:::
    (fun ctxt -> assert_tokens ~ctxt
      "\"A\", cycling" [STRING "A"; COMMA; SPORT Sport.Cycling; EOF])
  ; "xN" >:::
    (fun ctxt -> assert_tokens ~ctxt
      "(3x) [ ]" [L_PAREN; INTEGER 3; TIMES; R_PAREN;
                   L_BRACKET; R_BRACKET; EOF])
  ; "*N" >:::
    (fun ctxt -> assert_tokens ~ctxt
      "(4 *) [ ]" [L_PAREN; INTEGER 4; TIMES; R_PAREN;
                   L_BRACKET; R_BRACKET; EOF])
  ; "Step header" >:::
    (fun ctxt -> assert_tokens ~ctxt
      "\"B\", cooldown" [STRING "B"; COMMA;
                         INTENSITY Intensity.Cool_down; EOF])
  ; "Invalid intensity" >:::
    (fun _ctxt -> assert_raises
      Lexer.Error (fun () -> tokens "cool warm down up"))
  ; "Complete step" >:::
    (fun ctxt -> assert_tokens ~ctxt
      "\"A\", warmup, time < 1 min, hr < 80 %"
      [STRING "A"; COMMA; INTENSITY Intensity.Warm_up; COMMA;

```

```

        TIME; LESS; INTEGER 1; MIN; COMMA;
        HR; LESS; INTEGER 80; PERCENT; EOF])
; "Step, no spaces" >::
(fun ctxt -> assert_tokens ~ctxt
  "\\"B\\",rest,distance<5km,power>100%"
  [STRING "B"; COMMA; INTENSITY Intensity.Rest; COMMA;
   DISTANCE; LESS; INTEGER 5; KM; COMMA;
   POWER; GREATER; INTEGER 100; PERCENT; EOF])
; "Float" >::
(fun ctxt -> assert_tokens ~ctxt
  "12.25 0.0 001.001"
  [FLOAT 12.25; FLOAT 0.0; FLOAT 1.001; EOF])
; "String with \\n" >::
(fun ctxt -> assert_raises
  Lexer.Error (fun () -> tokens "\\\"abc\\ndef\\\""))
]

let assert_parses str expected ctxt =
  assert_equal ~ctxt (Repr.from_string str) expected

let parser_tests =
let open Workout in
let empty_single_step =
  {Workout.Step.name = None;
   duration = None;
   target = None;
   intensity = None} in
let empty_workout =
  {Workout.name = None; sport = None;
   steps = Workout.Step.Single empty_single_step, []} in
"Parser" >::: [
  "Empty input" >::
  (fun _ctxt ->
    assert_raises Parser.Error (fun () -> Repr.from_string ""))
; "Simplest workout" >::
  (assert_parses "[open]" empty_workout)
; "Simplest named workout" >::
  (assert_parses "\\\"Just ride\\\": cycling [open]"
    {empty_workout with Workout.name = Some "Just ride";
                      sport = Some Workout.Sport.Cycling})
; "Open-ended step with name" >::
  (assert_parses "[\\\"Xyz\\\": open]"
    {empty_workout with
     Workout.steps =
       Workout.Step.Single {empty_single_step with
                             Workout.Step.name = Some "Xyz"}, []})
; "Open-ended step with intensity" >::
  (assert_parses "[warmup, open]"
    {empty_workout with
     Workout.steps =
       Workout.Step.Single {empty_single_step with
                             Workout.Step.intensity =
                               Some Workout.Intensity.Warm_up}, []})
; "Two workout steps" >::
  (assert_parses
    "[\\\"A\\\": warmup, open; \\\"B\\\": active, open]"
    {empty_workout with
     Workout.steps =
       Workout.Step.Single {empty_single_step with
                             Workout.Step.name = Some "A";
                             intensity = Some Workout.Intensity.Warm_up},
       Workout.Step.Single {empty_single_step with
                             Workout.Step.name = Some "B";
                             intensity = Some Workout.Intensity.Active}})
]

```

```

[Workout.Step.Single {empty_single_step with
    Workout.Step.name = Some "B";
    intensity = Some Workout.Intensity.Active}])
; "Three workout steps" >::
(assert_parses "[open; open; open]"
{empty_workout with
Workout.steps =
    Workout.Step.Single empty_single_step,
    [Workout.Step.Single empty_single_step;
    Workout.Step.Single empty_single_step]}))

; "Step with time duration" >::
(assert_parses "[time 10 min]"
{empty_workout with
Workout.steps =
    Workout.Step.Single
    {empty_single_step with
        Workout.Step.duration = Some (
            Workout.Condition.Time
            (Workout.Condition.time_of_int 600))), []})
; "Steps with alt. time duration" >::
(assert_parses
"[time 01:15:30; time 10:00]"
{empty_workout with
Workout.steps = Non_empty_list.of_list [
    Workout.Step.Single
    {empty_single_step with
        Workout.Step.duration = Some (
            Workout.Condition.Time
            (Workout.Condition.time_of_int
                (3600 + 15 * 60 + 30)))}
    ; Workout.Step.Single
    {empty_single_step with
        Workout.Step.duration = Some (
            Workout.Condition.Time
            (Workout.Condition.time_of_int 600))}]
]})

; "Steps with distance duration" >::
(assert_parses
("[distance 5000;" ^
" distance 5000 m;" ^
" distance 5 km]")
(let until_5km =
    Workout.Step.Single
    {empty_single_step with
        Workout.Step.duration = Some (
            Workout.Condition.Distance
            (Workout.Condition.distance_of_int 5000))) in
{empty_workout with
Workout.steps = Non_empty_list.of_list [
    until_5km; until_5km; until_5km ]]}))

