**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

по дисциплине «**Современные средства *Front-end* разработки»**

на тему: «**Разработка приложения с использованием *express.js***»

Выполнил: магистрант гр. МАГ 40-21

Вергейчик А.П.

Принял: преподаватель

Белявский Ю.В.

Гомель, 2019

**Цель работы:** научиться создавать *node.js* приложение с использованием фреймворка «*express*». Хранение информации обеспечить с помощью базы данных «*mongoDB*».

**Задание:**

1. развернуть *express* приложение;
2. спроектировать mongoose схему для статьи блога;
3. (допускается использование *Rest API* + *MVC backend design* для создания, добавления, удаления статей);
4. сделать интерфейс для добавления статей, после отправки статьи, она должна попадать в базу данных.

**Ход работы:**

Для создания *express*-приложения необходимо выполнить в консоли команду *npm install express-generator –g.* С помощью данной команды будет сгенерировано базовое *express*-приложение с необходимой архитектурой.

Для установки *passport.js* необходимо выполнить в консоли команды *npm i passport*, *npm install passport-local*. На рисунке 1 представлена архитектура приложения.

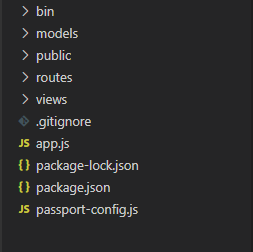


Рисунок 2 – Архитектура приложения

Данное приложение использует *MVC*-шаблон. В таблице 1 представлено описание модели *Article*, которая описывает конкретную запись.

Таблица 1 – Описание модели *Article*

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| *title* | *String* | Заголовок записи. |
| *preview* | *String* | Краткое описание записи. |
| *text* | *String* | Полный текст записи. |

В таблице 2 приведено описание полей модели User. Данная модель описывает авторизованного пользователя, который работает с приложением в текущий момент.

Таблица 2 – Описание модели *User*

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| *username* | *String* | Обязательное уникальное поле. Описывает |
| *password* | *String* | Обязательное поле, описывает пароль пользователя. |

На рисунке 3 приведен записи в *mongoDB*.

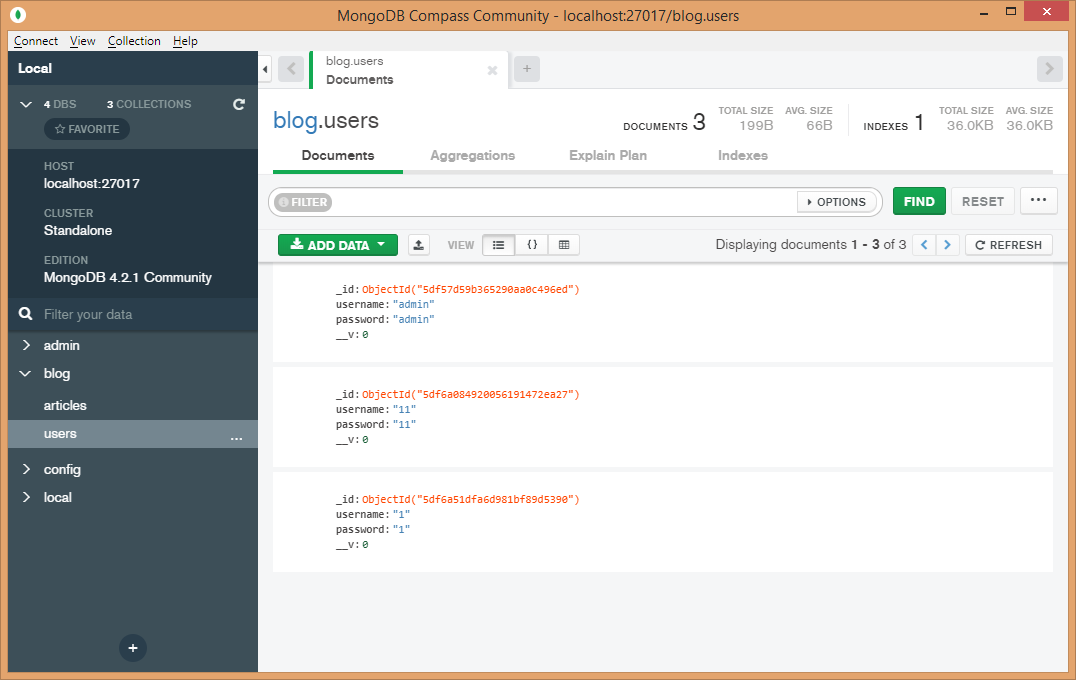


Рисунок 3 – Пример записи данных модели *User* в коллекции *Users*

Для запуска приложения необходимо вызвать c помощью команды *npm start*. На рисунке 4 приведен вид главной страницы.

