

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МОЭВМ**

**ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ**  
**по дисциплине «Нереляционные базы данных»**  
**Тема: ИС для бердвотчинга (Mongo)**

Студентка гр. 7382	_____	Лящевская А.П.
Студент гр. 7381	_____	Минуллин М.А.
Студентка гр. 7381	_____	Машина Ю.Д.
Преподаватель	_____	Заславский М.М.

Санкт-Петербург  
2020

## **ЗАДАНИЕ НА ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ**

Студентка Лящевская А.П., группа 7382

Студент Минуллин М.А., группа 7381

Студентка Машина Ю.Д., группа 7381

Тема работы: ИС для бердвотчинга (Mongo).

Исходные данные:

Создание приложения, в функционал которого входят ввод данных, пользователи и их профили (страницы), система достижений, комментарии, статистика.

Содержание пояснительной записки:

«Содержание»

«Введение»

«Сценарий использования»

«Модель данных»

«Разработка приложения»

«Вывод»

«Приложение»

Предполагаемый объем пояснительной записки:

Не менее 25 страниц.

Дата выдачи задания: 18.09.2020

Дата сдачи реферата:

Дата защиты реферата:

Студентка гр. 7382

\_\_\_\_\_

Лящевская А.П.

Студент гр. 7381

\_\_\_\_\_

Минуллин М.А.

Студентка гр. 7381

\_\_\_\_\_

Машина Ю.Д.

Преподаватель

\_\_\_\_\_

Заславский М.М.

## **АННОТАЦИЯ**

В рамках данного курса предполагалось какое-либо приложение в команде на одну из поставленных тем. Была выбрана тема создания приложения ИС для бердвотчинга (MongoDB).

Найти исходный код и дополнительную информацию можно по ссылке:  
<https://github.com/moevm/nosql2h20-bird-mongo>.

## **SUMMARY**

In this course our task was to develop an application in a team, according to one of the offered topics. We chose creating an IS application for birdwatching (MongoDB).

You can find the source code and additional information here:  
<https://github.com/moevm/nosql2h20-bird-mongo>.

## СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ.....	6
2. КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РЕШЕНИЮ .....	7
3. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ.....	8
4. МОДЕЛЬ ДАННЫХ.....	15
5. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ .....	20
6. ВЫВОДЫ.....	23
7. ПРИЛОЖЕНИЯ .....	24
8. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	25

## **1. ВВЕДЕНИЕ**

Целью работы является создание приложения, в функционал которого входят ввод данных, пользователи и их профили (страницы), система достижений, комментарии, статистика. Выбранный нами стек технологий включает в себя Node.js, Express.js, mongoose, react-bootstrap, AWS.

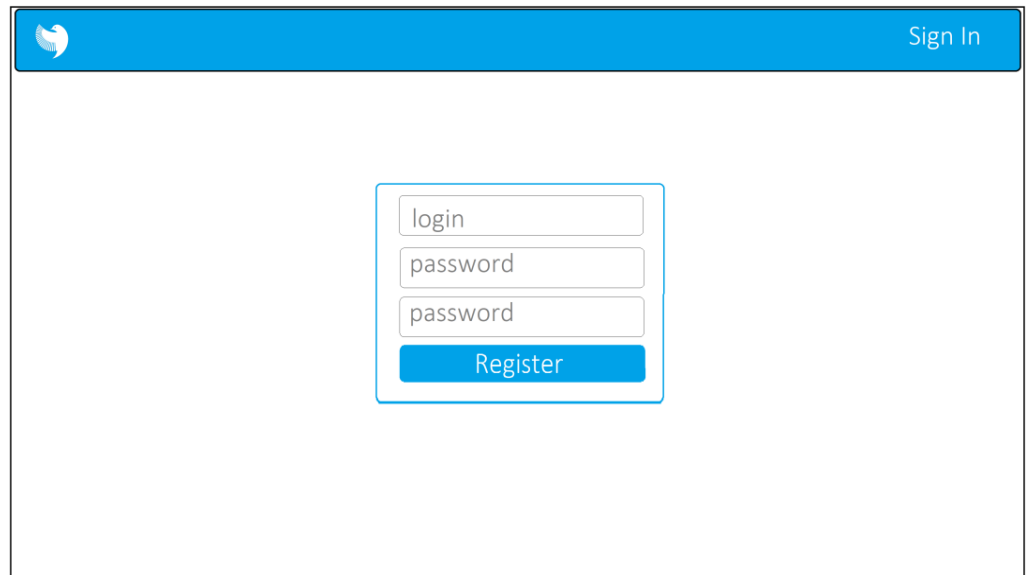
## **2. КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РЕШЕНИЮ**

Требуется разработать user-friendly приложение с использованием MongoDB. В функционал приложения должны входить ввод данных, пользователи и их профили (страницы), система достижений, комментарии, статистика.