; "Steps with calories duration" >::
(assert_parses
"[calories 1500; calories 300 kcal]"
{empty_workout with
Workout.steps = Non_empty_list.of_list [
    Workout.Step.Single
    {empty_single_step with
        Workout.Step.duration = Some (
            Workout.Condition.Calories
            (Workout.Condition.calories_of_int 1500))}]
})

```

```

; Workout.Step.Single
{empty_single_step with
  Workout.Step.duration = Some (
    Workout.Condition.Calories
      (Workout.Condition.calories_of_int 300)))}
  ]})
; "Steps with HR duration" >::
(assert_parses
  "[hr > 150; hr > 70 %; hr < 180 bpm]"
{empty_workout with
  steps = Non_empty_list.of_list [
    Step.Single
      {empty_single_step with
        Step.duration = Some (
          Condition.Heart_rate
            (Condition.Greater,
              (Heart_rate.Absolute
                (Heart_rate.absolute_of_int 150))))}
    ; Step.Single
      {empty_single_step with
        Step.duration = Some (
          Condition.Heart_rate
            (Condition.Greater,
              (Heart_rate.Percent
                (Heart_rate.percent_of_int 70))))}
    ; Step.Single
      {empty_single_step with
        Step.duration = Some (
          Condition.Heart_rate
            (Condition.Less,
              (Heart_rate.Absolute
                (Heart_rate.absolute_of_int 180))))}
    ]})
; "Steps with power duration" >::
(assert_parses
  "[power < 200 W; power > 300 %]"
{empty_workout with
  steps = Non_empty_list.of_list [
    Step.Single {empty_single_step with
      Step.duration = Some (
        Condition.Power
          (Condition.Less,
            (Power.Absolute
              (Power.absolute_of_int 200))))}
    ; Step.Single {empty_single_step with
      Step.duration = Some (
        Condition.Power
          (Condition.Greater,
            (Power.Percent
              (Power.percent_of_int 300))))}
    ]})
; "Repeat 2 times" >::
(assert_parses
  "[warmup, open; (2x) [active, time 10 min]]"
{empty_workout with
  steps = Non_empty_list.of_list
    [ Step.Single
      {empty_single_step with
        Step.intensity = Some Intensity.Warm_up}
    ; Step.Repeat
      {Step.condition =

```

```

        Repeat.Times (Repeat.times_of_int 2);
        steps = Non_empty_list.of_list
        [ Step.Single
            {empty_single_step with
                Step.intensity = Some Intensity.Active;
                duration = Some (Condition.Time
                    (Condition.time_of_int 600))) ] }
    ] })
; "Repeat until distance condition" >::
(assert_parses
  "[open; (distance 2 km) [open]]"
  {empty_workout with
    steps = Non_empty_list.of_list
    [ Step.Single empty_single_step
    ; Step.Repeat
        {Step.condition =
            Repeat.Until (Condition.Distance
                (Condition.distance_of_int 2000));
            steps = Non_empty_list.of_list
            [ Step.Single empty_single_step ] }
    ] ])
; "Workout step with target" >::
(assert_parses "[hr zone 2]"
  {empty_workout with
    steps =
      Step.Single {empty_single_step with
          Step.target = Some (
              Target.Heart_rate
              (Target.Heart_rate_value.Zone
                  (Heart_rate.zone_of_int 2))), []})
; "Workout step with speed target" >::
(assert_parses "[speed 25.2-36 km/h]"
  {empty_workout with
    steps =
      Step.Single {empty_single_step with
          Step.target = Some (
              Target.Speed
              (Target.Speed_value.(
                  Range (range_of_pair
                      (Speed.from_kmph 25.2,
                          Speed.from_kmph 36.0)))), []))
; "Workout step with cadence target" >::
(assert_parses "[cadence 95-110 rpm]"
  {empty_workout with
    steps =
      Step.Single {empty_single_step with
          Step.target = Some (
              Target.Cadence
              (Target.Cadence_value.(
                  Range (range_of_pair
                      (Cadence.of_int 95,
                          Cadence.of_int 110)))), []))
; "Workout steps with power target" >::
(assert_parses "[power zone 3; power 200-250 W]"
  {empty_workout with
    steps = Non_empty_list.of_list [
      Step.Single
        {empty_single_step with
          Step.target = Some (
              Target.Power
              (Target.Power_value.Zone

```

```

        (Power.zone_of_int 3)))))

; Step.Single
{empty_single_step with
Step.target = Some (
    Target.Power
    (Target.Power_value.
        Range (range_of_pair
            (Power.Absolute (absolute_of_int 200)),
            Power.Absolute (absolute_of_int 250))))}
    ]})

; "Reorder target range endpoints" >::
(assert_parses "[cadence 100-90]"
{empty_workout with
steps =
    Step.Single
    {empty_single_step with
        Step.target = Some (
            Target.Cadence
            (Target.Cadence_value.
                Range (range_of_pair
                    (Cadence.of_int 90,
                    Cadence.of_int 100)))), []))}

; "Step with both duration and target" >::
(assert_parses "[time 1 min, cadence zone 2]"
{empty_workout with
steps =
    Step.Single
    {empty_single_step with
        Step.duration = Some (
            Condition.Time (Condition.time_of_int 60)
        );
        target = Some (
            Target.Cadence
            (Target.Cadence_value.Zone (Cadence.zone_of_int 2))
        ), [])})
    ]}

let () = run_test_tt_main
(test_list [ lexer_tests; parser_tests ])

```

il2fit/il2fit.cpp

```

#include <algorithm>
#include <experimental/optional>
#include <functional>
#include <iostream>
#include <sstream>
#include <stdexcept>
#include <string>
#include <unordered_map>
#include <utility>

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wignored-qualifiers"

#include <fit_date_time.hpp>
#include <fit_encode.hpp>
#include <fit_file_creator_mesg.hpp>
#include <fit_file_id_mesg.hpp>
#include <fit_workout_mesg.hpp>

```

