МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
имени первого Президента России Б.Н. Ельцина

ИНСТИТУТ ЕСТЕСТВЕННЫХ НАУК И МАТЕМАТИКИ
Департамент математики, механики и компьютерных наук

Реализация поддержки задач в формате codeforces в образовательной платформе Ulearn

Направление подготовки 02.03.02 «Фундаментальная информатика и информационные технологии»

Директор департамента:
к. ф.-м.н., доц Е.С. Пьянзина

_____

Нормоконтролер:
О.А. Суслова

_____

Выпускная квалификационная
работа бакалавра
Логинова Александра
Владиславовича

_____

Научный руководитель:
ст.преп., П. В. Егоров

_____

асс., А. Н. Федоров

_____

Научный соруководитель:
канд. физ.-мат. наук, доц.,
О. В. Расин

_____

Екатеринбург
2021

Реферат

Выпускная квалификационная работа состоит из введения, 4 разделов и 16 подразделов, заключения, списка используемых источников состоящего из 7 источников. Работа изложена на 36 листах печатного текста, содержит 22 листинга.

Ключевые слова: POLYGON, TIMUS, CODEFORCES, ОБРАЗОВАТЕЛЬНЫЕ КУРСЫ, ULEARN, DOCKER, ОЛИМПИАДНЫЕ ЗАДАЧИ, АВТОПРОВЕРКА.

Целью дипломной работы является реализация поддержки задач в формате Codeforces в образовательной платформе Ulearn: возможность размещения таких курсов, автопроверки решений по задачам.

Методы исследования: анализ, синтез, наблюдение, сравнение. Источник данных - документация формата задач, статьи авторов, платформы выполняющие подобную задачу.

В ходе работы было разработано и внедрено в сервис Ulearn в компании Скб Контур решение позволяющее поддерживать задачи в формате Codeforces.

Содержание

Глоссарий

В настоящей работе применяются следующие термины

Ulearn — платформа с интерактивными онлайн-курсами по программированию

Codeforces — проект, объединяющий людей, которые интересуются и участвуют в соревнованиях по программированию. Является как социальной сетью, посвященной программированию и соревнованиям так и площадкой, где регулярно проводятся соревнования

Timus Online Judge — крупнейший в России архив задач по программированию с автоматической проверяющей системой.

Формат задач Codeforces — принятый стандарт внутреннего описания олимпиадных задач. Распространяется также на Timus Online Judge

Polygon — платформа для создания задач по программированию

Docker — программное обеспечение с открытым исходным кодом, применяемое для разработки, тестирования, доставки и запуска веб-приложений в средах с поддержкой контейнеризации

Чеккер — программа, выполняющая задачу проверки корректности ответа

Don't repeat yourself — принцип разработки программного обеспечения, нацеленный на снижение повторения информации различного рода, особенно в системах со множеством слоёв абстрагирования.

Single-responsibility principle — принцип ООП, обозначающий, что каждый объект должен иметь одну ответственность и эта ответственность должна быть полностью инкапсулирована в класс. Все его поведения должны быть направлены исключительно на обеспечение этой ответственности.

Обозначения и сокращения

В настоящей работе применяются следующие обозначения и сокращения

DRY — Don't repeat yourself

SRP — Single-responsibility principle

Введение

Актуальность выбранной темы дипломной работы обусловлена тем, что авторам курсов был необходим инструмент для формирования и публикации своего набора задач для студентов университетов, школьников и всех желающих изучить материал курса с выбором языка из множества допустимых. Преимущество системы Ulearn перед такими системами как Codeforces и Timus заключается в возможности:

— переиспользовать готовые задачи, подготовленные ранее в polygon, а не переводить в формат Ulearn;

— не изучать автору формат задач Ulearn, если он знаком с форматом Codeforces;

— проводить проверку кода студента на понятность, стиль принятый для текущего языка;

— регулировать стоимость каждого задания в баллах;

— размещать теорию и практику курса в одном месте.

Целью дипломной работы является реализация поддержки задач в формате Codeforces в образовательной платформе Ulearn: возможность размещения таких курсов, автопроверки решений по задачам.

Для достижения поставленной цели необходимо решить следующие задачи:

— проанализировать процесс подготовки задач, а так же структуру и формат задач, публикуемых в системах Codeforces/Timus;

— изучить текущую инфраструктуру и способ формирования курсов в образовательной системе Ulearn;

— спроектировать решение в рамках принятых в системе Ulearn паттернов;

— реализовать это решение;

— проверить работоспособность и удобство использования для авторов курсов.

Основная часть

1 Анализ процесса подготовки задач и их формата

В данном разделе анализируется процесс подготовки задач, а так же структура и формат задач, публикуемых в системах Codeforces/Timus. Процесс подготовки каждой задачи можно разбить на следующие этапы:

а) формирование легенды (идеи для развёрнутого условия задачи);

б) написание условия, подготовка правильных, неверных; и неэффективных решений задачи.

в) написание чекеров и валидаторов, разработка системы тестов.

