Московский государственный технический университет
имени Н. Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Программное обеспечение ЭВМ
и информационные технологии»

Расчётно-пояснительная записка
к курсовому проекту по базам данных на тему

Web-приложение Coding Tutors

Исполнитель: студент ИУ7-61 Петухов И. С. _____

Руководитель: преподаватель ИУ7 Просуков Е. А. _____

Москва, 2016

Задание на выполнение курсовой работы

1. Тема курсовой работы

«Web-приложение Coding Tutors (система, в которой программисты могут помогать желающим разобраться в программировании)»

2. Техническое задание

Разработать систему, позволяющую зарегистрированным пользователям создавать новые задачи, просматривать задачи других пользователей, предлагать свою кандидатуру на решение задачи с указанием комментария. Автор задачи может просматривать список кандидатов на решение задачи. Каждая заявка на решение задачи будет сопровождаться комментарием кандидата и индексом полезности, который высчитывается, исходя из качественных показателей кандидата (рейтинг) и совпадений персональных данных автора задачи и кандидата (общий ВУЗ). Пользователь может закрывать свои неактуальные задачи и благодарить кандидатов, что будет влиять на рейтинг. Общая страница задач должна иметь пагинацию. Пользователь должен иметь личный кабинет, с возможностью изменить некоторые персональные данные.

Содержание

Задание на выполнение курсовой работы 2
1. Тема курсовой работы 2
2. Техническое задание 2
Введение 4
1 Аналитический раздел 5
1.1 Описание предметной области 5
1.2 Определение требований к структуре базы данных 6
1.2.1 Определение объема и типов данных 6
1.2.2 Определение способа использования данных 7
1.2.3 Определение правил 7
1.3 Разработка логической модели данных 8
1.3.1 Определение сущностей, связей между сущностями и атрибутов сущностей 8
1.3.2 Диаграмма БД 9
2 Конструкторский раздел 11
2.1 Архитектура Web-приложения 11
2.1.1 Вид 11
2.1.2 Контроллеры 12
2.1.3 Модель 13
3 Технологический раздел 15
3.1 Выбор языка программирования 15
3.2 Модель 15
3.3 Вид 16
3.4 Контроллеры и веб сервер 18
3.5 Навигация по сайту 19
3.5.1 Для неавторизованного пользователя 19
3.5.2 Для авторизованного пользователя 19
Заключение 21

Введение

Программирование - важнейший предмет, который формирует логическое мышление человека, а так же является эффективным инструментом в решении большинства научных и повседневных задач. В настоящее время, предмет программирование преподается во многих школах и университетах. Однако, многие люди сталкиваются с проблемами в изучении этого предмета. Данная проблема связана с обучением в группах, что для многих затрудняет понимание. Наличие персонального репетитора обычно решает эту проблему. Большинство существующих сайтов позволяют найти репетитора на долгий срок, что обычно бывает не нужно. Поэтому принято решение разработать сайт для поиска помощников в решении конкретных задач. При создании сайта будут использованы современные подходы работы с базами данных и классические методы создания сервера приложения.

1 Аналитический раздел

1.1 Описание предметной области

Для студентов технических вузов обязателен предмет программирование. Но большинство имеет проблемы при изучении этого предмета. Многие находят выход в списывании, что несомненно подрывает российское образование. Некоторые обращаются к репетиторам, но эти услуги стоит больших денег, и формат таких занятий не совсем подходит к теме IT образования. Более правильный подход - находить помощника для конкретных возникших проблем, будь то лабораторная работа или контрольная по определенной теме.

Сайт должен позволять зарегистрироваться и авторизоваться неавторизованным пользователям. Регистрация предполагает ввод имени, фамилии, e-mail, пароля и выбора университета. При авторизации необходимо ввести e-mail и пароль. Авторизованные пользователи должны иметь возможность выйти из сайта.

На сайте должны быть два вида пользователей: человек, который запрашивает услугу (далее ученик) и человек, который предлагает услугу (далее репетитор). Однако, один человек может как нуждаться в услуге, так и предлагать её. Поэтому на сайте должен быть один вид пользователя, с одинаковыми возможностями как запрашивать услугу, так и предлагать её.