```

#include <fit_workout_step_msg.hpp>

#pragma GCC diagnostic pop

using std::cerr;
using std::cin;
using std::cout;
using std::endl;
using std::exception;
using std::experimental::bad_optional_access;
using std::experimental::nullopt;
using std::experimental::optional;
using std::function;
using std::getline;
using std::ios;
using std::iostream;
using std::istream;
using std::istringstream;
using std::make_pair;
using std::max;
using std::min;
using std::ostringstream;
using std::out_of_range;
using std::pair;
using std::runtime_error;
using std::string;
using std::stringstream;
using std::unordered_map;
using std::wstring;

#define S(_expr) \
    static_cast<ostringstream&>( \
        ostringstream().flush() << _expr).str()

namespace {

[[noreturn]] void
error(const string& descr = "") {
    throw runtime_error(descr);
}

string
trim(const string& s) {
    string ans = s;
    // Left
    const auto l = find_if_not(ans.begin(), ans.end(), isspace);
    ans.erase(ans.begin(), l);
    // Right
    const auto r = find_if_not(ans.rbegin(), ans.rend(), isspace).base();
    ans.erase(r, ans.end());
    return ans;
}

//-----
// Optional

const auto none = nullopt;

```

```

template <class T>
optional<T>
some(const T& v)
{
    return optional<T>(v);
}

//-----
// Line-based input

optional<string>
line(istream& input)
{
    string ans;
    getline(input, ans, '\n');
    if (input.bad()) {
        error("I/O error");
    }
    if (input.eof() && ans.empty()) {
        return none;
    }
    return some(ans);
}

//-----
// Parse value from input

template <class T>
T
value(istream& input);

template <>
string
value<string>(istream& input)
{
    try {
        return trim(line(input).value());
    } catch (const bad_optional_access&) {
        error("Unexpected end of file");
    }
}

template <>
wstring
value<wstring>(istream& input)
{
    // FIXME: Invalid conversion
    const auto s = value<string>(input);
    return wstring(s.begin(), s.end());
}

template <class T>
T
value(istream& input)
{
    const auto k = value<string>(input);
    T ans;
    istringstream iss(k);
    iss >> ans;
    if (iss.fail()) {
        error("Bad syntax near " + k + ")");
    }
}

```

```

        }
        return ans;
    }

//-----
// Parse value from input and check range

template <class T>
T
value(istream& input, const T& a, const T& b)
{
    const auto p = min(a, b);
    const auto q = max(a, b);
    const T ans = value<T>(input);
    if (ans < p || ans > q) {
        error(S("Value " << ans << " is out of range "
                "[" << p << ", " << q << "]"));
    }
    return ans;
}

template <class T>
T
value(istream& input, const pair<T, T>& range)
{
    return value<T>(input, range.first, range.second);
}

//-----
// Parse enum value from input

template <class T>
T
value(istream& input, const unordered_map<string, T>& table)
{
    const auto token = value<string>(input);
    try {
        return table.at(token);
    } catch (const out_of_range&) {
        error("Invalid enum value \" " + token + "\"");
    }
}

//-----
// Call actions on input tokens

void
match(const string& token,
      const unordered_map<string, function<void()> >& actions)
{
    try {
        actions.at(token)();
    } catch (const out_of_range&) {
        error("Bad token \" " + token + "\"");
    }
}

void
match(istream& input,
      const unordered_map<string, function<void()> >& actions)
{

```

```

        match(value<string>(input), actions);
    }

//-----
// Parse FIT messages from input

template <>
fit::FileCreatorMsg
value<fit::FileCreatorMsg>(istream& input)
{
    fit::FileCreatorMsg ans;

    for (bool done = false; !done;) {
        match(input, {
            { "hardware_version", [&] {
                ans.SetHardwareVersion(value<FIT_UINT8>(input)); } },
            { "software_version", [&] {
                ans.SetSoftwareVersion(value<FIT_UINT16>(input)); } },
            { "end", [&] {
                match(input, {
                    { "file_creator", [&] { done = true; } });
                });
            }
        });
    }

    return ans;
}

template <>
fit::FileIdMsg
value<fit::FileIdMsg>(istream& input)
{
    fit::FileIdMsg ans;

    // Default values
    ans.SetType(FIT_FILE_WORKOUT);
    ans.SetManufacturer(FIT_MANUFACTURER_GARMIN);
    ans.SetProduct(FIT_GARMIN_PRODUCT_EDGE500);
    ans.SetSerialNumber(54321);
    ans.SetTimeCreated(
        fit::DateTime(static_cast<time_t>(1454942443)).GetTimeStamp());

    for (bool done = false; !done;) {
        match(input, {
            { "number", [&] {
                ans.SetNumber(value<FIT_UINT16>(input)); } },
            { "serial_number", [&] {
                ans.SetSerialNumber(value<FIT_UINT32Z>(input)); } },
            { "time_created", [&] {
                ans.SetTimeCreated(value<FIT_DATE_TIME>(input)); } },
            { "end", [&] {
                match(input, {
                    { "file_id", [&] { done = true; } });
                });
            }
        });
    }