Для нашей работы наибольший интерес представляют шаги подготовки правильных, неверных и неэффективных решений, а так же написание чекеров, валидатором и системы тестов.

Изучим эти пункты основываясь на работе "Система подготовки олимпиадных задач по программированию в УрГУ"(Алексей Самсонов, Александр Ипатов).

1.1 Авторские решения

Автор курса должен подготовить по своей задаче набор различных решений: правильных, неверных и неэффективных. Желательно написать хотя бы одно верное решение на языке Java или C#, поскольку часто решения, написанные на этих языках, работают медленнее их точных аналогов на C++, что следует учитывать при выставлении ограничений по времени.

В задачах, где есть простое, но асимптотически медленное решение, и сложное эффективное решение, которое необходимо придумать и реализовать учащимся данного курса, автор должен определиться, решения с какой асимптотической сложностью должны засчитываться как «верные», а с какой — нет.

Например, все решения не хуже $O(n^2 \log n)$ должны засчитываться, а все решения за $O(n^3)$ — нет. Следует реализовать верное решение и асимптотически медленное решение, ускоренное с помощью неасимптотических оптимизаций. Ограничения на размер входных данных следует подбирать

таким образом, чтобы неэффективное решение работало дольше эффективного не менее чем в 8-10 раз. При этом ограничение по времени должно превышать время работы верного решения на максимальном тесте в 2-3 раза.

Отдельное внимание следует уделить формально неверным решениям, которые, однако, могут находить правильный ответ для большинства входных данных. Во многих задачах хороший результат дают случайные алгоритмы. Команда разработчиков должна реализовать такие решения и проверить, что они не проходят некоторые из подготовленных тестов.

1.2 Чекеры

Чекеры необходимы в тех ситуациях, когда ответ на задачу для некоторых входных данных неоднозначен. В частности, чекеры активно используются в задачах, где помимо ответа нужно выдать сертификат — информацию, по которой легко убедиться в корректности такого ответа.

Например, в задаче на поиск минимума функции ответом служит искомый минимум, а сертификатом — аргумент функции, при котором этот минимум достигается; в задаче на поиск кратчайшего расстояния между двумя вершинами графа ответом служит длина кратчайшего пути между вершинами, а сертификатом — сам этот путь.

Иногда требование выдачи сертификата усложняет решение задачи. Однако это требование позволяет подготовить задачу надёжнее и сделать процесс подготовки более удобным: с помощью чекера можно удостовериться в том, что авторское решение находит корректный ответ; в противном случае, чекер выдаёт информацию о том, где именно в ответе содержится ошибка, что позволяет быстрее обнаружить ошибку в эталонном решении.

Обратите внимание, что хотя чекер не решает задачу самостоятельно, за счёт сертификата он может проверить, какой из двух ответов лучше. Также стоит отметить, что если авторское решение выдаёт некорректный ответ на какой-либо тест, то при проверке этого решения чекер выдаст вердикт «неправильный ответ», даже если ответ, выданный авторским решением, будет в точности совпадать с ответом в наборе тестов!

1.3 Тесты

Процесс подготовки тестов к любой задаче начинается с создания валидатора — программы, проверяющей корректность входных данных. Валидатор проверяет, что формат файла со входными данными в точности соответствует описанному в условии: везде в файле стоит нужное количество пробелов и переводов строк, в конце файла нет лишней информации, все данные удовлетворяют указанным ограничениям. В валидаторах иногда проверяются и более сложные ограничения — например, связность заданного в файле графа или отсутствие общих точек у двух геометрических фигур.

Автор задачи старается разработать максимально полный набор тестов. Он должен включать множество небольших тестов на разные случаи, ответы на которые легко проверить вручную, тесты на крайние случаи, большие тесты, сгенерированные по шаблону, а также достаточное количество случайных тестов разного размера. Также следует убедиться в том, что асимптотически медленные, эвристические и случайные решения не проходят какие-либо из подготовленных тестов. Для этого нужно либо специально готовить тесты против таких решений, либо воспользоваться техникой стресс-тестирования.

Для проведения стресс-тестирования пишется генератор случайных тестов и чекер. Далее генерируется случайный тест, на нём запускаются два решения (обычно — эталонное и неверное), полученные ответы сравниваются. Если ответы расходятся, то тест против неверного решения готов. Если же ответы совпадают, генерируется новый случайный тест, и так до тех пор, пока не обнаружится расхождение. Стресс-тестирование позволяет находить неочевидные тесты против эвристик, увеличивает надёжность подготовки задачи: если в задаче требуется выдать сертификат, можно запускать стресс-тестирование эталонного решения и проверять, что оно пройдёт все сгенерированные тесты. Интересной техникой подготовки тестов является стресс-тестирование эталонных решений с малыми изменениями. Это позволяет быстро строить сложные тесты на крайние случаи и лучше исследовать задачу.

1.4 Разработка задачи

Для разработки задачи используется система polygon.codeforces.com

Polygon поддерживает весь цикл разработки:

а) написание постановки задачи;

б) подготовка тестовых данных (поддерживаются генераторы);

в) модельные решения (в том числе правильные и заведомо неправильные);

г) автоматическая проверка.