### 3. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

#### 3.1 Макеты UI

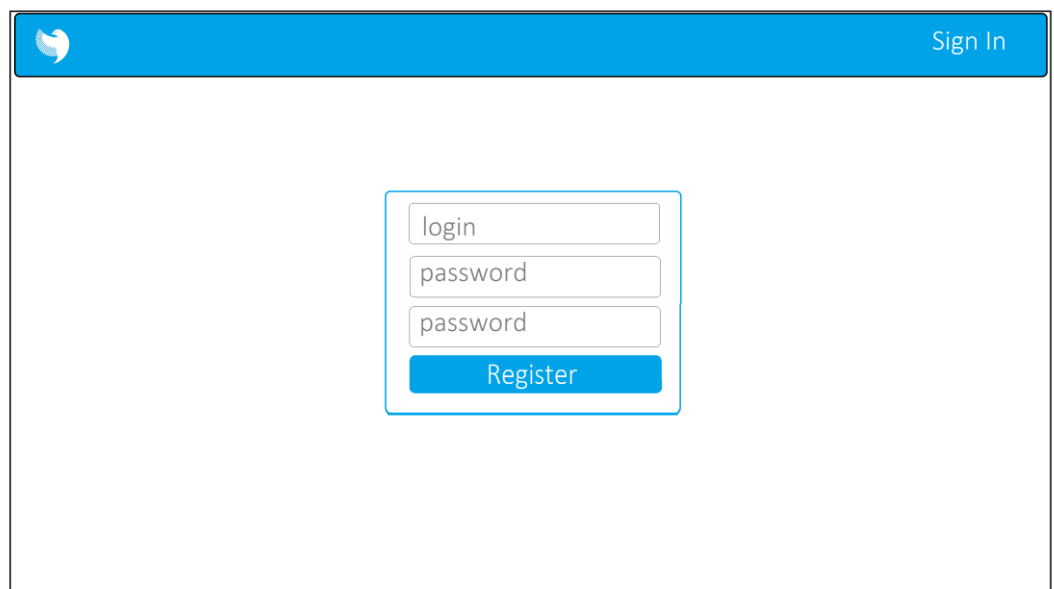
##### 1. Авторизация.



The mockup shows a web interface for authorization. At the top is a blue navigation bar. On the left side of this bar is a white bird icon, and on the right is the text 'Sign In'. The main content area is white and contains a centered white box with a blue border. Inside this box are three input fields: the first is labeled 'login', the second and third are both labeled 'password'. Below these fields is a blue button with the text 'Register' in white.

Рисунок 3.1. — Авторизация.

##### 2. Регистрация.



This mockup is identical to the one for authorization. It features a blue header bar with a white bird icon on the left and a 'Sign In' link on the right. The main content area is white and contains a centered white box with a blue border. Inside this box are three input fields: the first is labeled 'login', the second and third are both labeled 'password'. Below these fields is a blue button with the text 'Register' in white.

Рисунок 3.2. — Регистрация.

##### 3. Страница новостей, содержащая все посты всех пользователей.



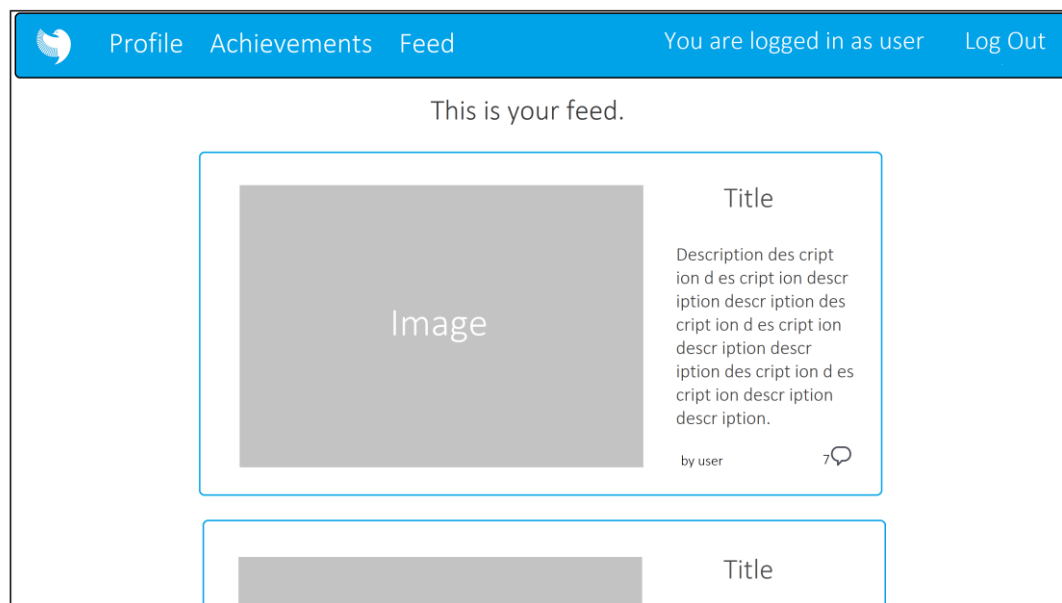


Рисунок 3.3. — Страница новостей, содержащая все посты всех пользователей.

#### 4. Создание поста.

Рисунок 3.4. — Создание поста.