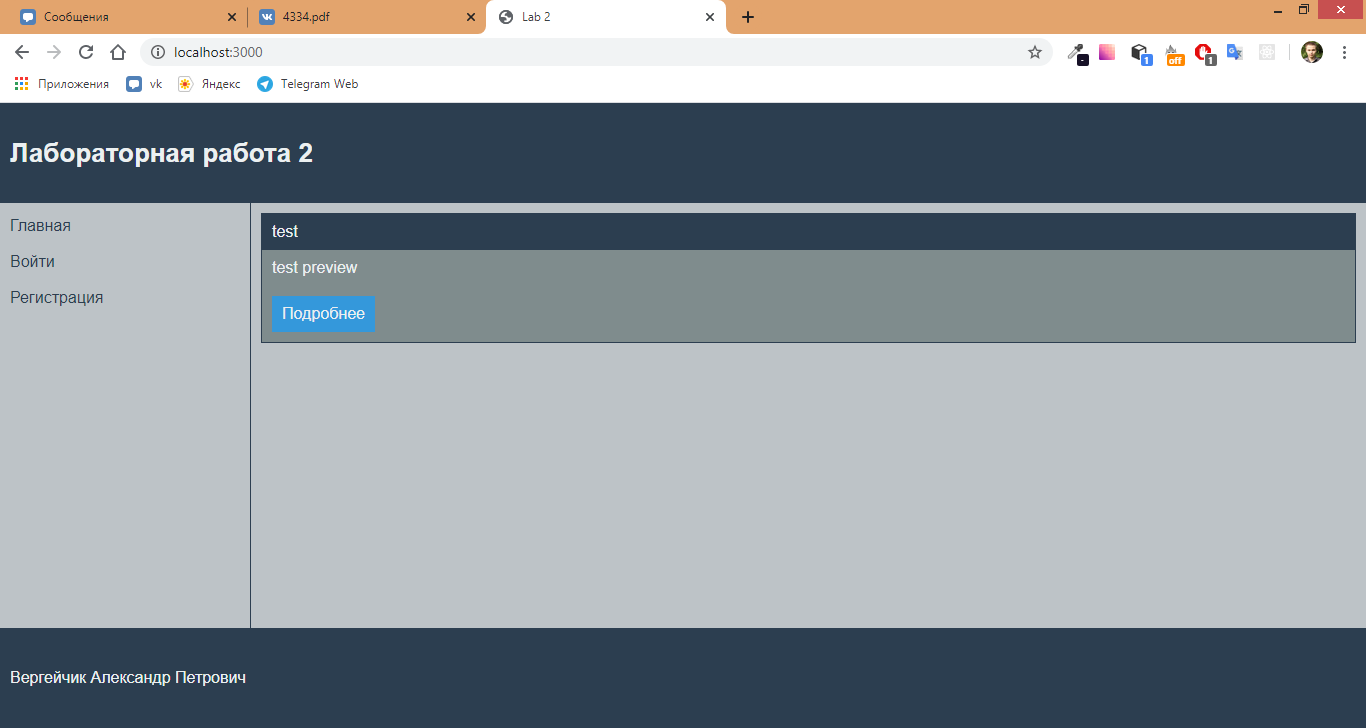


Рисунок 4 – Главная страница

Слева отображается меню, в котором можно выбрать необходимую страницу и перейти на нее. При переходе происходит проверка на права доступа с помощью *passport.js.* Если пользователь не авторизован и попытается перейти на страницу добавления записи, то он будет перенаправлен на страницу авторизации.

Данное приложение является одностраничным, все страницы подставляются с помощью языка программирования *JavaScript*.

На рисунке 5 представлена страница отдельной записи.