Основная функция сайта - это создание задач, по которым нужно получить помощь. Пользователь (ученик) должен иметь возможность создавать задачу, просматривать свои задачи, отсортированные по времени, закрывать неактуальные задачи, видеть предложения, оставленные репетиторами. Т.о. задача состоит из: названия, текста условия, флага актуальности, даты создания.

Перед тем, как создать задачу, пользователю может пригодиться поиск по всем опубликованным задачам, и просмотреть список откликнувшихся репетиторов к этим задачам. Это позволит не создавать новую задачу, а сразу обратиться к нужному репетитору, который сможет помочь.

Репетиторы должны иметь возможность просматривать все актуальные задачи, оставлять заявки к ним или видеть, что заявка уже оставлена им ранее.

Заявка должна включать в себя номер задачи, к которой она оставляется, идентификатор репетитора и комментарий репетитора.

Для удобного просмотра задач, необходимо ввести понятие категория. Пользователь, при создании задачи, должен выбрать, к какой категории относится задача. Репетитор может выбрать категорию, по которой он хочет видеть актуальные задачи. Пример задачи в таблице 1.1.

Так как задач может быть много, необходимо ввести пагинацию.

На одну задачу могут откликнуться большое количество репетиторов, поэтому необходимо помочь ученику выбрать нужного репетитора. Для этого вводится

Таблица 1.1 — Пример задачи

Название	Помогите решить задачу на C
Категория	Structured
Описание	В задаче нужно вывести на экран интеграл функции y = x * x на отрезке -43 1020

понятие полезности заявки. Этот показатель будет высчитываться из совпадений данных ученика и репетитора, а также рейтинга репетитора. Например при регистрации необходимо указать университет, и если ученик и репетитор учатся/учились в одном университете, то ценность заявки возрастает. Связанно это с тем, что ученик и репетитор будут иметь возможность заниматься прямо в университете, а так же с тем, что репетитор возможно будет иметь представление, об преподавателях ученика, и преподаваемой программе в этом университете.

При просмотре заявок к задаче, автор задачи должен иметь возможность связаться с репетитором. Для этого указывается e-mail репетитора. Также автор может оценить заявку, что и будет влиять на рейтинг репетитора. Пример списка репетиторов к задаче в таблице 1.2.

Таблица 1.2 — Пример списка репетиторов

email	соответствие	note
timur2@mail.ru	0	Это легко, помогу бесплатно
ilya16@yandex.ru	10	Помогу за еду

Так как создаваемый сайт имеет образовательную цель, то было бы полезно привести список рекомендуемой литературы. Этот список нужно разбить по категориям. Добавить книгу может любой человек. Книги можно оценивать.

1.2 Определение требований к структуре базы данных

Прежде чем начинать разработку модели данных, необходимо определить объема и типы данных, а также способа использования данных.

1.2.1 Определение объема и типов данных

а) Из анализа предметной области можно выделить следующие категории данных:

- Пользователи
- Задачи
- Книги
- Заявки

— Категории

— Университеты

б) Для каждой категории данных необходимо учитывать типы сведений, указанных в таблице 1.3.

Таблица 1.3 — Категории и типы сведений

Категория	Типы сведений
Пользователи	e-mail, университет, имя, фамилия, пароль, рейтинг
Задачи	пользователь создатель, категория, названия, описание, флаг актуальности, дата создания
Книги	название, автор книги, рейтинг, категория
Заявки	задача, пользователь создатель, заметка, полезность заявки
Категории	название
Университеты	полное название, короткое название

1.2.2 Определение способа использования данных

а) Из анализа предметной области можно определить следующие категории пользователей:

— Неавторизованные пользователи

— Авторизованные пользователи (пользователи)

б) Каждая категория пользователей выполняет задачи, указанные в таблице 1.4.

Таблица 1.4 — Категории пользователей и задачи

Категории пользователей	Задачи
Авторизованные пользователи	выход, функции учениов и репетиторов
Неавторизованные пользователи	регистрация, авторизация

в) Из анализа предметной области можно определить следующие логические категории пользователей:

— Ученики

— Репетиторы

г) Каждая логическая категория пользователей выполняет задачи, указанные в таблице 1.5.