#### 5. Пост, автором которого пользователь не является.

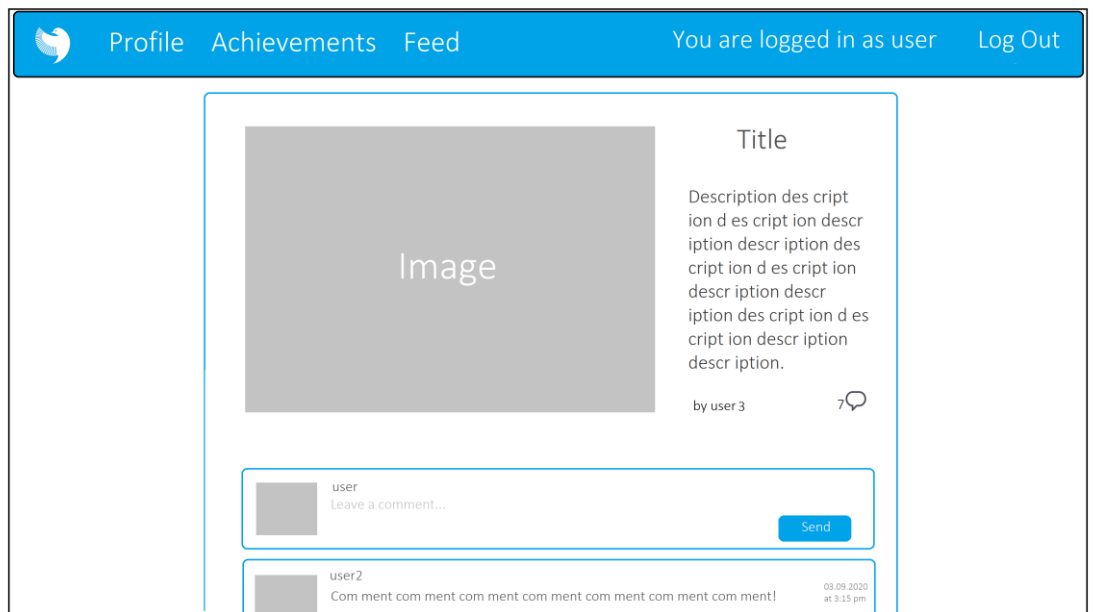


Рисунок 3.5 — Пост, автором которого пользователь не является.

## 6. Профиль пользователя.

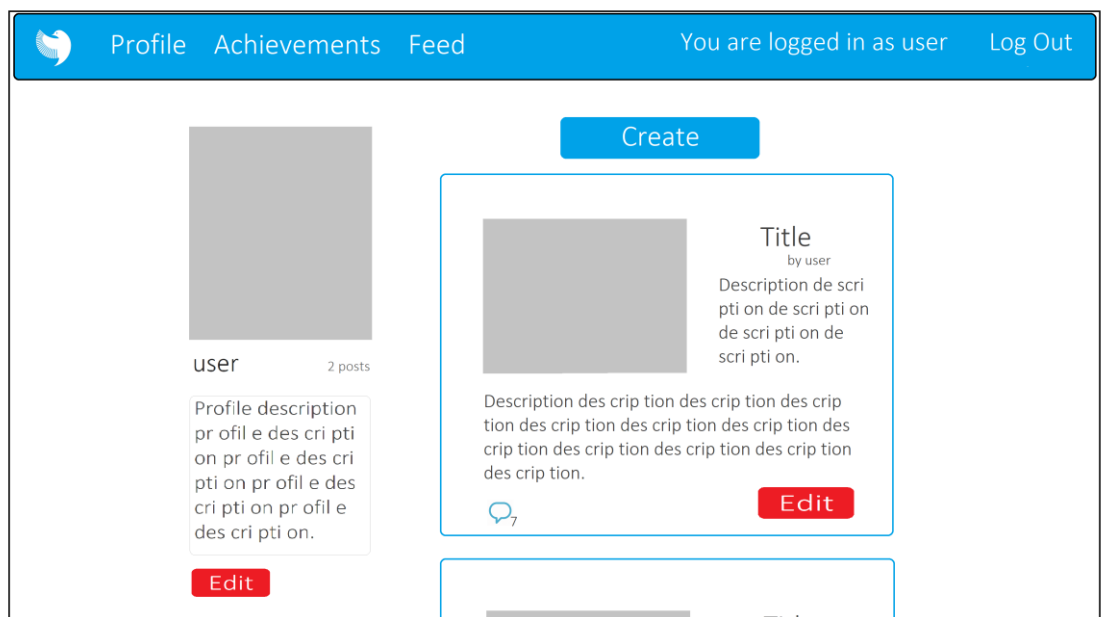


Рисунок 3.6 — Профиль пользователя.

## 7. Редактирование своего поста.

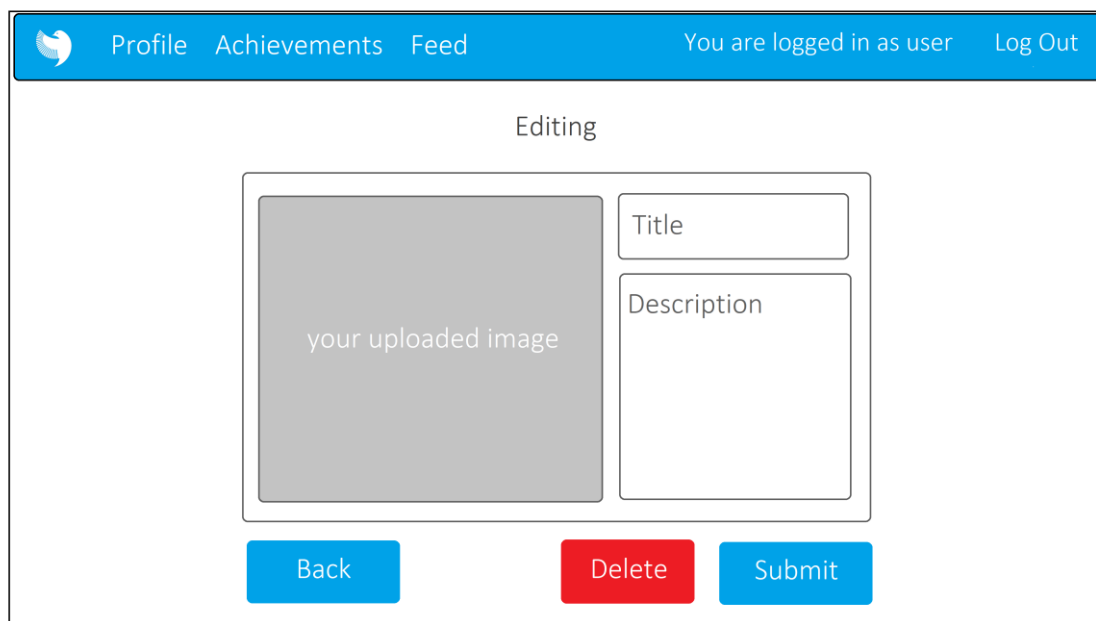


Рисунок 3.7 — Редактирование своего поста.