Результатом разработки задачи в данном сервисе является архив, содержимое которого представлено в Листинге 1.1

Листинг 1 — Содержимое архива задачи

```
files
    /...
scripts
    /...
solutions
    /...
statements
    /.html
        /russian
            /problem.html
            /problem-statements.css
    /russian
        /problem.tex
        /problem-properties.json
statement-sections
    /...
tests
    /01
    /01.a
    /02
```

```
    /02.a
    /...
check.cpp
check.exe
problem.xml
```

Наибольший интерес для нашей работы представляют следующие пункты:

— директория solutions, в которой находятся авторские решения;

— директория statements, в которой находятся постановка задачи: html-разметка, pdf-файл, latex-разметка на которых находится легенда задачи, условия, примеры входных и выходных данных и тд;

— директория tests, в которой находятся входные данные тестов и их ответы;

— файл check.cpp;

— файл problem.xml.

Разберем некоторые из этих пункты подробнее.

Файл problem.xml представляет интерес благодаря тому, что в нем мы можем найти информацию, которая соотносит авторские решения и их статус: accepted, main, wrong-answer и т.д.

Помимо статуса решения есть так же информация об языке, на котором оно написано и о версии используемого компилятора.

Пример данного файла представлен в листинге 1.2.

Листинг 2 — Пример информации об авторских решениях

```
<solutions>
    <solution tag="accepted">
        <source
           path="solutions/AGProgressionCpp.cpptype="cpp.g++14"/>
        <binary
           path="solutions/AGProgressionCpp.exetype="exe.win32"/>
    </solution>
    <solution tag="wrong-answer">
```

```
7         <source path="solutions/Formula.cs" type="csharp.mono"/>
8         <binary path="solutions/Formula.exe" type="exe.win32"/>
9     </solution>
10    <solution tag="main">
11        <source path="solutions/Main.cs" type="csharp.mono"/>
12        <binary path="solutions/Main.exe" type="exe.win32"/>
13    </solution>
14 </solutions>
```

Файл check.cpp является тем самым чекером для текущей задачи, описанный в пункте 1.1.2

Директория statements как уже было описано выше содержит визуальное представление данной задачи. Директорий представлена в листинге 1.3.

Листинг 3 — Пример содержимого директории statements

```
1 .html
2     /russian
3         /problem.html
4         /problem-statements.css
5 russian
6     /problem.tex
7     /problem-properties.json
```

1.5 Результат анализа

По итогам проведенного исследования процесса подготовки задач и их формата, были изучены шаги и главные тонкости составления задач, а так же выделены проблемы и важные места, которые необходимо будет учесть в реализации собственной системы:

а) вся ответственность за написание тестов и различных авторских решений остается на создателе курса. Система Ulearn будет предоставлять возможность размещения задач и проверки решений;

б) Опираясь на то, что в каталоге задачи всегда присутствует информация о визуальном представлении для пользователя, то система должна автоматически генерировать его без дополнительных действий со стороны автора курса;

2 Инфраструктура Ulearn

В этом разделе описана текущая инфраструктура в образовательной системе Ulearn, а именно

а) как создаются слайды курса;
б) как происходит их обработка;
в) как осуществляется проверка отправленного студентом решения.

2.1 Структура курса

Курс в системе Ulearn состоит из множества слайдов, каждый из которых может быть одним из трех типов:

а) Lesson — теоретический слайд-урок;
б) Quiz — тест с вопросами разных типов;
в) Exercise — задача на программирование.

Задачи на программирование делятся на 3 типа:

а) SingleFileExercise — задача на языке C# из одного файла, где проверяется, что вывод равен константе;
б) CsProjectExercise — задача на языке C#;
в) UniversalExercise — задача на языке, который определен конфигурацией docker-образа под эту задачу.

Каждому слайду соответствует xml-файл, который задает тип слайда, информацию, которая будет показана при его отображении и прочие данные.

Типизация объектов в xml-файле задаётся в соответствующей xsd-схеме.

Рассмотрим несколько конфигурации нескольких типов слайдов. Первый тип представлен в листинге 1.4.

Листинг 4 — Пример xml-файла lesson-слайда

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <slide xmlns="https://ulearn.me/schema/v2" title="Theory"
    id="7ac1f6d9-41d6-4309-8eda-8786ccc3f990">
```

```
    <markdown>
        // Slide data
    </markdown>
</slide>
```

В данной xml-разметке находится главный тег — slide у которого присутствуют обязательные атрибуты: title (название данного слайда) и id (уникальный идентификатор слайда). Внутри тега slide находится тег markdown в который можно размещаться всю информационную нагрузку слайда: текст, ссылки и тд.

Листинг 5 — Пример xml-файла exercise-слайда