1.2.3 Определение правил

— данные о пользователе включают: имя, фамилию, e-mail, университет, пароль

Таблица 1.5 — Логические категории пользователей и их задачи

Категории пользователей	Задачи
Ученики	создавать задачи, закрывать свои задачи, оценивать заявки, осуществлять поиск по задачам
Репетиторы	просматривать актуальные задачи по категориям, оставлять заявки
Общие задачи	добавлять книги, просматривать книги по категориям, оценивать книги

— имя, фамилия, e-mail и пароль не могут быть меньше 4 символов

— пароль должен хранится в виде хеша от пароля в целях безопасности

— университет выбирается из предоставленного списка

— авторизация пользователя происходит по e-mail и паролю

— сведения о задаче включают название, описание, категорию

— сведения о книге включают название, авторы, категорию

— пользователь должен иметь возможность просматривать список книг по категориям

— пользователь должен иметь возможность просматривать список задач по категориям

— пользователь может оценить книгу

— пользователь может оценить одну книгу только один раз

— пользователь может оставить заявку на задачу

— данные о заявке на решение задачи включают комментарий пользователя

— пользователь может оценивать заявку только на свои задачи

— пользователь может оценить одну заявку только один раз

— пользователь должен иметь возможность осуществлять поиск по задачам

1.3 Разработка логической модели данных

1.3.1 Определение сущностей, связей между сущностями и атрибутов сущностей

Чтобы определить, какие таблицы следует добавить к базе данных, следует обратиться к требованиям к системе, разработанным при анализе предметной области. Каждая выделенная категория (всего их 6) представляет основную таблицу и соответствующий табличный объект в структуре базы данных. Обозначим каждую таблицу названием одной из категорий. Для согласованности назовем таблицы так: Member, Task, Book, Category, WantToHelp, University.

Согласно правилам из системных требований, пользователь может оценить одну книгу только один раз. Таблицы Member и Book имеют связь многие ко многим. Один пользователь может оценить несколько книг и одна книга может быть оценена несколькими пользователями. Добавим еще одну таблицу (Likebook) для учета оценки пользователя к книге. Теперь в общей сложности должно быть 7 таблиц.

Для определения столбцов таблиц обратимся к требованиям к системе, разработанным при анализе предметной области. Для каждой категории данных определена информация, которая входит в эту категорию. Эта информация определяет столбцы. Добавляя к каждой таблице столбцы, следует помнить, что когда столбцы ссылаются на данные из связанной таблицы, обычно требуется столбец с идентификатором из связанной таблицы.

Таблица 1.6 — Таблицы и столбцы

Таблица	Столбцы
Member	email, university, firstname, surname, hashpassword, likes
Task	taskId, memberEmail, categoryId, title, text, open, countWantToHelp, datetime
Book	bookId, categoryId, title, author, likes
Category	categoryId, name
WantToHelp	WantToHelpId, taskId, memberEmail, note, levelOfComplance, isLike
University	shortname, fullname
Likebook	memberEmail, bookId