## 8. Профиль, не принадлежащий пользователю.

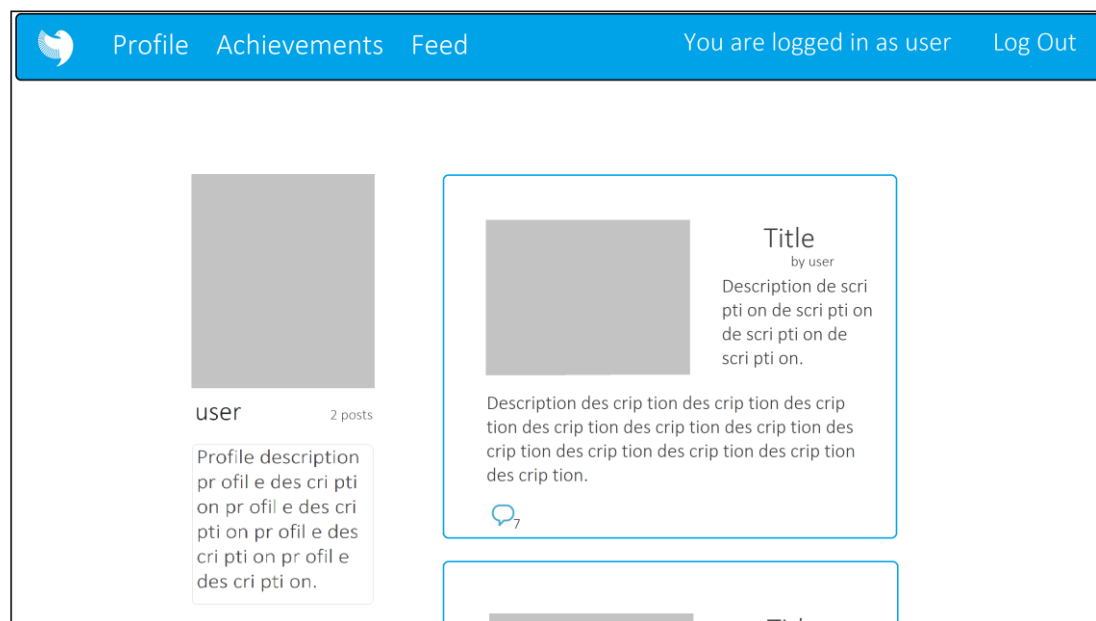


Рисунок 3.8 — Профиль, не принадлежащий пользователю.

## 9. Возможность администратора импортировать/экспортировать все данные приложения.

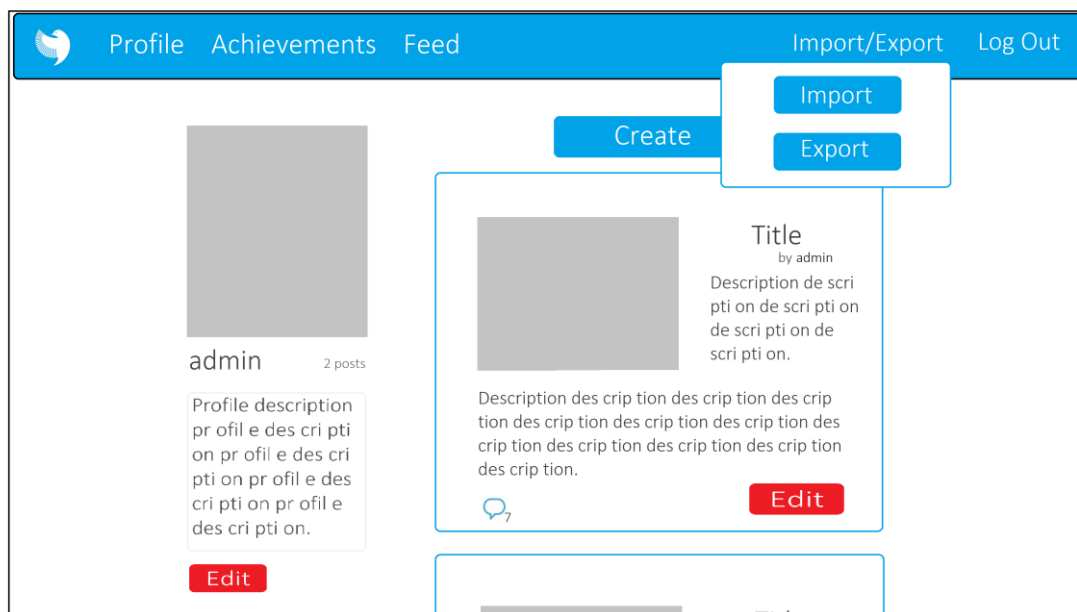


Рисунок 3.9 — Профиль, не принадлежащий пользователю.