```
<?xml version="1.0"?>
<slide.exercise id="717f67cc-f53f-45e3-92f1-8fa20572c6d5"
   title="Practice" xmlns="https://ulearn.me/schema/v2">
  <scoring group="exercise" passedTestsScore="2"
     codeReviewScore="8"/>
  <markdown>
    // Task information
  </markdown>
  <exercise.universal exerciseDirName="src/01_fahrenheit"
     noStudentZip="true">
    <hideSolutions>true</hideSolutions>
    <hideExpectedOutput>true</hideExpectedOutput>
    <userCodeFile>task.js</userCodeFile>
    <includePathForChecker>../TestsRunner</includePathForChecker>
    <dockerImageName>js-sandbox</dockerImageName>
    <run>node docker-test-runner.js</run>
    <region>Task</region>
  </exercise.universal>
</slide.exercise>
```

В разметке слайда типа exercise, который представлен в листинге 1.5, различия начинаются с главного тега: вместо slide в данном типе находится slide.exercise.

Внутри slide.exercise, помимо уже известного markdown, присутствуют блоки: scoring — тег, отвечающий за конфигурацию расчета баллов для этой задачи, exercise.universal — тег, вместе со своими аттрибутами и вложенными тегами, отвечают за конфигурацию самого задания, механизма проверки и тд.

2.2 Обработка конфигурации слайдов

За обработку конфигураций слайдов ответственны сущности реализующие следующий контракт, представленный на листинге 1.6:

Листинг 6 — Контракт конфигуратора слайдов

```
1 public interface ISlide
2     {
3         Guid Id { get; }
4         string Title { get; }
5         SlideBlock[] Blocks { get; }
6 
7         void Validate(SlideLoadingContext context);
8         void BuildUp(SlideLoadingContext context);
9     }
```

Метод Validate отвечают за проверку корректности заполнения конфигурации. А BuildUp — за логику построения по данным конфигурации html-разметки.

2.3 Обработка конфигурации блоков

За обработку конфигураций блоков внутри слайда ответственны сущности, которые наследуются абстрактного класса SlideBlock. Главные пункты этого класса соответствуют конфигураторам слайдов: методы Validate и BuildUp.

2.4 Проверка решения студента

После того как пользователь ввел своё решение в редактор сервиса и нажал кнопку "Отправить"вызывается метод RunSolution контроллера ExerciseController, в который передаётся информация о курсе, слайде и отправленном решении. Его структура описана в листинге 1.7.

Листинг 7 — Сигнатура метода RunSolution

```
public async Task<ActionResult<RunSolutionResponse>>
   RunSolution(
            [FromRoute] Course course,
            [FromRoute] Guid slideId,
            [FromBody] RunSolutionParameters parameters,
            [FromQuery] Language language)
```

Данный метод осуществляет некоторые первичные проверки отправленного кода: на размер, на то, что код этой задачи уже проверяется у текущего пользователя и тд. А после этого ставит ответ студента в очередь на проверку.

Сервис RunCheckerJob с определенной частотой проверяет очередь отправленных решений. Если очередь непустая, то сервис забирает оттуда решение и запускает для него проверку в docker-контейнере. Во время запуска отправляет внутрь контейнера всё необходимое для проверки задачи: чеккер, решение студента и прочие файлы указанные при настройке задачи в xml-файле.

В Листинге 1.8 представлен пример конфигурирования docker-образа. Стоит обратить внимание на создание внутри образа нового пользователя student. Данный подход пригодится в дальнейшем, когда встанет задача конфигурирования образа и важно будет учесть права на чтения тестовых данных в режиме студента.

Листинг 8 — Dockerfile для проверки задач по курсу JavaScript

```
FROM node:14-alpine

RUN apk update && apk upgrade

```

```
# Add new user for sandbox
RUN addgroup -S student && adduser -S -g student student \
    && chown -R student:student /home/student

COPY ./app/ /app/

RUN mkdir -p /source \
    && chown -R student:student /app

WORKDIR app

# Run user as non privileged.
USER student

# Install deps for tests.
RUN npm ci
```

Стоит заметить, что все чеккеры, необходимые для проверки решения студента должны быть написаны самим составителем курса и они будут специфичны для конкретного языка из-за зависимости от фреймворка тестирования.

После проверки из docker-контейнера возвращается результат в RunCheckerJob по которому мы можем узнать статус решения.

Бывают статусы, представленные в листинге 1.9:

Листинг 9 — Модель описывающая статус решения

```
public enum Verdict
    {
        Ok = 1,
        CompilationError = 2,
        RuntimeError = 3,
        SecurityException = 4,
        SandboxError = 5,
        OutputLimit = 6,
```