1.3.2 Диаграмма БД

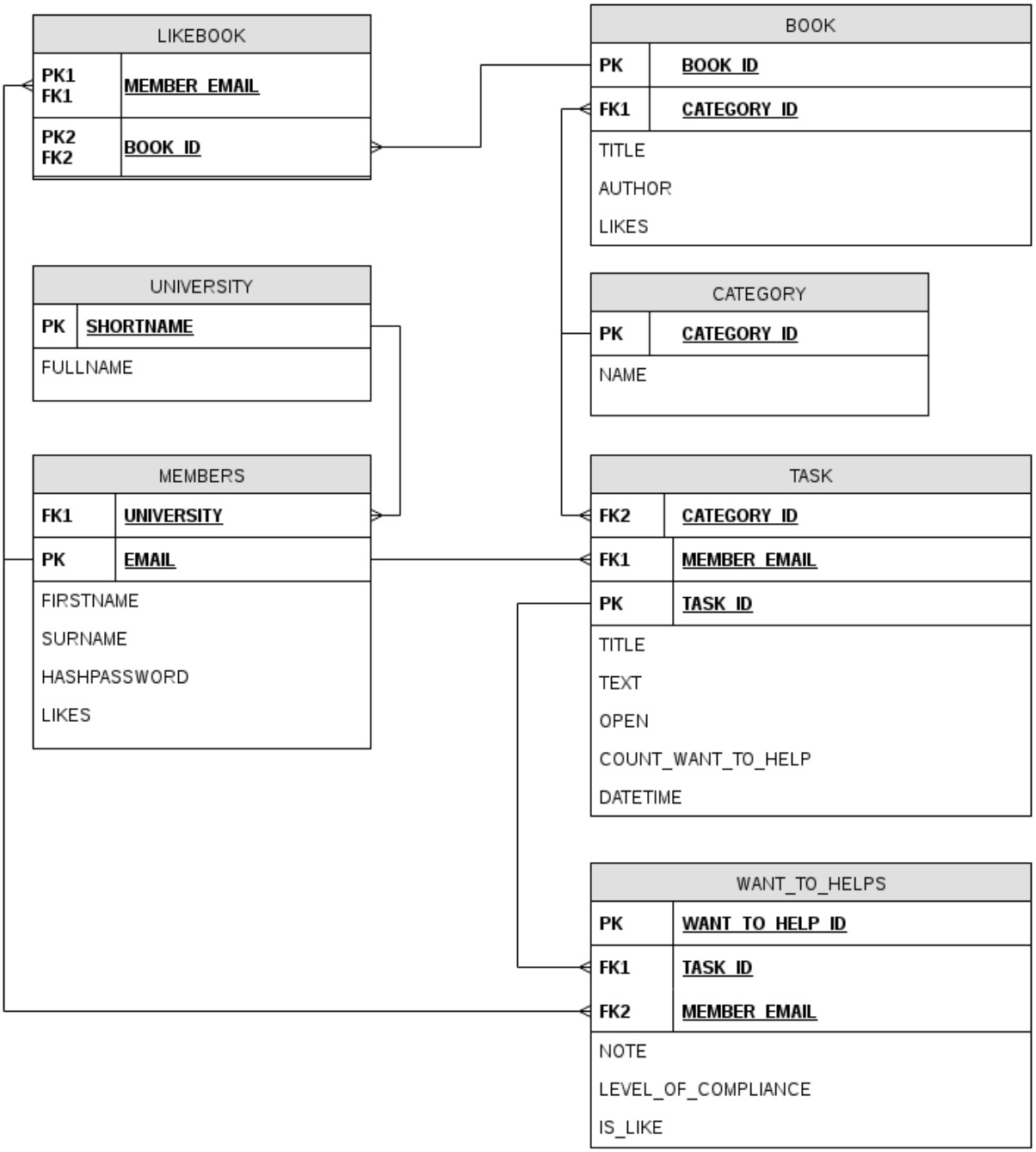

Рисунок 1.1 — ER - диаграмма.

2 Конструкторский раздел

2.1 Архитектура Web-приложения

Будет использоваться популярный шаблон проектирования Web приложений Model-View-Controller. Этот шаблон разделяет работу веб-приложения на три отдельные функциональные роли: модель данных (model), пользовательский интерфейс (view) и управляющую логику (controller). Таким образом, изменения, вносимые в один из компонентов, оказывают минимально возможное воздействие на другие компоненты.

В данном паттерне модель не зависит от представления или управляющей логики, что делает возможным проектирование модели как независимого компонента и, например, создавать несколько представлений для одной модели.

Фактически, связка вида и контроллера является интерфейсом пользователя. Причем, если компоненты вида обычно можно повторно использовать в других компонентах системы, то контроллер часто является специфичным для данного конкретного случая.

Модель не зависит ни от вида, ни от контроллера, что позволяет одновременно строить различные интерфейсы пользователя для взаимодействия с одной и той же моделью данных.

2.1.1 Вид

Вид (View) представляет собой компонент системы для отображения состояния модели в понятном человеку представлении. Это могут быть диалоги, формы и другие визуальные и не визуальные (например, синтезатор речи) средства взаимодействия человека с системой. Вид не изменяет данные напрямую (режим только чтение), данные изменяются при помощи контроллера.