## 10. Достижения.

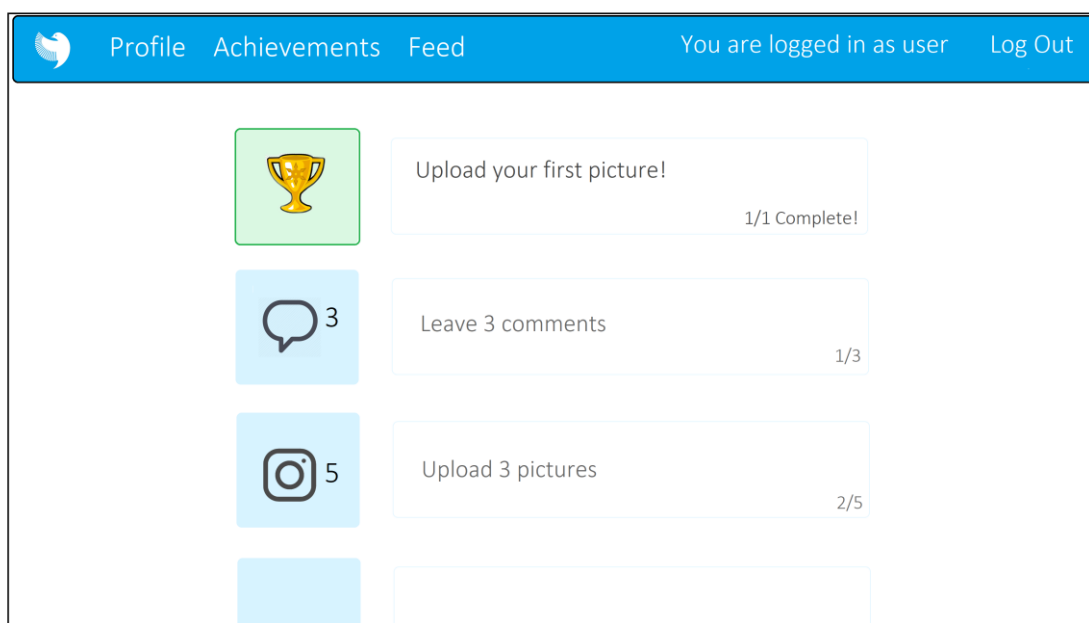


Рисунок 3.10 — Достижения.

## 3.2 Сценарии использования.

Сценарий – «Импорт-экспорт базы данных».

Действующее лицо: Админ

1. Админ заходит на сайт нашего приложения.

2. Админ входит в систему под своим логином и паролем.
3. Админ попадает на страницу Feed.
4. Админ переходит на страницу "Profile".
5. Админ нажимает на кнопку "Import/Export". Видит всплывающее меню.
6. Админ выбирает, что он хочет сделать: импортировать или экспортировать соответственно.
7. В обоих случаях открывается всплывающее окно, для выбора места экспорта или импорта файла .json.

Сценарий – «Просмотр новостной ленты BirdWatching»:

Действующее лицо: Пользователь

1. Пользователь заходит на сайт нашего приложения.
2. Пользователь входит в систему под своим логином и паролем.
3. Пользователь попадает на страницу Feed.
4. Пользователь просматривает новостную ленту.

Дополнительный сценарий – «Добавление нового поста, редактирование поста и удаление старого поста».

Действующее лицо: Пользователь

1. Пользователь заходит на сайт нашего приложения.
2. Пользователь входит в систему под своим логином и паролем.
3. Пользователь попадает на страницу Feed.
4. Пользователь попадает на страничку «Profile», нажав кнопку «Profile».
5. Пользователь может удалить любой из существующих его постов, нажав на кнопку «delete».

6. Пользователь может добавить новый пост, нажав на кнопку «Create» или исправить уже существующий пост, нажав на кнопку "Edit" у выбранного поста.

1. В случае, когда пользователь нажимает кнопку «Create», он попадает на страничку «Post».

1. Пользователь может внести название нового поста, добавить описание и картинку методом drag-and-drop, либо выбрав ее в сплывающем окне после нажатия на место картинки. После нужно сохранить пост, нажав для этого кнопку «Submit».

2. В случае, когда пользователь нажимает кнопку «Edit», он попадает на страничку «Edit-Delete Modal».

1. Пользователь может изменить что-либо из представленного: картинку, название, описание поста.

Дополнительный сценарий – «Просмотр чужого профиля и комментирование поста»:

1. Пользователь заходит на сайт нашего приложения.  
2. Пользователь входит в систему под своим логином и паролем.  
3. Пользователь попадает на страницу Feed.  
4. Просматривая ленту новостей пользователь интересуется постом другого пользователя.

5. Пользователь попадает на страничку другого пользователя, нажав на имя пользователя под постом.

6. Пользователь может прочитать уже имеющиеся комментарии и добавить новый, написав в специальном окне желаемый комментарий.

7. Для отправки комментария требуется нажать "Send" или enter.

## 4. МОДЕЛЬ ДАННЫХ

### 4.1. Схема базы данных (графическое представление нереляционной модели данных).

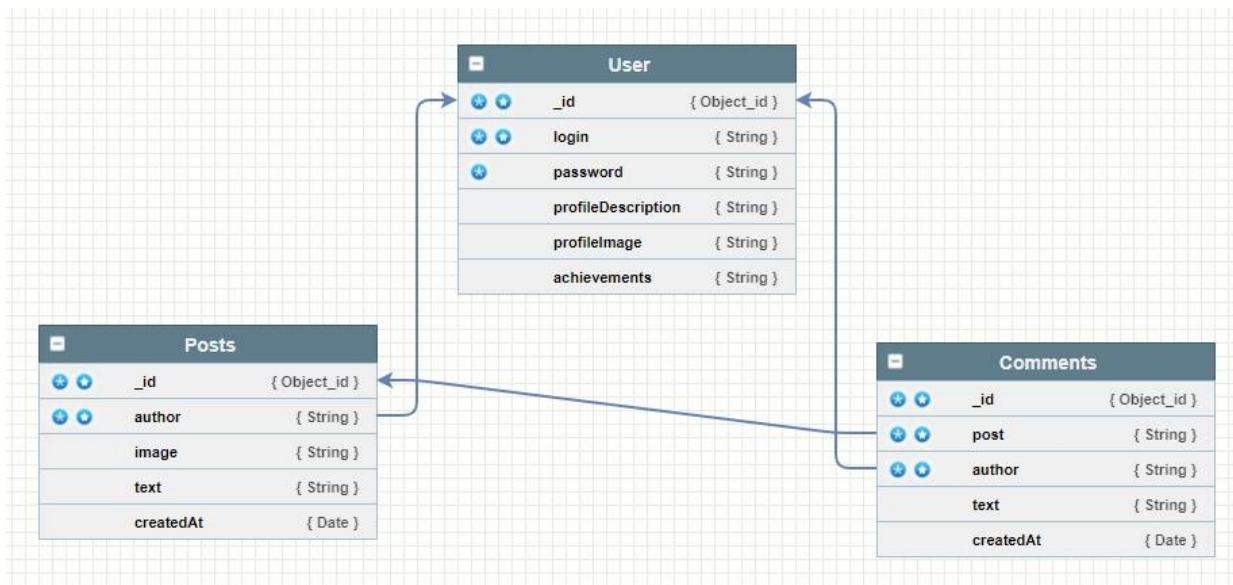


Рисунок 4.1 — Графическое представление модели БД.