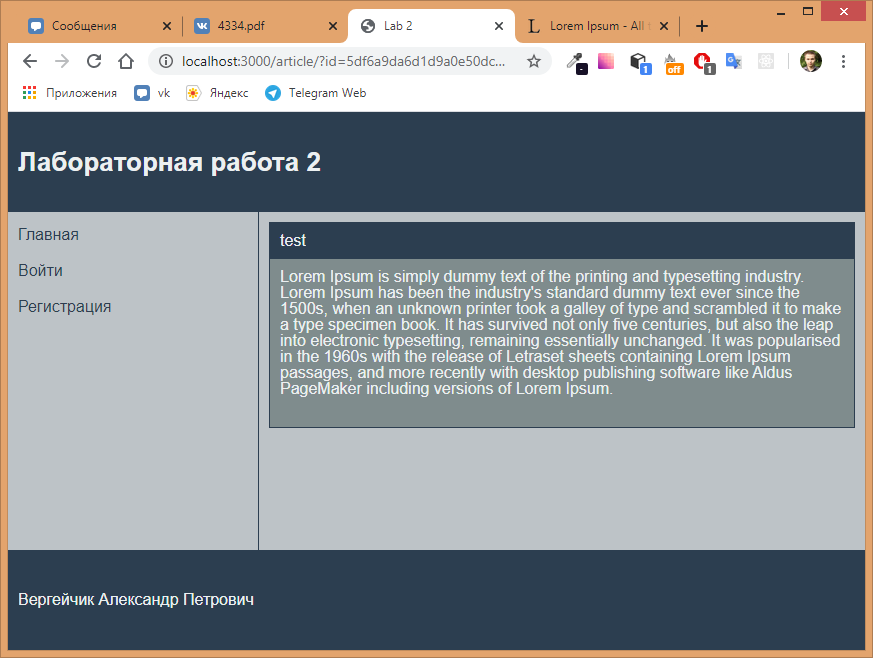


Рисунок 5 – Страница записи

При заполнении любых полей происходит их проверка на соответствие формату поля и на пустоту. На рисунке 6 показан пример валидации.

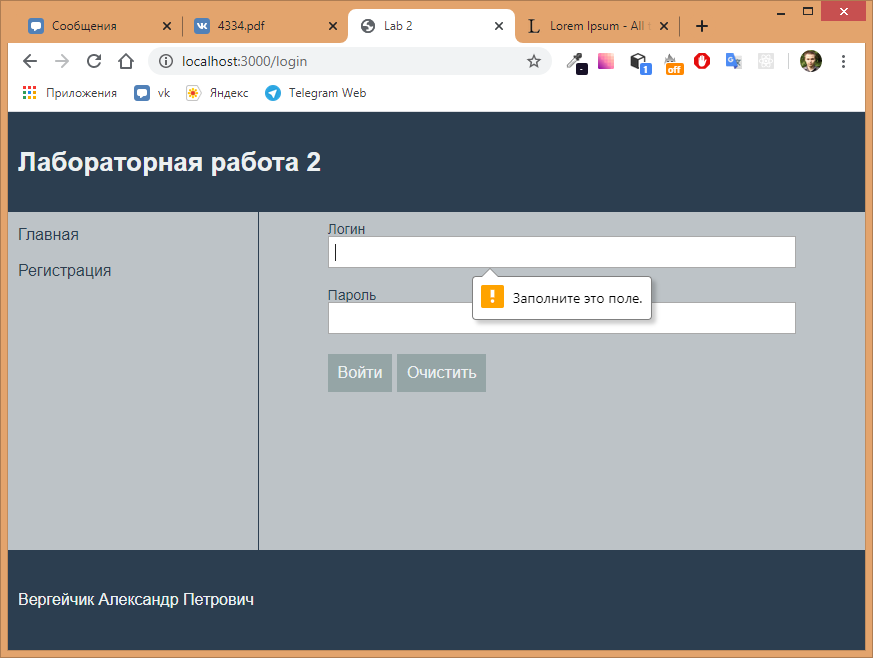


Рисунок 6 – Пример валидации формы

На рисунке 7 приведена страница добавления новой записи.