В данной работе вид будет представлен страницами:

а) add-book (добавление книги)
б) authorization (авторизация)
в) create-task (создание задачи)
г) edit-profile (редактирование профиля)
д) index (главная страница)
е) list-books (список книг)
ж) list-my-tasks (список задач пользователя)
з) list-search-tasks (результат поиска по задачам)
и) list-tasks (список задач пользователей)
к) registration (регистрация)

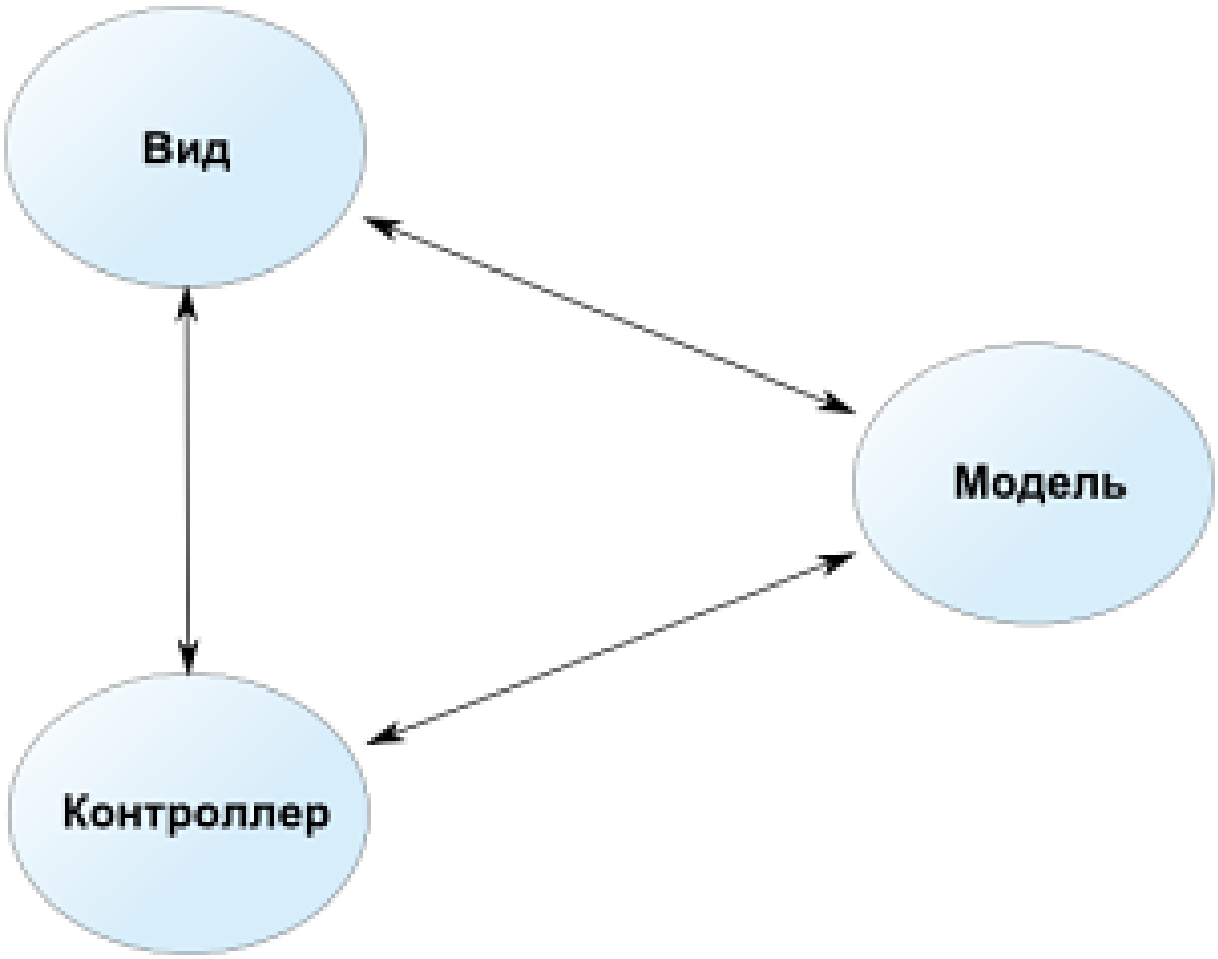

Рисунок 2.1 — Структура шаблона “Модель – Вид – Контроллер”.