```
9         TimeLimit = 7,
10        MemoryLimit = 8,
11        WrongAnswer = 9
12    }
```

Результат сохраняется в базу данных и отображается пользователю.

3 Проектирование решения

В этом разделе спроектировано решение в рамках принятых в системе Ulearn паттернов.

Исходя из рассмотренной структуры в пункте 1.2.1 становится ясно, что задачи должны находиться на слайде типа Exercise. Дальнейшие шаги имеют несколько вариантов реализации. Рассмотрим их и выделим плюсы и минусы каждого.

а) Создать новый тип задач, который будет независим от уже существующих в системе. Прописать его типизацию в xsd-схеме. Создать сущность для обработки конфигурации, реализующую контракт ISlide. А так же реализовать систему проверки;

Плюсы:

1) Автору при заполнении xml-файла не будет подсказываться ничего лишнего, относящегося к другим типам задач.

Минусы:

1) Нарушение принципа DRY, так как большую часть логики можно было переиспользовать из уже реализованной в системе.

б) Внести логику реализуемых задач внутрь сущностей типа UniversalExercise;

Плюсы:

1) Логика системы переиспользуется;

2) сокращается количество кода, которое необходимо написать.

Минусы:

1) Нарушение принципа SRP, так как в нашей конфигурации будет поля, которые не нужны в UniversalExercise и наоборот.

в) Создать новый тип задач, сделать его наследником типа UniversalExercise и в xsd-схеме и в обработке конфигурации (так как остальные специализированы только на проверке задач, реализованных на языке C#);

Плюсы:

1) Логика системы переиспользуется;

2) сокращается количество кода, которое необходимо написать.

Минусы:

1) Автор при создании слайда в xml будет видеть в подсказки, которые не участвуют в конфигурации нашей задачи.

Проанализировав данные варианты было принято решение совместить вариант а) и вариант в). В результате получается решение:
Создать новый тип задач, сделать его наследником типа UniversalExercise в обработке конфигурации, но прописать для него независимую типизацию в xsd-схеме. Таким образом единственный минус варианта в) будет решен.

4 Техническая реализация

В этом разделе описана техническая часть работы, реализованная в рамках спроектированных в прошлом пункте идей.

4.1 Xsd-схема блока задачи

Как можно увидеть в Листинге 1.10 типизация тега slide.polygon не наследуется ни от каких других типов, а значит автору будет подсказываться только то, что действительно нужно.

Листинг 10 — Реализация xsd-схемы тега slide.polygon

```
<xs:element name="slide.polygon" type="PolygonSlide" />
<xs:complexType name="PolygonSlide">
    <xs:all>
        <xs:element name="scoring"type="ExerciseScoring"
           minOccurs="0"/>
        <xs:element name="polygonPath"type="xs:string"/>
        <xs:element name="markdown"type="MarkdownBlock"
           minOccurs="0"/>
        <xs:element
           name="exercise.polygon"type="PolygonExercise" />
    </xs:all>
    <xs:attribute name="id" type="NonEmptyString"use="required"
       />
    <xs:attribute name="hide" type="xs:boolean" />
</xs:complexType>
```

4.2 Обработка конфигурации слайда

За обработку конфигурации слайдов с задачами отвечает сущность ExerciseSlide. Если задачи типа UniversalExercise, то внутри ExerciseSlide находится блок поэтому необходимо наш конфигуратор PolygonExerciseSlide отнаследовать от него.

Первая задача, которую нужно решить для генерации слайда — получить доступ к данным о задаче. Для этого вынесем свойство нашей сущности путь до директории с этой информацией.

На этапе валидации слайда необходимо знать, что автор курса не забыл про это поле и указал его. Поэтому оно проверяется явно. Это представлено на листинге 1.11.

Листинг 11 — Проверка полей

```
1    [XmlElement("polygonPath")]
2    public string PolygonPath { get; set; }
3    public override void Validate(SlideLoadingContext context)
4        {
5            if (string.IsNullOrEmpty(PolygonPath))
6                throw new CourseLoadingException("polygonPath
                     not found in slide.polygon");
7            base.Validate(context);
8        }
```

Полный текст задачи генерируется автоматически благодаря вложенной в архив задачи информации. В рассмотренном листинге 1.1 показано, что существует файлы problem.html и problem-statements.css, которые находятся по пути /statements/.html/russian/problem.html и /statements/.html/russian/problem-statements.html соответственно. Они определяют визуальное представление задачи, которое состоит из легенды, примеров входных и выходных данных, информации об ограничениях задачи. Эти файлы используются в нашей обработке конфигурации: файлы читаются и их содержимое подставляется в html-код, выводящийся пользователю.

Оставлена возможность для автора курса на своё усмотрение переопределить выводящуюся пользователю информацию. Для этого ему необходимо внутри слайда добавить тег markdown с необходимыми данными и он будет рассматриваться приемущественнее, чем текст в архиве задачи.

Так же если в архиве задачи есть визуальное представление в виде pdf, то пользователю дается ссылка на этот файл.

Эта логика представлена в листинге 1.12.

Листинг 12 — Формирование блоков