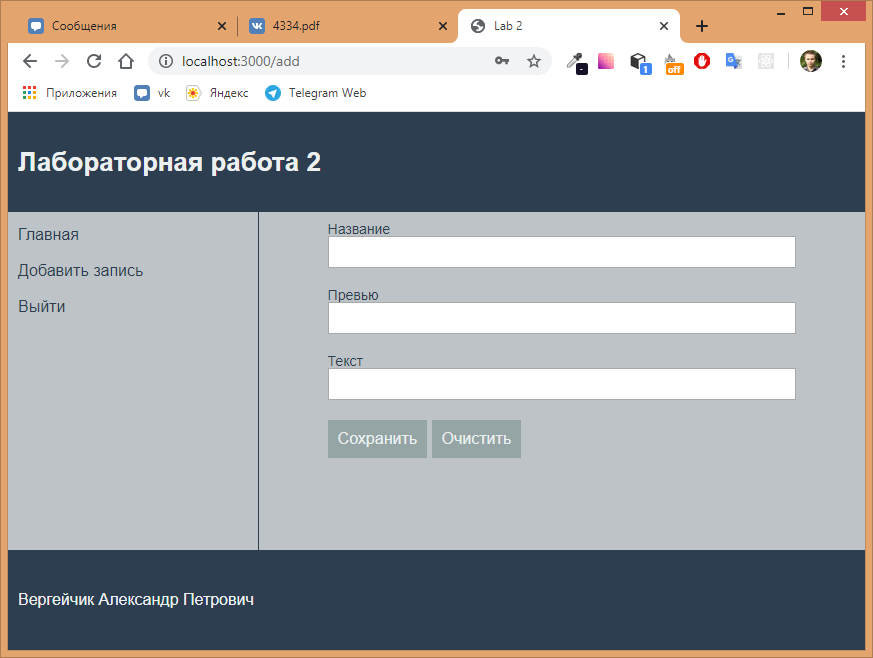


Рисунок 7 – Страница добавления записи

Для авторизации и регистрации используется библиотека *passport.js*. В качестве стратегии авторизации используется локальная стратегия. Она не зависит от каких-то сервисов и использует для авторизации и регистрации логин и пароль пользователя. Данные об авторизации хранятся в файле сессии.

**Вывод:** в ходе лабораторной работы было создано приложение с использованием *node.js* и *express.js*. В качестве базы данных используется *mongoDB*. Схема данных описывается с помощью *mongoose* схемы данных.

Архитектура приложения построена с использованием *MVC* подхода и содержит модели, которые описывают модель данных, представления – *HTML*-страницы с использованием синтаксиса препроцессора *PUG* (отвечает за вставку *JavaScript*-кода в *HTML*-страницу), и роутеров, которые содержат всю логику страницы и выступают в качестве контроллера.

**Приложение А**

(обязательное)

**Классы моделей представления данных**

**Article:**

var mongoose = require("mongoose");

var Schema = mongoose.Schema;

var articleShema = new Schema({

title: String,

preview: String,

text: String

});

var Article = mongoose.model("Article", articleShema);

module.exports = Article;

**User:**

var mongoose = require("mongoose");

var Schema = mongoose.Schema;

var UserSchema = new Schema({

username: {

type: String,

unique: true,

required: true

},

password: {

type: String,

required: true

},

});

var User = mongoose.model("User", UserSchema);

module.exports = User;

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Описание логики роутеров (контроллеров)**

**Add:**

var express = require('express');

var router = express.Router();

var Article = require('../models/Article');

/\* GET add page. \*/

router.get('/', function(req, res, next) {

res.render('add', { layout: false });

});

/\* POST add page. \*/

router.post('/', function(req, res, next) {

var title = req.body.title;

var preview = req.body.preview;

var text = req.body.text;

if(title && preview && text) {

Article.create({ title: title, preview: preview, text: text }, function(err, doc){

if(err) return console.log(err);

res.redirect('/');

});

}

});

module.exports = router;

**Article:**

var express = require('express');

var router = express.Router();

var Article = require('../models/Article');

/\* GET article page. \*/

router.get('/', function(req, res, next) {

if(req.query.id) {

Article.findById(req.query.id, function(err, doc){

res.render('article', { article: doc, isAuthenticated: req.isAuthenticated(), layout: false });

});

}

});

module.exports = router;

**Index:**

var express = require('express');

var router = express.Router();

var Article = require('../models/Article');

/\* GET home page. \*/

router.get('/', function(req, res, next) {

Article.find({}, function(err, docs){

res.render('index', { articles: docs, isAuthenticated: req.isAuthenticated(), layout: false });

});

});

module.exports = router;

**Login:**

var express = require('express');

var router = express.Router();

var passport = require('passport');

/\* GET login page. \*/

router.get('/', function(req, res, next) {

res.render('login', { layout: false });

});

/\* POST add page. \*/

router.post('/', function(req, res, next) {

passport.authenticate('local', function(err, user) {

if (!err) {

req.logIn(user, function(err){

if (!err) {

return res.redirect('/');

} else {

res.redirect('/login');

}

});

} else {

res.redirect('/login');

}

})(req, res, next);

});

module.exports = router;

**Logout:**

var express = require('express');

var router = express.Router();

/\* GET add page. \*/

router.get('/', function(req, res, next) {

req.logOut();

res.redirect('/');

});

module.exports = router;

**Registration:**

var express = require('express');

var router = express.Router();

var User = require('../models/User');

/\* GET registration page. \*/

router.get('/', function(req, res, next) {

if (!req.isAuthenticated()) {

res.render('registration', { layout: false });

} else {

res.redirect('/');

}

});

/\* POST registration page. \*/

router.post('/', function(req, res, next) {

var user = {

username: req.body.username,

password: req.body.password

}

if(user.username && user.password) {

User.findOne({ username: user.username }, function(err, response) {

if (!response) {

User.create(user, function(err) {

if (err) {

console.log(err);

res.redirect('/registration');

} else {

res.redirect('/login');

}

});

} else {

res.redirect('/registration');

}

});

}

});

module.exports = router;