2.1.2 Контроллеры

Контроллер (Controller) является средством, при помощи которого пользователи взаимодействуют с системой. Это может быть клавиатура, мышь и т. д. А также является управляющим элементом для обмена данными и сообщениями между видом и моделью.

В данной работе контроллеры представлены классами в коде, которые обрабатывают get и post запросы из страниц вида.

а) AddBookServlet
   — Post запрос: Добавление книги в БД.

б) AuthorizationServlet
   — Post запрос: Авторизация пользователя.

в) CategoryServlet
   — Post запрос: Установка категории и вывод списка задач этой категории.

г) CloseTaskServlet
   — Post запрос: Закрытие задачи

д) CreateTaskServlet

— Post запрос: Добавление задачи в БД.

е) EditProfileServlet

— Post запрос: Изменение данных пользователя.

ж) LibraryServlet

— Post запрос: Установка категории и вывод списка книг этой категории.

з) LikeBookServlet

— Post запрос: Оценка книги.

и) LikeWantToHelpServlet

— Post запрос: Оценка запроса.

к) RegistrationServlet

— Post запрос: Регистрация пользователя.

л) LogoutServlet

— Get запрос: Выход авторизованного пользователя.

м) SearchServlet

— Post запрос: Текстовый поиск по задачам.

н) WantToHelpServlet

— Post запрос: Добавление заявки на помощь.

2.1.3 Модель

Модель (Model) представляет собой данные, с которыми оперирует приложение. Это могут быть как данные базы данных, так и любая другая структура данных, описывающая некоторые объекты системы и их состояние.

Различают пассивный и активный режимы работы с моделью данных.

При **пассивном режиме** данные на клиентской стороне перечитываются при выполнении некоторых действий пользователя, например, запроса на вывод информации об объектах.

При **активном режиме** вводятся дополнительные обработчики и слушатели сообщений, которые посылают компонентам уровня представления сообщения о том, что данные модели были изменены и требуется обновление данных на стороне клиента “View”.

В данной работе используется пассивный режим работы с моделью данных.

Модели представлены классами:

а) Book

б) Category

в) LikeBook

г) Member

д) Task

е) University

ж) WantToHelp

Использование реляционной базы данных для хранения объектно-ориентированных данных приводит к семантическому разрыву, заставляя программистов писать программное обеспечение, которое должно уметь, как обрабатывать данные в объектно-ориентированном виде, так и уметь сохранить эти данные в реляционной форме. Эта постоянная необходимость в преобразовании между двумя разными формами данных не только сильно снижает производительность, но и создает трудности для программистов, так как обе формы данных накладывают ограничения друг на друга. Разработано множество пакетов, устраняющих необходимость в преобразовании объектов для хранения в реляционных базах данных.

Некоторые пакеты решают эту проблему, предоставляя библиотеки классов, способных выполнять такие преобразования автоматически. Имея список таблиц в базе данных и объектов в программе, они автоматически преобразуют запросы из одного вида в другой.

С точки зрения программиста система должна выглядеть как постоянное хранилище объектов. Он может просто создавать объекты и работать с ними как обычно, а они автоматически будут сохраняться в реляционной базе данных.

ORM (англ. Object-Relational Mapping, рус. объектно-реляционное отображение) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, избавляет программиста от написания большого количества кода, часто однообразного и подверженного ошибкам, тем самым значительно повышая скорость разработки. Кроме того, большинство современных реализаций ORM позволяют программисту при необходимости самому жёстко задать код SQL-запросов, который будет использоваться при тех или иных действиях (сохранение в базу данных, загрузка, поиск и т. д.) с постоянным объектом.

На практике всё не так просто и очевидно. Все системы ORM обычно проявляют себя в том или ином виде, уменьшая в некотором роде возможность игнорирования базы данных. Более того, слой транзакций может быть медленным и неэффективным (особенно в терминах сгенерированного SQL). Все это может привести к тому, что программы будут работать медленнее и использовать больше памяти, чем программы, написанные «вручную».

На данном этапе разработки системы принято решение использовать технологию ORM.