    return ans;
}

template <>
```

```

fit::WorkoutMesg
value<fit::WorkoutMesg>(istream& input)
{
    static const unordered_map<string, FIT_SPORT> sports = {
        { "generic" , FIT_SPORT_GENERIC },
        { "running" , FIT_SPORT_RUNNING },
        { "cycling" , FIT_SPORT_CYCLING },
        { "transition" , FIT_SPORT_TRANSITION },
        { "fitness_equipment" , FIT_SPORT_FITNESS_EQUIPMENT },
        { "swimming" , FIT_SPORT_SWIMMING },
        { "basketball" , FIT_SPORT_BASKETBALL },
        { "soccer" , FIT_SPORT_SOCCER },
        { "tennis" , FIT_SPORT_TENNIS },
        { "american_football" , FIT_SPORT_AMERICAN FOOTBALL },
        { "training" , FIT_SPORT_TRAINING },
        { "walking" , FIT_SPORT_WALKING },
        { "cross_country_skiing" , FIT_SPORT_CROSS_COUNTRY_SKIING },
        { "alpine_skiing" , FIT_SPORT_ALPINE_SKIING },
        { "snowboarding" , FIT_SPORT_SNOWBOARDING },
        { "rowing" , FIT_SPORT_ROWING },
        { "mountaineering" , FIT_SPORT_MOUNTAINEERING },
        { "hiking" , FIT_SPORT_HIKING },
        { "multisport" , FIT_SPORT_MULTISPORT },
        { "paddling" , FIT_SPORT_PADDLING },
        { "flying" , FIT_SPORT_FLYING },
        { "e_biking" , FIT_SPORT_E_BIKING },
        { "motorcycling" , FIT_SPORT_MOTORCYCLING },
        { "boating" , FIT_SPORT_BOATING },
        { "driving" , FIT_SPORT_DRIVING },
        { "golf" , FIT_SPORT_GOLF },
        { "hang_gliding" , FIT_SPORT_HANG_GLIDING },
        { "horseback_riding" , FIT_SPORT_HORSEBACK RIDING },
        { "hunting" , FIT_SPORT_HUNTING },
        { "fishing" , FIT_SPORT_FISHING },
        { "inline_skating" , FIT_SPORT_INLINE_SKATING },
        { "rock_climbing" , FIT_SPORT_ROCK_CLIMBING },
        { "sailing" , FIT_SPORT_SAILING },
        { "ice_skating" , FIT_SPORT_ICE_SKATING },
        { "sky_diving" , FIT_SPORT_SKY_DIVING },
        { "snowshoeing" , FIT_SPORT_SNOWSHOEING },
        { "snowmobiling" , FIT_SPORT_SNOWMOBILING },
        { "stand_up_paddleboarding" , FIT_SPORT_STAND_UP_PADDLEBOARDING },
        { "surfing" , FIT_SPORT_SURFING },
        { "wakeboarding" , FIT_SPORT_WAKEBOARDING },
        { "water_skiing" , FIT_SPORT_WATER_SKIING },
        { "kayaking" , FIT_SPORT_KAYAKING },
        { "rafting" , FIT_SPORT_RAFTING },
        { "windsurfing" , FIT_SPORT_WINDSURFING },
        { "kitesurfing" , FIT_SPORT_KITESURFING }
    };
}

fit::WorkoutMesg ans;

// Default values
ans.SetSport(FIT_SPORT_CYCLING);
ans.SetCapabilities(FIT_WORKOUT_CAPABILITIES_INVALID);
ans.SetNumValidSteps(1);

for (bool done = false; !done;) {
    match(input, {
        { "capabilities" , [&] {

```

```

        ans.SetCapabilities(
            value<FIT_WORKOUT_CAPABILITIES>(input)); } },
    { "num_valid_steps", [&] {
        ans.SetNumValidSteps(
            value<FIT_UINT16>(input, 1, 10000)); } },
    { "sport", [&] {
        ans.SetSport(value<FIT_SPORT>(input, sports)); } },
    { "wkt_name", [&] {
        ans.SetWktName(value<FIT_WSTRING>(input)); } },
    { "end", [&] {
        match(input, {
            "workout", [&] { done = true; } });
    } }
));
}

return ans;
}

template <>
fit::WorkoutStepMesg
value<fit::WorkoutStepMesg>(istream& input)
{
    static const unordered_map<string, FIT_INTENSITY> intensities = {
        { "active" , FIT_INTENSITY_ACTIVE },
        { "rest"   , FIT_INTENSITY_REST },
        { "warmup" , FIT_INTENSITY_WARMUP },
        { "cooldown" , FIT_INTENSITY_COOLDOWN }
    };

    static const unordered_map<string, FIT_WKT_STEP_DURATION> duration_types = {
        { "time"           , FIT_WKT_STEP_DURATION_TIME },
        { "distance"       , FIT_WKT_STEP_DURATION_DISTANCE },
        { "hr_less_than"   , FIT_WKT_STEP_DURATION_HR_LESS_THAN },
        { "hr_greater_than" ,
            FIT_WKT_STEP_DURATION_HR_GREATER_THAN
            { "calories" , FIT_WKT_STEP_DURATION_CALORIES },
        },
        { "open"           , FIT_WKT_STEP_DURATION_OPEN },
        { "repeat_until_steps_cmplt" ,
            FIT_WKT_STEP_DURATION_REPEAT_UNTIL_STEPS_CMPLT
            { "repeat_until_time" ,
                FIT_WKT_STEP_DURATION_REPEAT_UNTIL_TIME
                { "repeat_until_distance" ,
                    FIT_WKT_STEP_DURATION_REPEAT_UNTIL_DISTANCE
                    { "repeat_until_calories" ,
                        FIT_WKT_STEP_DURATION_REPEAT_UNTIL_CALORIES
                        { "repeat_until_hr_less_than" ,
                            FIT_WKT_STEP_DURATION_REPEAT_UNTIL_HR_LESS_THAN
                            { "repeat_until_hr_greater_than" ,
                                FIT_WKT_STEP_DURATION_REPEAT_UNTIL_HR_GREATER_THAN
                                { "repeat_until_power_less_than" ,
                                    FIT_WKT_STEP_DURATION_REPEAT_UNTIL_POWER_LESS_THAN
                                    { "repeat_until_power_greater_than" ,
                                        FIT_WKT_STEP_DURATION_REPEAT_UNTIL_POWER_GREATER_THAN
                                        { "power_less_than" ,
                                            FIT_WKT_STEP_DURATION_POWER_LESS_THAN
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    };
}

```