### 4.2. Список сущностей модели

Разработанная модель данных включает следующие коллекции:

Users, posts, comments

### 4.3. Описание назначений коллекций, типов данных и сущностей.

Описание полей коллекции users:

Название	Тип данных	Описание
isAdmin	Boolean	Является ли данный пользователь администратором
login	String	Логин пользователя для авторизации, также является его никнеймом
password	String	Пароль пользователя
description	String	Краткая информация о себе, которую может указать пользователь

Название	Тип данных	Описание
image	String	Ссылка на картинку с аватаром пользователя

Описание полей коллекции posts:

Название	Тип данных	Описание
authorId	Object ID	id автора поста
image	String	Ссылка на картинку из поста
text	String	Текст поста
createdAt	Date	Дата и время создания поста

Описание полей коллекции comments:

Название	Тип данных	Описание
authorId	Object ID	id пользователя, оставившего комментарий
postId	Object ID	id поста, под которым оставлен комментарий
text	String	Текст комментария
createdAt	Date	Дата и время создания комментария

#### 4.4. Аналог модели данных для SQL СУБД.

Аналог модели данных для SQL СУБД совпадает с нереляционным.

#### 4.5. Оценка удельного объема информации, хранимой в модели.

Размер одного пользователя =  $12 + 1 + 1 * \text{login\_length} + 1 * \text{password\_length} + 1 * \text{profileDescription\_length} + 1 * \text{profileImage\_link\_length} =$   
 $13 + 1 * \text{login\_length} + 1 * \text{password\_length} + 1 * \text{profileDescription\_length} + 1 * \text{profileImage\_link\_length}$



Размер одного поста =  $12 + 12 + \text{image\_link\_length} + 1 + \text{text\_length} + 8 = 32 + \text{image\_link\_length} + 1 + \text{text\_length}$

Размер одного комментария =  $12 + 12 + 12 + 1 * \text{text\_length} + 8 = 44 + 1 * \text{text\_length}$

Средний размер поля image составляет 75 байт. Средний размер других полей, на который можно было бы с уверенностью опираться, определить невозможно, так как разные люди будут выбирать разные размеры паролей, описаний профиля и длины комментария или поста, а наши данные синтетические и малочисленны. Тогда:

Размер одного пользователя =  $88 + 1 * \text{login\_length} + 1 * \text{password\_length} + 1 * \text{profileDescription\_length}$

Размер одного поста =  $107 + 1 + \text{text\_length}$

Размер одного комментария =  $44 + 1 * \text{text\_length}$

Объем нашей БД не может быть статичным, но можно посчитать, от чего он будет зависеть.

На основе этого общий объем БД составляет:

$N * (88 + 1 * \text{login\_length} + 1 * \text{password\_length} + 1 * \text{profileDescription\_length}) + M * (107 + 1 + \text{text\_length}) + L * (44 + 1 * \text{text\_length})$ ,  $N, M, L \geq 0$ .

N - кол-во пользователей.

M - кол-во постов.

L - кол-во комментариев.

## 4.6. Запросы к модели, с помощью которых реализуются сценарии использования.

### Создание нового пользователя (MongoDB)

```
db.users.insertOne(
  '_id': id,
  'login': login,
  'password': password,
  'image': image,
  'description': description
)
```

### Создание нового пользователя (SQL)

```
INSERT INTO users
VALUES (login, password, image, description);
```

### Создание нового поста (MongoDB)

```
db.posts.insertOne(
  '_id': id,
  'author': author,
  'image': image,
  'text': text,
  'createdAt': date,
)
```

### Создание нового поста (SQL)

```
INSERT INTO posts
VALUES (author, image, text, createdAt);
```

### Создание нового комментария (MongoDB)

```
db.posts.insertOne(
  '_id': id,
  'author': author,
  'image': image,
  'post': post,
  'createdAt': date,
)
```

### Создание нового комментария (SQL)

```
INSERT INTO comments
VALUES (author, post, text, createdAt);
```

### Подсчет общего числа пользователей (MongoDB)

```
db.users.count()
```

### Подсчет общего числа пользователей (SQL)

```
SELECT COUNT(*) FROM users;
```

### Подсчет общего числа постов (MongoDB)

```
db.posts.count()
```

### Подсчет общего числа постов (SQL)

```
SELECT COUNT(*) FROM posts;
```

### Подсчет общего числа комментариев (MongoDB)

```
db.comments.count()
```

### Подсчет общего числа комментариев (SQL)

```
SELECT COUNT(*) FROM comments;
```

### Выборка и группировка данных для диаграммы постов (MongoDB)

```
db.posts.find(  
  createdAt: {  
    $gte: startDate  
    $lte: endDate  
  },  
  author: author  
)
```

### Выборка и группировка данных для диаграммы комментариев (MongoDB)

```
db.comments.find(  
  createdAt: {  
    $gte: startDate  
    $lte: endDate  
  },  
  author: author  
)
```

### Выборка и группировка данных для диаграммы постов и комментариев (SQL)

```
SELECT *  
FROM posts, comments  
WHERE createdAt >= startDate  
      AND createdAt <= endDate
```