3 Технологический раздел

3.1 Выбор языка программирования

В качестве языка программирования был выбрал язык **Java**. Данный язык активно используется в качестве написания серверов веб приложений, а также имеет большое количество библиотек и фреймворков, облегчающих создание целостного веб приложения.

При создания приложения использовалась среда разработки программного обеспечения **IntelliJ IDEA** от компании JetBrains. Данная IDE очень удобна для написания и отладки программ, имеет большое количество горячих клавиш, ускоряющих написание кода.

3.2 Модель

Модель представлена Java - классом.

Листинг 3.1 — Пример модели Book.

```
1  public class Book {
2      private int bookId;
3      private Category category;
4      private String title;
5      private String author;
6      private int likes = 0;
7
8      public Book(String title, String author, Category category) {
9          this.title = title;
10         this.author = author;
11         this.category = category;
12     }
13 }
```

Класс модели с помощью ORM отображаются на реляционную таблицу базы данных. Концепцию ORM в Java реализует JPA. **Java Persistence API (JPA)** — API, входящий с версии Java 5 в состав платформ Java SE и Java EE, предоставляет возможность сохранять в удобном виде Java-объекты в базе данных.

Листинг 3.2 — Пример модели Book в контексе ORM.

```
1  public class Book {
2      @Id
3      @Column(name = "BOOKID")
4      @GeneratedValue(strategy = GenerationType.AUTO)
5      private int bookId;
6      @ManyToOne
7      @JoinColumn(name = "CATEGORYID")
8      private Category category;
```

```
9       @Column(name = "TITLE")
10      private String title;
11      @Column(name = "AUTHOR")
12      private String author;
13      @Column(name = "LIKES")
14      private int likes = 0;
15
16      public Book(String title, String author, Category category) {
17          this.title = title;
18          this.author = author;
19          this.category = category;
20      }
21  }
```

3.3 Вид

Вид создан с помощью JSP. **JSP (JavaServer Pages)** — технология, позволяющая веб-разработчикам создавать содержимое, которое имеет как статические, так и динамические компоненты. Страница JSP содержит текст двух типов: статические исходные данные, которые могут быть оформлены в одном из текстовых форматов HTML, SVG, WML, или XML, и JSP- элементы, которые конструируют динамическое содержимое.

Листинг 3.3 — Пример страницы list-tasks.jsp

```
1  <%
2  int catId = new Integer((String)
       request.getSession().getAttribute("category"));
3  Category category = (Category) BaseDAO.find(Category.class, catId);
4
5  List<Task> tasks = TaskDAO.findOpenByCategory(category, start, 10);
6
7  for (Task task : tasks) {
8      if (!task.getMemberNeed().getEmail().equals(member.getEmail())) {%>
9          <div class="panel panel-success">
10
11             <div class="panel-heading">
12                 <%= "#" + task.getTaskId() + " | " + task.getTitle() + " |
       " + task.getCategory().getName() %>
13             </div>
14
15             <div class="panel-body">
16                 <%= task.getText() %>
17             </div>
18
19             <form action="want-to-help" method="post">
```

```
20                  <div class=" input-group panel-body">
21                      <% String comment =
    WantToHelpDAO.getNoteForTaskByEmail(task, member); %>
22                      <% if (comment == null) { %>
23                      <span class="input-group-btn">
24                          <button class="btn btn-success "
    type="submit">Помочь!</button>
25                      </span>
26                      <input type="text" class="form-control" name = "note"
    placeholder="note">
27                      <% } else { %>
28                      <span class="input-group-addon">
29                          Ваш комментарий
30                      </span>
31                      <input type="text" class="form-control" value="<%=
    comment %>" readonly>
32                      <% } %>
33                  </div>
34
35                  <input type="hidden"  name = "task-id" value = "<%=
    task.getTaskId() %>">
36                  <input type="hidden"  name = "member-email" value = "<%=
    member.getEmail() %>">
37              </form>
38
39
40              <div class="panel-footer">
41                  <h5> Хотят помочь: <strong> <%= task.getCountWantToHelp()%>
    </strong> </h5>
42                  <hr>
43                  <%
44                      long duration = new Date().getTime() -
    task.getDateTimeField().getTime();
45                      long diffdays =
    TimeUnit.MILLISECONDS.toDays(duration);
46                  %>
47                  <h5> Дата публикации: <strong> <%= diffdays%> days
    ago</strong> </h5>
48              </div>
49          </div>
50      <%}
51  }%>
```