```

        { "power_greater_than" ,
FIT_WKT_STEP_DURATION_POWER_GREATER_THAN , },
        { "repetition_time" ,
FIT_WKT_STEP_DURATION_REPEATITION_TIME , }
};

static const unordered_map<string, FIT_WKT_STEP_TARGET> target_types = {
    { "speed" , FIT_WKT_STEP_TARGET_SPEED },
    { "heart_rate" , FIT_WKT_STEP_TARGET_HEART_RATE },
    { "open" , FIT_WKT_STEP_TARGET_OPEN },
    { "cadence" , FIT_WKT_STEP_TARGET_CADENCE },
    { "power" , FIT_WKT_STEP_TARGET_POWER },
    { "grade" , FIT_WKT_STEP_TARGET_GRADE },
    { "resistance" , FIT_WKT_STEP_TARGET_RESISTANCE }
};

static const pair<FIT_WORKOUT_HR, FIT_WORKOUT_HR> hr_range =
make_pair(0, 355);

static const pair<FIT_WORKOUT_POWER, FIT_WORKOUT_POWER> power_range =
make_pair(0, 11000);

fit::WorkoutStepMesg ans;

// Default values
ans.SetMessageIndex(FIT_MESSAGE_INDEX_INVALID);
ans.SetDurationType(FIT_WKT_STEP_DURATION_OPEN);
ans.SetTargetType(FIT_WKT_STEP_TARGET_OPEN);

for (bool done = false; !done;) {
    match(input, {
        { "custom_target_cadence_high", [&] {
            ans.SetCustomTargetCadenceHigh(
                value<FIT_UINT32>(input)); // rpm
        } },
        { "custom_target_cadence_low", [&] {
            ans.SetCustomTargetCadenceLow(
                value<FIT_UINT32>(input)); // rpm
        } },
        { "custom_target_heart_rate_high", [&] {
            ans.SetCustomTargetHeartRateHigh(
                value(input, hr_range)); // % or bpm
        } },
        { "custom_target_heart_rate_low", [&] {
            ans.SetCustomTargetHeartRateLow(
                value(input, hr_range)); // % or bpm
        } },
        { "custom_target_power_high", [&] {
            ans.SetCustomTargetPowerHigh(
                value(input, power_range)); // % or W
        } },
        { "custom_target_power_low", [&] {
            ans.SetCustomTargetPowerLow(
                value(input, power_range)); // % or W
        } },
        { "custom_target_speed_high", [&] {
            ans.SetCustomTargetSpeedHigh(
                value<FIT_FLOAT32>(input)); // m/s
        } },
        { "custom_target_speed_low", [&] {
            ans.SetCustomTargetSpeedLow(

```

```

        value<FIT_FLOAT32>(input)); // m/s
    } },
{ "custom_target_value_high", [&] {
    ans.SetCustomTargetValueHigh(value<FIT_UINT32>(input));
} },
{ "custom_target_value_low", [&] {
    ans.SetCustomTargetValueLow(value<FIT_UINT32>(input));
} },
{ "duration_calories", [&] {
    // TODO: restrict by range?
    ans.SetDurationCalories(
        value<FIT_UINT32>(input)); // kcal
} },
{ "duration_distance", [&] {
    // TODO: restrict by range?
    ans.SetDurationDistance(value<FIT_FLOAT32>(input)); // m
} },
{ "duration_hr", [&] {
    ans.SetDurationHr(value(input, hr_range)); // % or bpm
} },
{ "duration_power", [&] {
    ans.SetDurationPower(
        value(input, power_range)); // % or W
} },
{ "duration_step", [&] {
    ans.SetDurationStep(value<FIT_UINT32>(input));
} },
{ "duration_time", [&] {
    // TODO: restrict by range?
    ans.SetDurationTime(value<FIT_FLOAT32>(input)); // s
} },
{ "duration_type", [&] {
    ans.SetDurationType(value(input, duration_types));
} },
{ "duration_value", [&] {
    ans.SetDurationValue(value<FIT_UINT32>(input));
} },
{ "intensity", [&] {
    ans.SetIntensity(value(input, intensities));
} },
{ "message_index", [&] {
    ans.SetMessageIndex(
        value<FIT_MESSAGE_INDEX>(input, 0, 0FFF));
} },
{ "repeat_calories", [&] {
    // TODO: restrict by range?
    ans.SetRepeatCalories(value<FIT_UINT32>(input)); // kcal
} },
{ "repeat_distance", [&] {
    // TODO: restrict by range?
    ans.SetRepeatDistance(value<FIT_FLOAT32>(input)); // m
} },
{ "repeat_hr", [&] {
    ans.SetRepeatHr(value(input, hr_range)); // % or bpm
} },
{ "repeat_power", [&] {
    ans.SetRepeatPower(value(input, power_range)); // % or W
} },
{ "repeat_steps", [&] {
    ans.SetRepeatSteps(value<FIT_UINT32>(input, 1, 1000));
} },

```

```

        { "repeat_time", [&] {
            // TODO: restrict by range?
            ans.SetRepeatTime(value<FIT_FLOAT32>(input)); // s
        } },
        { "target_hr_zone", [&] {
            // HR Zone (1-5); Custom = 0;
            ans.SetTargetHrZone(value<FIT_UINT32>(input, 0, 5));
        } },
        { "target_power_zone", [&] {
            // Power Zone (1-7); Custom = 0;
            ans.SetTargetPowerZone(value<FIT_UINT32>(input, 0, 7));
        } },
        { "target_type", [&] {
            ans.SetTargetType(value(input, target_types));
        } },
        { "target_value", [&] {
            ans.SetTargetValue(value<FIT_UINT32>(input));
        } },
        { "wkt_step_name", [&] {
            ans.SetWktStepName(value<FIT_WSTRING>(input));
        } },
        { "end", [&] {
            match(input, {
                { "workout_step", [&] { done = true; } })
            });
        } }
    });