```
private IEnumerable<SlideBlock> GetBlocksProblem(
        string statementsPath,
        string courseId,
        Guid slideId)
        {
            var markdownBlock = Blocks.FirstOrDefault(block =>
               block is MarkdownBlock);
            if (markdownBlock != null)
            {
                yield return markdownBlock;
            }
            else
               if(Directory.Exists(Path.Combine(statementsPath,
               ".html")))
            {
                var htmlDirectoryPath =
                   Path.Combine(statementsPath, ".html",
                   "russian");
                var htmlData = File.ReadAllText(
                Path.Combine(htmlDirectoryPath, "problem.html"));
                yield return RenderFromHtml(htmlData);
            }

            var pdfLink = PolygonPath +
               "/statements/.pdf/russian/problem.pdf";
            yield return new MarkdownBlock($"[Download problem
               conditions in PDF format]({pdfLink})");
        }
```

Далее описана непосредственно обработка конфигурации в методе BuildUp. Рассмотрим листинг 1.13.

Листинг 13 — Обработка конфигурации в PolygonExerciseSlide

```
1  public override void BuildUp(SlideLoadingContext context)
2  {
3      var statementsPath =
           Path.Combine(context.Unit.Directory.FullName,
           PolygonPath, "statements");
4
5      Blocks = GetBlocksProblem(statementsPath, context.CourseId,
           Id)
6              .Concat(Blocks.Where(block => !(block is
                 MarkdownBlock)))
7              .ToArray();
8
9      var problem = GetProblem(
10         Path.Combine(context.Unit.Directory.FullName,
              PolygonPath, "problem.xml"),
11         context.CourseSettings.DefaultLanguage
12     );
13
14     var polygonExercise = Blocks.Single(block => block is
           PolygonExerciseBlock) as PolygonExerciseBlock;
15
16     polygonExercise!.ExerciseDirPath =
           Path.Combine(PolygonPath);
17     polygonExercise.TimeLimitPerTest = problem.TimeLimit;
18     polygonExercise.TimeLimit =
           (int)Math.Ceiling(problem.TimeLimit * problem.TestCount);
19     polygonExercise.UserCodeFilePath =
           problem.PathAuthorSolution;
```

```
20      polygonExercise.Language =
            LanguageHelpers.GuessByExtension(new
            FileInfo(polygonExercise.UserCodeFilePath));
21      polygonExercise.DefaultLanguage =
            context.CourseSettings.DefaultLanguage;
22      polygonExercise.RunCommand = $"python3.8 main.py
            {polygonExercise.Language}
            {polygonExercise.TimeLimitPerTest}
            {polygonExercise.UserCodeFilePath.Split('/', '\\')[1]}";
23
24      Title = problem.Title;
25
26      PrepareSolution(
27          Path.Combine(context.Unit.Directory.FullName,
                PolygonPath, polygonExercise.UserCodeFilePath)
28      );
29
30      base.BuildUp(context);
31  }
```

В строке 3 формируется путь до каталога statements.

В строках 5-7, используя уже рассмотренный метод, генерируются блоки представления задачи.

В строках 9-12 вызывается метод, который из файла problem.xml получает информацию о задаче: ограничение по времени на тест, количество тестов, название задачи, путь до авторского решения.

В строке 14 из множества блоков данного слайда получаю конфигуратор блока PolygonExercise и в строках 16-22 происходит его инициализация:

а) пути до директории с данными задачи;

б) ограничении времени на тест и на выполнение всех тестов;

в) пути до авторского решения;

г) языка авторского решения и языка по-умолчанию(язык, который будет предложен пользователю курса в первую очередь);

д) команду для запуска проверяющей программы внутри docker-контейнера.

В строках 26-28 происходит вызов метода PrepareSolution, задача которого обернуть авторского решение в region, как это представлено в листинге 1.14. Внутрь этого блока будет подставляться решение студента при отправке.

Листинг 14 — Оборачивание кода в region

```
1 \\region Task
2     Code
3 \\endregion Task
```

Завершает метод вызов base.BuildUp, который конфигурируют всю остальную логику связанную с формированием слайдов.

4.3 Обработка конфигурации блока

За обработку конфигурации блока, как уже было сказано в прошлом пункте, отвечает класс PolygonExerciseBlock, который является потомком UniversalExerciseBlock.

В данном классе расположена информация о языках, проверка на которых доступна студенту. Отрывок инициализации представлен в листинге 1.15.

Листинг 15 — Отрывок инициализации LanguagesInfo

```
1 public static Dictionary<Language, LanguageLaunchInfo>
    LanguagesInfo = new Dictionary<Language, LanguageLaunchInfo>
2         {
3             [Common.Language.Java] = new LanguageLaunchInfo
4             {
5                 Compiler = "Java 14",
6                 CompileCommand = "javac {source}",
7                 RunCommand = "java {compilation-result-file}"
```

```
        },
        ...
    }
```

Наибольший интерес в данном классе представляют метод создания RunnerSubmission — сущности содержащую в себе информацию об отправке студентом кода на проверку и способе его запуска. Единственно, что входит в ответственность PolygonExerciseBlock в этой задаче — формирование команды запуска. Всё остальную логику выполняет UniversalExerciseBlock. Данная логика описана в листинге 1.16.

Листинг 16 — Формирование RunnerSubmission