3.4 Контроллеры и веб сервер

Контроллеры реализуются Java сервлетами. **Сервлет** является интерфейсом Java, реализация которого расширяет функциональные возможности сервера. Сервлет взаимодействует с клиентами посредством принципа запрос-ответ. Хотя сервлеты могут обслуживать любые запросы, они обычно используются для расширения веб-серверов.

В качестве веб сервера используется Apache Tomcat. **Tomcat** — контейнер сервлетов с открытым исходным кодом, разрабатываемый Apache Software Foundation. Реализует спецификацию сервлетов и спецификацию JavaServer Pages (JSP). Написан на языке Java.

Листинг 3.4 — Пример сервлета CreateTaskServlet

```
1  public class CreateTaskServlet extends HttpServlet {
2
3      protected void doPost(HttpServletRequest request, HttpServletResponse
       response) throws ServletException, IOException {
4          PrintWriter pw = response.getWriter();
5          response.setContentType("text/html; charset=UTF-8");
6          request.setCharacterEncoding("UTF-8");
7
8          if (!verificationParametersInRequest(request)) {
9              pw.println(ServletHelper.BAD_FORM);
10             pw.println(ServletHelper.getHtmlRedirect("/create-task.jsp"));
11             return;
12         }
13
14         Task task = new Task();
15         Member member = (Member)
       request.getSession().getAttribute("member");
16
17         task.setMemberNeed(member);
18         task.setText(request.getParameter("text"));
19         task.setTitle(request.getParameter("title"));
20         task.setCategory(CategoryDAO.getMap().get(new
       Integer(request.getParameter("category"))));
21         task.setDateTimeField(new Timestamp(new Date().getTime()));
22
23         if (!TaskDAO.insert(task)) {
24             pw.println(ServletHelper.ERROR);
25             pw.println(ServletHelper.getHtmlRedirect("/create-task.jsp"));
26             return;
27         }
28
29         pw.println(ServletHelper.SUCCESSFUL);
30         pw.println(ServletHelper.getHtmlRedirect("/list-my-tasks.jsp"));
```

```
31      }
32
33      private boolean verificationParametersInRequest(HttpServletRequest
        request) {
34          if (request.getParameter("title").length() < 4) {
35              return false;
36          }
37
38          if (request.getParameter("text").length() < 4) {
39              return false;
40          }
41
42          return true;
43      }
44  }
```

3.5 Навигация по сайту

3.5.1 Для неавторизованного пользователя

Рисунок 3.1 — Шапка сайта для неавторизованного пользователя.

Нажав на соответствующие ссылки можно перейти к страницам регистрации и авторизации.

3.5.2 Для авторизованного пользователя

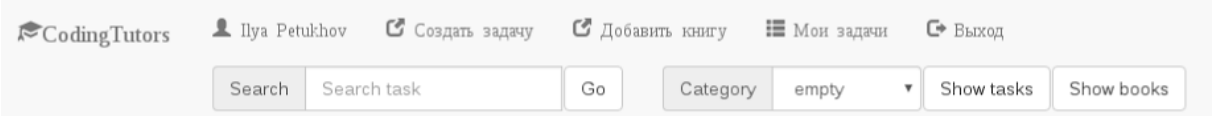

Рисунок 3.2 — Шапка сайта для авторизованного пользователя.

Перейдя по соответствующим ссылкам, можно:

а) редактировать профиль
б) выйти
в) создать задачу
г) добавить книгу
д) перейти к списку своих задач
е) осуществить поиск по задачам

ж) выбрать категорию для отображения задач/книг

з) перейти к списку активных задач

и) перейти к списку книг

Заключение

В ходе выполнения работы был реализован программный продукт, полностью отвечающий требованиям, изложенным в техническом задании, а именно веб приложение Coding tutors. Методы написания приложения позволяют в дальнейшем развивать приложение, легко изменять существующую функциональность и добавлять новую.