    return ans;
}

void
il2fit(istream& input, ostream& output)
{
    fit::Encode encode;

    encode.Open(output);

    bool empty = true;

    while (const auto lopt = line(input)) {
        bool eof = false;

        match(lopt.value(), {
            { "begin", [&] {
                match(input, {
                    { "file_creator", [&] {

encode.Write(value<fit::FileCreatorMesg>(input));
empty = false; }}),
                    { "file_id", [&] {

encode.Write(value<fit:: fileIdMesg>(input));
empty = false; }}),
                    { "workout", [&] {

encode.Write(value<fit::WorkoutMesg>(input));
empty = false; }}),
                    { "workout_step", [&] {

encode.Write(value<fit::WorkoutStepMesg>(input));

```

```

                empty = false; } }
            } );
        } },  

        { "EOF", [&] { eof = true; } }
    ) );

    if (eof) {
        break;
    }
}

if (empty) {
    error("No messages in the FIT file");
}

if (!encode.Close()) {
    error("FIT encoder failed");
}
}

} // namespace

//-----
// Main

#ifndef _WITH_TESTS

int main()
{
    stringstream output(ios::out | ios::binary);

    try {
        il2fit(cin, output);
    } catch (const exception& exn) {
        cerr << exn.what() << endl;
        return 1;
    }

    cout << output.str();

    return 0;
}

#else // _WITH_TESTS

#define CATCH_CONFIG_MAIN
#include <catch.hpp>

//-----
// Cases for trim()

TEST_CASE("Trim empty string", "[trim]")
{
    CHECK(trim("") == "");
}

TEST_CASE("Trim string, empty ans", "[trim]")
{
    CHECK(trim(" \t\t ") == "");
}

```

```

TEST_CASE("Trim string left", "[trim]")
{
    CHECK(trim("\tabc") == "abc");
}

TEST_CASE("Trim string right", "[trim]")
{
    CHECK(trim("abc\t ") == "abc");
}

TEST_CASE("Trim left and right", "[trim]")
{
    CHECK(trim("\tabc\t ") == "abc");
}

//-----
// Cases for line()

TEST_CASE("EOF on empty input", "[line]")
{
    istringstream input;
    CHECK(line(input) == none);
}

TEST_CASE("Empty line", "[line]")
{
    istringstream input("\n");
    CHECK(line(input) == some(string(""))));
    CHECK(line(input) == none);
}

TEST_CASE("Incomplete line", "[line]")
{
    istringstream input("abc");
    CHECK(line(input) == some(string("abc")));
    CHECK(line(input) == none);
}

TEST_CASE("Single complete line", "[line]")
{
    istringstream input("abc\n");
    CHECK(line(input) == some(string("abc")));
    CHECK(line(input) == none);
}

TEST_CASE("Multiple lines", "[line]")
{
    istringstream input("abc\ndef\n");
    CHECK(line(input) == some(string("abc")));
    CHECK(line(input) == some(string("def")));
    CHECK(line(input) == none);
}

//-----
// Cases for value()

TEST_CASE("Error on empty input", "[value]")
{
    istringstream input;
    CHECK_THROWS_AS(value<string>(input), runtime_error);
}

```

```

TEST_CASE("Parse string value", "[value]")
{
    istringstream input("abc\n");
    CHECK(value<string>(input) == "abc");
}

TEST_CASE("Parse string value with padding", "[value]")
{
    istringstream input(" abc\t\t\n");
    CHECK(value<string>(input) == "abc");
}

TEST_CASE("Parse int value", "[value]")
{
    istringstream input("123\n");
    CHECK(value<int>(input) == 123);
}

TEST_CASE("Parse double value", "[value]")
{
    istringstream input("45.6\n");
    CHECK(value<double>(input) == Approx(45.6));
}

//-----
// Cases for value() with range

TEST_CASE("Parse value with range", "[value]")
{
    istringstream input("55\n");
    CHECK(value<int>(input, 0, 100) == 55);
}

TEST_CASE("Parse value out of range", "[value]")
{
    istringstream input("55\n");
    CHECK_THROWS_AS(value<int>(input, 0, 10), runtime_error);
}

TEST_CASE("Parse value, range endpoints out of order", "[value]")
{
    istringstream input("55\n");
    CHECK(value<int>(input, 100, 0) == 55);
}

TEST_CASE("Parse value, range pair", "[value]")
{
    istringstream input("55\n55\n");
    CHECK(value<int>(input, 0, 100) ==
          value<int>(input, make_pair(0, 100)));
}

//-----
// Cases for value() for enumerations

TEST_CASE("Parse enum value", "[value]")
{
    istringstream input("xyz\n");
    CHECK(value<int>(input, {
        { "xyz", 10 },

```

```

        { "abc", 12 }
    )) == 10);
}

TEST_CASE("Parse enum value, invalid token", "[value]")
{
    istringstream input("xyz\n");
    CHECK_THROWS_AS(value<int>(input, {
        { "abc", 20 },
        { "def", 30 }
    }), runtime_error);
}

//-----
// Cases for match()

TEST_CASE("Empty match table", "[match]")
{
    istringstream input("xyz\n\nabc\n");
    static const unordered_map<string, function<void()>> empty;
    CHECK_THROWS_AS(match(input, empty), runtime_error);
    CHECK_THROWS_AS(match(input, empty), runtime_error);
    CHECK_THROWS_AS(match(input, empty), runtime_error);
}

TEST_CASE("Match table", "[match]")
{
    istringstream input("xyz\n\nabc\n");
    bool ok = false;
    match(input, { { "xyz", [&] { ok = true; } } });
    CHECK(ok);
}

//-----
// Cases for value<fit::FileCreatorMesg>

TEST_CASE("Valid file_creator", "[file_creator][value]")
{
    istringstream input(
        "hardware_version\n"
        "150\n"
        "software_version\n"
        "330\n"
        "end\n"
        "file_creator\n"
    );
    CHECK_NOTHROW(value<fit::FileCreatorMesg>(input));
}