### Поиск пользователя (MongoDB)

```
db.users.find(  
  {  
    login:  
    {  
      "$regex": search,  
      "$options": "i"  
    }  
  }  
)
```

### Поиск пользователя (SQL)

```
SELECT user  
FROM users  
WHERE login LIKE search
```

## 5. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

### 5.1. Схема экранов приложения

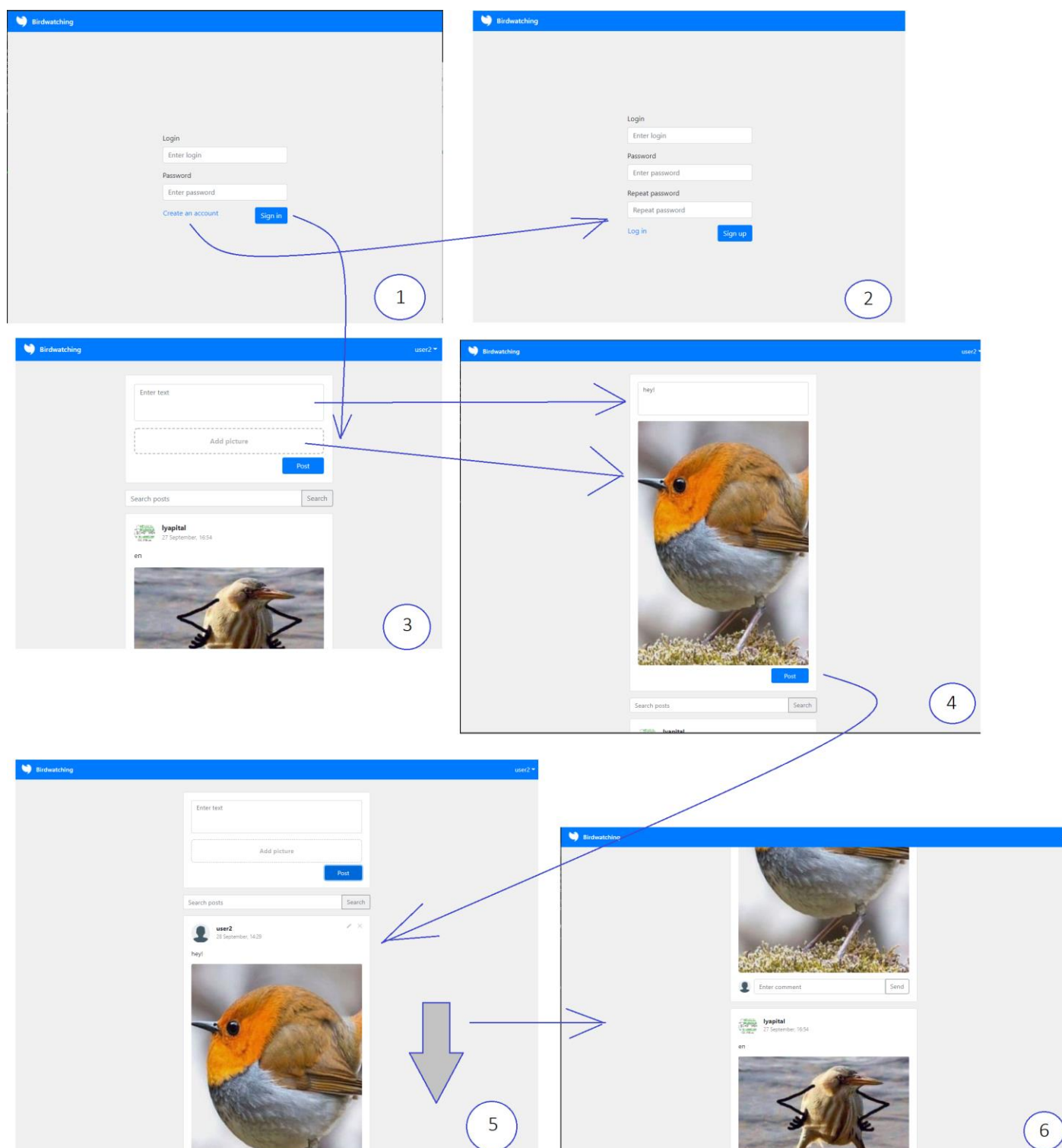


Рисунок 5.1 — Схема экранов приложения, часть 1.