```
public RunnerSubmission CreateSubmission(string submissionId,
   string code, Language language)
{
    var submission =
        base.CreateSubmission(submissionId, code);
    if (!(submission is CommandRunnerSubmission
       commandRunnerSubmission))
        return submission;

    commandRunnerSubmission.RunCommand =
       RunCommandWithArguments(language);

    return commandRunnerSubmission;
}

private string RunCommandWithArguments(Language language)
{
    return $"python3.8 main.py {language} {TimeLimitPerTest}
       {UserCodeFilePath.Split('/', '\\')[1]}";
}
```

Внутри Docker-контейнера будет выполняться команда, которая находится в свойстве RunCommand. Она запустит python-код и передаст ему

в аргументы язык, на котором написано решение, ограничение по времени на тест, а так же путь до файла с решением.

4.4 Проверка решения студента

После отправки решения студентом, запускается docker-контейнер в который подкладывается отправленное решение и начинает выполнение проверяющая программа.

Рассмотрим конфигурацию из Dockerfile, описанную в листинге 1.17.

Листинг 17 — Конфигурация docker-образа

```
FROM ubuntu:20.04

ENV NVM_DIR /usr/local/nvm
ENV NODE_VERSION 14.15.3
ARG DEBIAN_FRONTEND=noninteractive
ENV TZ=Europe/Ekaterinburg

RUN apt-get update
&& apt-get -y upgrade
&& apt-get install -y wget
&& apt-get -y install curl \
# Python 3.8
&& apt-get -y install python3.8 \
# C, C++
&& apt-get -y install build-essential \
# C\#
&& wget https://packages.microsoft.com/config
    /ubuntu/20.10/packages-microsoft-prod.deb -O
       packages-microsoft-prod.deb \
&& dpkg -i packages-microsoft-prod.deb \
&& apt-get update && apt-get install -y apt-transport-https &&
   apt-get install -y dotnet-sdk-5.0 \
```

```
21 && apt-get install -y apt-transport-https && apt-get install -y
      dotnet-runtime-5.0
22 # Java
23 RUN apt-get install -y openjdk-14-jdk \
24 # JavaScript
25 && curl --silent -o- https://raw.githubusercontent.com
26     /creationix/nvm/v0.31.2/install.sh | bash \
27 && . $NVM_DIR/nvm.sh \
28 && nvm install $NODE_VERSION \
29 && nvm alias default $NODE_VERSION \
30 && nvm use default
31
32 ENV NODE_PATH $NVM_DIR/v$NODE_VERSION/lib/node_modules
33 ENV PATH $NVM_DIR/versions/node/v$NODE_VERSION/bin:$PATH
34 ENV NODE_REPL_HISTORY ''
35
36 COPY ./app/ /app/
37
38 WORKDIR app
39
40 RUN useradd student && chmod 700 /app/tests
```

Комментариями в коде помечены языки, которые становятся доступны после выполнения этих команд.

Обратим внимание на строку 39, в которой создается новый пользователь и устанавливаются права доступа на директорию с тестами. Таким образом обеспечивается невозможность получения тестовых данных из решения.

Далее рассмотрим механизм проверки решения.

Первоочередная задача, которая возникает в процессе выполнения проверя — определить язык решения и команды для компиляции и выполнения исходного кода.

Для инкапсуляции данных об этих командах был создан контракт ISourceCodeRunInfo. Рассмотрим листинге 1.18.

Листинг 18 — Контракт ISourceCodeRunInfo

```
class ISourceCodeRunInfo:
    __metaclass__ = ABCMeta
    @abstractmethod
    def file_extension(self): pass
    @abstractmethod
    def format_build_command(self,
        code_filename: str,
        result_filename: str) -> str: pass
    @abstractmethod
    def need_build(self) -> bool: pass
    @abstractmethod
    def format_run_command(self, filename: str) -> str: pass
```

Метод `file_extension` возвращает расширение файла с исходным кодом.

Метод `format_build_command` выполняет задачу формирования команды для компилирования исходного кода. В случае если исходный код некомпилируемый, например, исходных код на языке Python, то возвращается None.

Метод `format_run_command` возвращает команду для запуска решения.

Приведу пример реализации данной сущность для языка C++ в листинге 1.19.

Листинг 19 — Реализация ISourceCodeRunInfo для языка C++

```
class CppRunInfo(ISourceCodeRunInfo):
    def file_extension(self):
        return '.cpp'
    def format_run_command(self, filename: str) -> str:
        return f'./{filename}'
    def need_build(self) -> bool:
        return True
```

```
8     def format_build_command(self, code_filename: str,
          result_filename: str) -> str:
9         return f'g++ -o {result_filename} -O2 {code_filename}'
```

Для поиска нужной реализации данного контракта по названию языка была написана функция `get_run_info_by_language_name`, она представлена в листинге 1.20.

Листинг 20 — Функция поиска по названию языка реализации ISourceCodeRunInfo