//-----
// Cases for value<fit:: fileIdMesg>

TEST_CASE("Valid file_id", "[file_id][value]")
{
    istringstream input(
        "time_created\n"
        "1457736704\n"
        "serial_number\n"
        "2501\n"
        "number\n"
        "5\n"
    );
}

```

```

        "end\n"
        "file_id\n"
    );
CHECK_NOTHROW(value<fit::FileIdMesg>(input));
}

//-----
// Cases for value<fit::WorkoutMesg>

TEST_CASE("Valid workout", "[value] [workout]")
{
    istringstream input(
        "wkt_name\n"
        "Tempo\n"
        "sport\n"
        "fishing\n"
        "num_valid_steps\n"
        "15\n"
        "capabilities\n"
        "31\n"
        "end\n"
        "workout\n"
    );
    CHECK_NOTHROW(value<fit::WorkoutMesg>(input));
}

//-----
// Cases for value<fit::WorkoutStepMesg>

TEST_CASE("Valid workout_step", "[value] [workout_step]")
{
    istringstream input(
        "message_index\n"
        "3\n"
        "wkt_step_name\n"
        "Intro\n"
        "intensity\n"
        "warmup\n"
        "duration_time\n"
        "31.5\n"
        "target_value\n"
        "0\n"
        "custom_target_heart_rate_low\n"
        "50\n"
        "custom_target_heart_rate_high\n"
        "270\n"
        "end\n"
        "workout_step\n"
    );
    CHECK_NOTHROW(value<fit::WorkoutStepMesg>(input));
}

//-----
// Cases for il2fit

TEST_CASE("Fail on empty IL input", "[il2fit]")
{
    istringstream input;
    stringstream output;
    CHECK_THROWS_AS(il2fit(input, output), runtime_error);
}

```

```

TEST_CASE("Invalid IL input", "[il2fit]")
{
    istringstream input(
        "begin\n"
        "nonsense\n"
        "end\n"
        "nonsense\n"
    );
    stringstream output;
    CHECK_THROWS_AS(il2fit(input, output), runtime_error);
}

TEST_CASE("Valid IL input", "[il2fit]")
{
    istringstream input(
        "begin\n"
        "file_id\n"
        "end\n"
        "file_id\n"
        "begin\n"
        "file_creator\n"
        "end\n"
        "file_creator\n"
        "begin\n"
        "workout\n"
        "end\n"
        "workout\n"
        "begin\n"
        "workout_step\n"
        "end\n"
        "workout_step\n"
    );
    stringstream output;
    CHECK_NO_THROW(il2fit(input, output));
    CHECK_FALSE(output.str().empty());
}

#endif // _WITH_TESTS

```

wrk2fit

```

#!/bin/sh

D=$(dirname $0)
$D/wrk2il "$@" | $D/il2fit

```

wrk2fit-dnd

```

#!/usr/bin/env tclsh

package require Tk 8.5
package require tkdnd 2.6

wm title . "wrk2fit"
wm minsize . 120 120

set defaultText "WRK -> FIT"

```

```

set fitFileTypes {{"FIT Workout Files" ".fit"} {"All Files" "*"}}
set wrkFileTypes {{"Wokrout Description Files" ".wrk"} {"All Files" "*"}}

ttk::label .dropLabel -text $defaultText \
    -anchor center -justify center -wraplength 120

grid rowconfigure . 0 -weight 1
grid columnconfigure . 0 -weight 1

grid .dropLabel -row 0 -column 0 -sticky nesw -padx 5 -pady 5

tkdnd::drop_target register .dropLabel {DND_Files DND_Text}

proc onError {message} {
    global defaultText
    .dropLabel configure -background #dc322f -text $message
    after 1000 {
        .dropLabel configure -background {} -text $defaultText
    }
}

proc writeFile {path data} {
    # TODO: handle errors
    set fd [open $path {BINARY CREAT WRONLY}]
    puts -nonewline $fd $data
    close $fd
}

proc handleText {wrk} {
    # TODO
}

proc handleFile {wrkFile} {
    global fitFileTypes
    try {
        set fit [exec ./wrk2il < $wrkFile | ./il2fit]
        set path [tk_getSaveFile -filetypes $fitFileTypes]
        writeFile $path $fit
    } trap CHILDSTATUS {- opts} {
        onError "Bad WRK"
    }
}

bind .dropLabel <Double-1> {
    set filePath [tk_getOpenFile -filetypes $wrkFileTypes]
    if {$filePath ne ""} {
        handleFile $filePath
    } else {
        onError "No file selected"
    }
}

bind .dropLabel <><Drop:DND_Files>> {
    handleFile [lindex %D 0]
}

bind .dropLabel <><Drop:DND_Text>> {
    handleText %D
}

```

ПРИЛОЖЕНИЕ Б

Исходный код основных модулей веб-службы

`web/src/wrked_port.erl`

```
-module(wrked_port).

%% API
-export([il2fit/1, wrk2fit/1, wrk2fit/3, wrk2il/2, wrk2il/4]).

%%%=====
%%% API
%%%=====

%%-----%
%% @doc
%% @end
%%-----%
-spec il2fit(iodata()) -> {ok, iodata()} | error | timeout.

il2fit(I1) ->
    exec(
        _Path = application:get_env(wrked, il2fit_path, "bin/il2fit"),
        _Args = [],
        _Body = [I1, <<"EOF\n">>]
    ).

%%-----%
%% @doc
%% @end
%%-----%
-spec wrk2fit(iodata()) -> {ok, iodata()} | error | timeout.

wrk2fit(Wrk) ->
    wrk2fit(Wrk, _Name = undefined, _Sport = undefined).