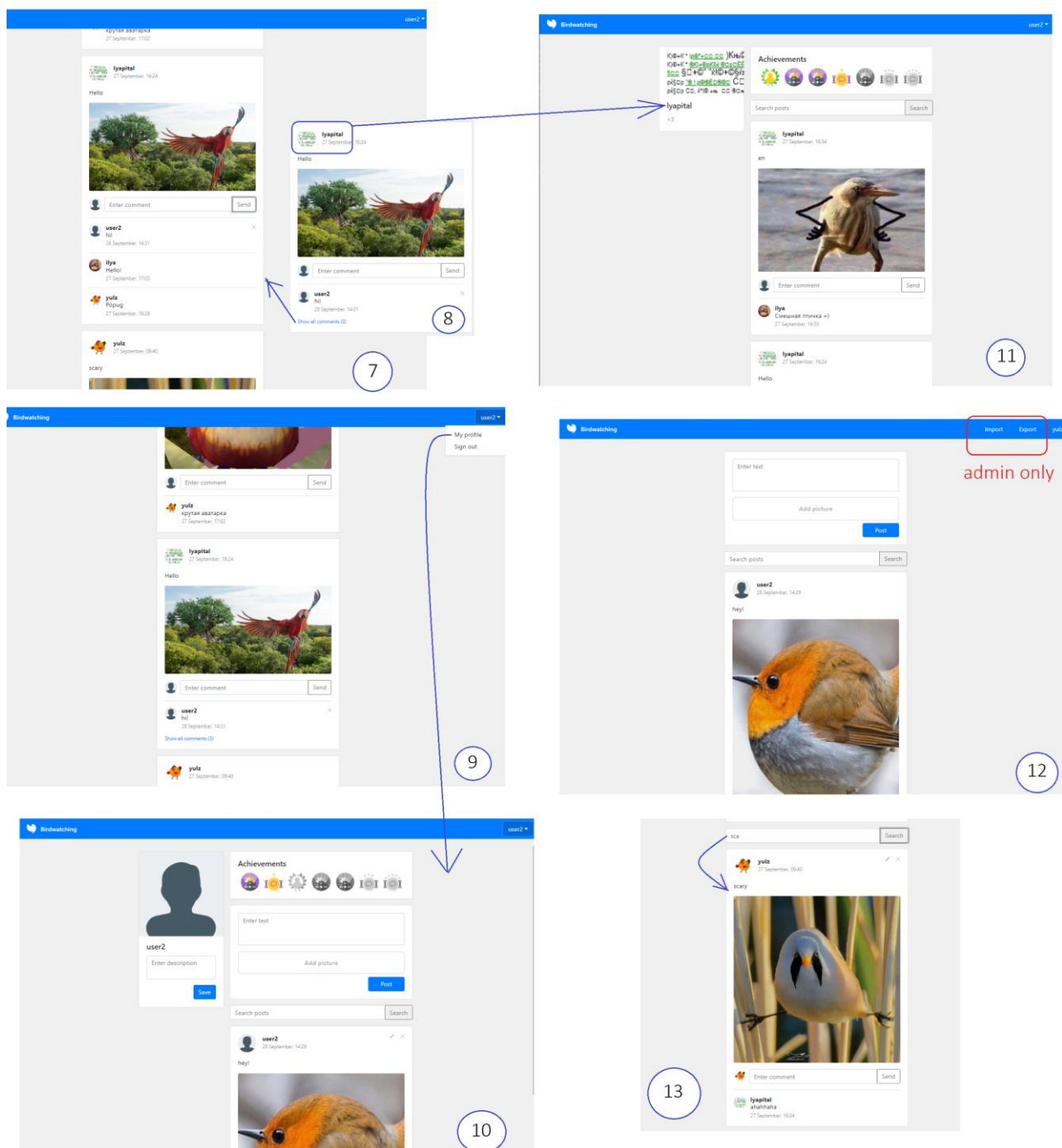


Рисунок 5.2 — Схема экранов приложения, часть 2.

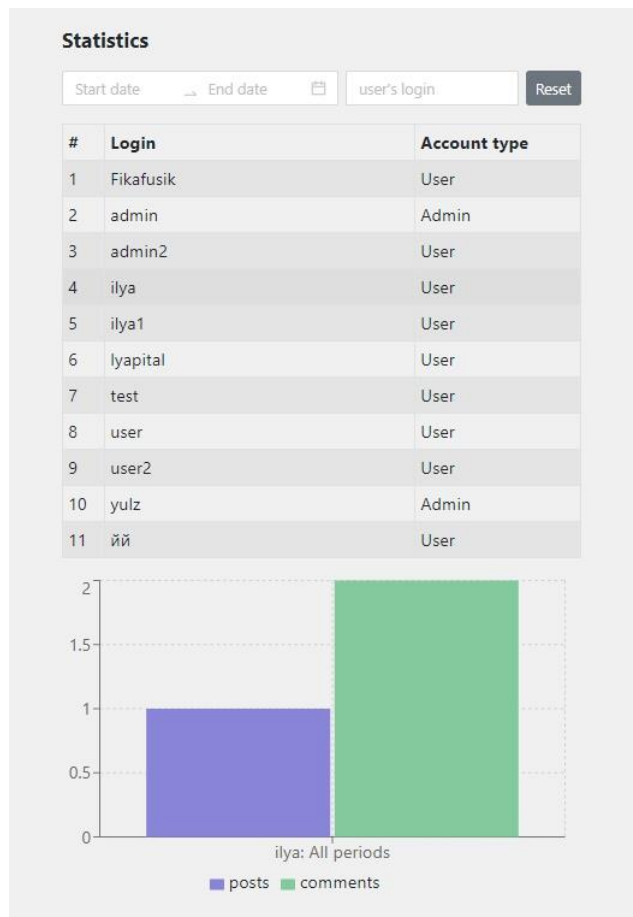


Рисунок 5.3 — Схема экранов приложения, часть 3.

## 5.2. Используемые технологии.

Node.js, Express.js, MongoDB, mongoose, react-bootstrap, redux, AWS.

## 5.3. Ссылки на приложение.

1. Github: <https://github.com/moevm/nosql2h20-bird-mongo>
2. Heroku: <https://birdwatching-frontend.herokuapp.com/>

## **6. ВЫВОДЫ**

### **6.1. Достигнутые результаты.**

Требуется разработать user-friendly приложение с использованием MongoDB. В функционал приложения должны входить ввод данных, пользователи и их профили (страницы), система достижений, комментарии, статистика.

### **6.2. Недостатки и пути для улучшения полученного решения.**

Единственным недостатком приложения является неширокий спектр возможностей. Это исправляется добавлением новых ориентированных на пользователя функций.

### **6.3. Будущее развитие решения.**

Дальнейшее развитие приложения не планируется.

## **7. ПРИЛОЖЕНИЯ**

### **7.1. Документация по сборке и развертыванию приложения.**

Инструкция для Docker.

1. Скачать репозиторий.
2. Внутри папки App открыть терминал.
3. `sudo docker-compose up --build`.



## **8. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Документация Mongoose.js — <https://mongoosejs.com/docs/api.html>.
2. Документация MongoDB — <https://docs.mongodb.com/>.