```
1 def get_run_info_by_language_name(language_name: str) ->
      ISourceCodeRunInfo:
2     if language_name == 'cpp':
3         return CppRunInfo()
4     elif language_name == 'python3':
5         return PythonRunInfo()
6     ...
```

В листинге 1.21 представленн класс TaskCodeRunner, который реализует логику сборку, запуск кода и запуск теста, используя взаимодействие с процессами.

Листинг 21 — Сигнатура класса TaskCodeRunner

```
1 class TaskCodeRunner:
2     def build(self, code_filename: str, result_filename: str):
          pass
3     def run(self, test_file: str): pass
4     def run_test(self, code_filename: str, test_file: str): pass
```

Проверка решения студента осуществляется по следующему алгоритму:

а) сборка исходного кода чеккера check.cpp, который мы получили из директории задачи;

б) предобработка решения студента: удаления строк определения региона; в случае с java кодом переименование названия файла в соответствие с названием класса внутри файла;

в) получение ISourceCodeRunInfo по названию языка полученному в аргументах программы;

г) компиляция исходного кода студента, если это требуется

д) для каждого теста

1) запуск программы студента на этом тесте

2) сравнение результата выполнения с авторским решением используя чеккер

е) обработка ошибок, при возникновении

Результатом выполнения проверяющей программы является json, структура которого продемонстрирована в листинге 1.22.

Листинг 22 — RunningResult

```
{
    "Verdict": "Ok" | "WrongAnswer" | "TimeLimit" | ...
    "TestResultInfo": {
           "TestNumber": "1" | "2" | ... ,
           "Input": // test data,
           "CorrectOutput": // correct test output ,
           "StudentOutput": // student output
       }
}
```

4.5 Логирование

Помимо этого, у авторов курсов могут возникнуть вопросы почему решение студента не проходит определенные тесты или подозрение в том, что система работает неправильно. Поэтому было реализовано логирование жизненного цикла проверяющей системы с выводом в интерфейс, но только для администраторов данного курса.

4.6 Добавление нового языка

В дальнейшем у авторов может возникнуть желание добавить новый язык программирования, чтобы у студентов был еще более разнообразный

выбор. Для встраивания еще одного языка следует выполнить следующие шаги для этого языка:

а) прописать команды установки окружния в dockerfile;

б) реализовать сущность ISourceCodeRunInfo;

в) прописать условие для обнаружения ISourceCodeRunInfo по названию языка в методе `get_run_info_by_language_name`;

г) в классе PolygonExerciseBlock прописать информацию о языке в поле LanguagesInfo.

4.7 Тестирование

Логика, реализованная в рамках данной работы, была протестирована несколькими методами.

а) логика работы docker-образа была проверена unit-тестом, который запускает его, отправляет туда тестовую задачу с авторским решением и проверяет, что результатом выполнения стал статус Ok;

б) курс для тестирования системы Ulearn был дополнен несколькими слайдами, где заложена возможность ручного тестирования данной логики.

Заключение

В рамках работы была поставлена следующая цель: реализовать поддержку задач в формате Codeforces в образовательной платформе Ulearn, а именно возможность размещения таких курсов, автопроверки решений по задачам.

В работе был проведен анализ предметной области, а именно создания задачи формата Codeforces, описаны пункты, за которые будет ответственна система и пункты, которые остаются на ответственность авторов курсов. Была изучена текущая инфраструктура системы Ulearn и рассмотрены варианты её расширения для текущей задачи. Из этих вариантов был выделен один, который не имел противоречий с принципами ООП и с установленными в системе Ulearn правилами.

Данное решение был реализовано, протестировано методами ручного и unit тестирования и выпущено в релиз.

После релиза реализовывались пожелания от авторов курса. Например, возможность вывода информации о тесте(входные данные, правильные результат, результат решения) при неправильном результате решения студента.

Также аосле релиза благодаря новым возможностям был создан курс по изучению языка программирования Python для школьников.

В дальнейших планах присутствует создание курса по алгоритмам и структурам данных для студентов УрФУ.

Список использованных источников

1. *Система подготовки олимпиадных задач по программированию в УрГУ* Алексей Самсонов, Александр Ипатов
URL: `https://sp.urfu.ru/library/system.html` (дата обращения: 15.10.2020).

2. *E-learning platform by SKB Kontur.* kontur-edu. URL: `https://github.com/kontur-edu/Ulearn` SKB Kontur.

3. *Коротко о testlib.h.* Mike Mirzayanov. URL: `https://codeforces.com/testlib` (дата обращения: 16.10.2020)

4. *Валидаторы на testlib.h.* I_love_Hoang_Yen. URL: `https://codeforces.com/blog/entry/18426` (дата обращения: 16.10.2020)

5. *Генераторы на testlib.h.* Mike Mirzayanov. URL: `https://codeforces.com/blog/entry/18291` (дата обращения: 16.10.2020)

6. *Checkers with testlib.h.* Zlobober. URL: `https://codeforces.com/blog/entry/18431` (дата обращения: 16.10.2020)

7. Mike Mirzayanov. URL: `https://polygon.codeforces.com/` (дата обращения: 16.10.2020)