%%-----%
%% @doc
%% @end
%%-----%
-spec wrk2fit(iodata(),
               binary() | undefined,
               binary() | undefined) -> {ok, iodata()} | error | timeout.

wrk2fit(Wrk, Name, Sport) ->
    case wrk2il(Wrk, translate, Name, Sport) of
        {ok, I1} -> il2fit(I1);
        error      -> error;
        timeout    -> timeout
    end.

%%-----%
%% @doc
%% @end
%%-----%
-spec wrk2il(iodata(), translate | minimize) ->
    {ok, iodata()} | error | timeout.
```

```

wrk2il(Wrk, Mode) ->
    wrk2il(Wrk, Mode, _Name = undefined, _Sport = undefined).

%%-----
%% @doc
%% @end
%%-----
-spec wrk2il(iodata(), translate | minimize,
             binary() | undefined,
             binary() | undefined) -> {ok, iodata()} | error | timeout.

wrk2il(Wrk, Mode, Name, Sport) ->
    exec(
        _Path = application:get_env(wrked, wrk2il_path, "bin/wrk2il"),
        _Args = lists:flatmap(
            fun({_K, _V = undefined}) -> [],
           ({_K, V}) -> [K, V] end,
            [{<<"-name">>, Name},
             {<<"-sport">>, Sport},
             {<<"-mode">>,
              case Mode of
                  translate -> <<"tr">>;
                  minimize -> <<"min">>
              end} ]),
        Body = [Wrk, <<"EOF">>]
    ).

%%%=====
%%% Internal functions
%%%=====

%%-----
%% @private
%% @doc
%% @end
%%-----
-spec exec(file:name(), [string() | binary()], iodata()) ->
    {ok, iodata()} | error | timeout.

exec(Path, Args, Body) ->
    Port = open_port({spawn_executable, Path},
                     [{args, Args}, binary, exit_status]),
    port_command(Port, Body),
    receive_loop(Port, _Ans = <<>>).

%%-----
%% @private
%% @doc
%% @end
%%-----
-spec receive_loop(port(), iodata()) -> {ok, iodata()} | error | timeout.

receive_loop(Port, Ans) ->
    receive
        {Port, {data, Data}} -> receive_loop(Port, [Ans, Data]);
        {Port, {exit_status, 0}} -> {ok, Ans};
        {Port, {exit_status, _Code}} -> error
    after 1000 -> port_close(Port), timeout
    end.

```

web/src/wrked_app.erl

```
-module(wrked_app).  
-behaviour(application).  
  
%% Application callbacks  
-export([start/2, stop/1]).  
  
%%===== %% Application callbacks %%=====  
  
start(_StartType, _StartArgs) ->  
    Constr =  
        [ { sport,  
            fun(Sport) ->  
                lists:member(  
                    Sport, [ <<"cycling">>, <<"running">>,  
                            <<"swimming">>, <<"walking">> ])  
            end }  
        ],  
    Paths = [ { <</workouts/:sport/:name/:wrk">>, Constr, wrked_handler2, [] },  
              { <</workouts/:sport/:wrk">>, Constr, wrked_handler2, [] },  
              { <</workouts/:wrk">>, wrked_handler2, [] },  
              { <<"/">>, cowboy_static, {priv_file, wrked, <<"index.html">>} },  
              { <<"/[...]">>, cowboy_static,  
                  {priv_dir, wrked, <<"">>, [{mimetypes, cow_mimetypes, all}]} }  
            ],  
    Host = {'_', Paths},  
    Dispatch = cowboy_router:compile([Host]),  
    Addr = application:get_env(wrked, addr, {127,0,0,1}),  
    Port = application:get_env(wrked, port, 8080),  
    {ok, _Pid} =  
        cowboy:start_http(wrked_http, 100,  
                          [{ip, Addr}, {port, Port}],  
                          [{compress, true}, {env, [{dispatch, Dispatch}]}]),  
    wrked_sup:start_link().  
  
stop(_State) ->  
    ok.
```

web/src/wrked_handler2.erl

```
-module(wrked_handler2).  
-behaviour(common_handler).  
  
%% Common handler callbacks  
-export([init/2]).  
  
%% REST handler callbacks  
-export([content_types_provided/2, last_modified/2,  
        malformed_request/2]).  
  
%% API  
-export([to_fit/2]).  
  
%%%===== %% Common handler callbacks %%=====
```

```

init(Req, _Opts) ->
    {cowboy_rest, Req, _State = undefined}.

%%=====
%% REST handler callbacks
%%=====

content_types_provided(Req, State) ->
    Result = [{{"application"}, {"vnd.ant.fit"}, '*'}, to_fit],
    {Result, Req, State}.

last_modified(Req, State) ->
    Result = {{2016,05,10}, {22,24,43}},
    {Result, Req, State}.

malformed_request(Req, State) ->
    Name = cowboy_req:binding(name, Req),
    Sport = cowboy_req:binding(sport, Req),
    Wrk = cowboy_req:binding(wrk, Req),
    case wrked_port:wrk2fit(Wrk, Name, Sport) of
        {ok, Fit} ->
            Req2 = cowboy_req:set_resp_header(
                {"content-disposition"}, [{"attachment; filename="}, filename(Name, Sport)], Req),
            {false, Req2, _State = Fit};
        error -> {true, Req, State}
    end.

%%=====
%% API
%%=====

to_fit(Req, State = Fit) -> {Fit, Req, State}.

%%=====
%% Internal functions
%%=====

filename(Name, Sport) ->
    [<<"workout->>,
     case Sport of
         undefined -> <>>;
         _SomeSport -> [Sport, $-]
     end,
     case Name of
         undefined ->
             {{Y, M, D}, {H, N, _S}} = erlang:universaltime(),
             io_lib:format("~4..0B~2..0B~2..0BT~2..0B~2..0BZ",
                           [Y, M, D, H, N]);
         _SomeName -> Name
     end,
     <<".fit">>].

```